Ing. Václav Jirovský, CSc.

Principy počítačů

KNIHOVNA MFF UK
2565033490

matfyzpress
VYDAVATELSTVÍ UK MFF PRAHA
2000

hybovati je lidské, ale k tomu, aby člověk skutečně něco „zvoral", potřebuje počítač.

Artur Bloch: „Murphy's Law"

Ústřední knihovna
matem. fyz. fakulty UK
odd. matematické
Sokolovská 83
186 00 PRAHA 8 - Karlín
SKM 56/2002

Všechna práva vyhrazena. Tato publikace ani žádná její část nesmí být reprodukována nebo šířena v žádné formě, elektronické nebo mechanické včetně fotokopií, bez písemného souhlasu vydavatele.

© Ing. Václav Jirovský, CSc., 2000

© MATFYZPRESS, vydavatelství Matematicko-fyzikální fakulty
Univerzity Karlovy v Praze, 2000

ISBN 80-85863-51-0

Obsah

1 ÚVOD ... 1
2 HISTORIE ... 3
2.1 ZÁZNAM ČÍSEL ... 3
2.2 PRVNÍ VYNÁLEZCI VÝPOČETNÍCH POMŮCEK ... 4
2.3 PRŮMYSLOVÁ REVOLUCE V ANGLII ... 4
2.4 STROJE NA KONCI 19. STOLETÍ ... 6
2.5 PRVNÍ POLOVINA 20. STOLETÍ ... 7
2.5.1 Konrád Zuse ... 7
2.5.2 Alan Mathison Turing ... 8
2.6 ROZMACH POČÍTAČŮ ... 10
2.6.1 John Louis von Neumann ... 11
2.6.2 Počítač Mark I ... 13
2.6.3 Počítač EDSAC ... 14
2.7 POČÍTAČOVÝ PRŮMYSL ... 15
2.8 POVÁLEČNÉ OBDOBÍ V ČECHÁCH ... 16
2.8.1 Antonín Svoboda ... 19
2.9 ROZVOJ MIKROPROCESORŮ A PERSONÁLNÍCH POČÍTAČŮ ... 20
3 ARCHITEKTURA, ORGANIZACE A IMPLEMENTACE POČÍTAČE ... 23
3.1 ARCHITEKTURA POČÍTAČE ... 23
3.2 ORGANIZACE POČÍTAČE ... 25
3.3 IMPLEMENTACE POČÍTAČE ... 28
3.4 PROGRAMOVÁNÍ ... 30
3.5 ZÁKLADNÍ PRVKY POČÍTAČE ... 31
3.6 FORMÁLNÍ POPIS POČÍTAČOVÉHO SYSTÉMU ... 31
4 REPREZENTACE DAT ... 37
4.1 LOGICKÉ HODNOTY ... 37
4.2 ZNAKY ... 38
4.3 ČÍSELNÉ SOUSTAVY ... 38
4.3.1 Polyadické číselné soustavy ... 39
4.3.2 Nepolyadické číselné soustavy ... 39
4.3.3 Konverze mezi polyadickými soustavami o různém základu ... 40
4.3.4 Chyby při konverzi ... 42
4.4 ČÍSLA V PEVNÉ ŘÁDOVÉ ČÁRCE ... 44
4.4.1 Binární čísla se znaménkem ... 45
4.4.2 Dvojkový doplněk ... 46
4.4.3 Čísla v kódu BCD ... 46
4.5 ČÍSLA V PLOVOUCÍ ŘÁDOVÉ ČÁRCE ... 47
4.5.1 Přesnost záznamu reálných čísel ... 50
4.5.2 Doporučení IEEE-754 (854) ... 51
4.5.3 Reprezentace reálných čísel v procesorech Intel ... 53
4.5.3.1 Nuly se znaménkem ... 53
4.5.3.2 Normalizované a denormalizované konečné hodnoty ... 53
4.5.3.3 Nekonečna se znaménkem ... 55
4.5.3.4 Nečíselné hodnoty ... 55
4.5.3.5 Formáty čísel ... 55
4.6 SPECIÁLNÍ TYPY ZÁZNAMU ČÍSEL ... 56

4.7 DATA S AUTOMATICKOU IDENTIFIKACÍ ... 56
4.7.1 *Tagované paměti* ... 57
4.7.2 *Deskriptory dat* ... 57
4.7.3 *Objekty* ... 58
4.8 STRATEGIE MATEMATICKÝCH OPERACÍ ... 58
4.8.1 *Operace sčítání a odčítání* ... 58
4.8.2 *Operace násobení a dělení* ... 59
5 INSTRUKCE ... 61
5.1 INSTRUKČNÍ REPERTOÁR ... 62
5.1.1 *Variace operačních kódů* ... 62
5.1.2 *Variace operandů* ... 63
5.1.3 *Modalita polí operandu* ... 63
5.1.4 *Modality ochrany paměti* ... 63
5.2 OPERANDY INSTRUKCÍ ... 64
5.2.1 *Čtyřadresové instrukce* ... 64
5.2.2 *Tříadresové instrukce* ... 64
5.2.3 *Dvouadresové instrukce* ... 65
5.2.4 *Jednoadresové instrukce* ... 65
5.2.5 *Bezadresové počítače* ... 65
5.3 SKUPINY INSTRUKCÍ ... 66
5.3.1 *Instrukce přesunu dat* ... 66
5.3.2 *Skupina aritmeticko-logických instrukcí* ... 66
5.3.3 *Skupina instrukcí posuvu a rotace* ... 67
5.3.4 *Skupina skokových instrukcí* ... 68
5.3.5 *Instrukce pro operace se zásobníkem* ... 68
5.3.6 *Instrukce řízení stroje* ... 69
5.3.7 *Skupina instrukcí vstupu a výstupu* ... 69
5.4 INSTRUKČNÍ REPERTOÁR - VELIKOST, SLOŽITOST A ÚPLNOST ... 70
5.5 ZPŮSOBY ADRESOVÁNÍ ... 71
5.5.1 *Přímé adresování* ... 71
5.5.2 *Nepřímé adresování* ... 73
5.5.3 *Relativní adresování* ... 73
5.5.4 *Indexované adresování* ... 75
6 PROCESORY ... 77
6.1 ŘADIČ ... 79
6.2 FORMÁTY MIKROINSTRUKCÍ ... 83
6.3 NANOPROGRAMOVÁNÍ ... 84
6.4 VIRTUÁLNÍ PROCESOR ... 86
6.5 OPTIMALIZACE PRÁCE PROCESORU ... 88
6.5.1 *Procesory s proudovým zpracováním* ... 88
6.5.2 *Vektorové procesory* ... 90
6.5.3 *Superskalární procesory* ... 93
6.5.4 *Přerovnání instrukcí* ... 94
6.5.5 *Optimalizace instrukcí skoku* ... 96
6.5.6 *Instrukce vázané daty* ... 98
6.6 DALŠÍ ZVYŠOVÁNÍ VÝKONNOSTI PROCESORŮ ... 98
6.6.1 *Koncepce architektury RISC* ... 99
6.6.2 *Taxonomie procesorů RISC* ... 102
6.6.3 *Přehled vlastností komerčně dodávaných procesorů* ... 103
6.6.3.1 Procesory typu SPARC ... 104
6.6.3.2 Procesor IBM - RS/6000 ... 104
6.6.3.3 Procesor Intel 860 ... 105
6.6.4 *Architektury Post-RISC* ... 106
6.6.4.1 Jednotka predekódování a vyrovnávací paměti instrukcí (1) ... 107

6.6.4.2 Predikční jednotka (Fetch/Flow) (2) ... 108
6.6.4.3 Dekódování a skoky (Decode/Branch) (3) ... 108
6.6.4.4 Přidělování instrukcí a změna uspořádání (4) ... 109
6.6.4.5 Exekuční jednotky (5) ... 109
6.6.4.6 Odkládací jednotka a vyrovnávací paměť pro ukončené instrukce (6) ... 110
6.6.4.7 Proudová linka zpracování instrukcí u procesorů třídy post-RISC ... 111
6.6.5 *Třída post-RISC jako procesor řízený daty* ... 111
6.6.6 *Příklady vybraných procesorů* ... 112
6.6.6.1 DEC Alpha 21 164 (21 264) ... 112
6.6.6.2 IBM PowerPC 604 ... 113
6.6.6.3 Intel Pentium P6 ... 115
7 PAMĚTI ... 117
7.1 CHARAKTERISTIKY PAMĚTI ... 117
7.1.1 *Rozlišení pamětí podle přístupu k datům* ... 118
7.1.2 *Rozlišení pamětí podle implementace* ... 119
7.1.3 *Rozlišení pamětí podle umístění v systému* ... 121
7.2 VNITŘNÍ ORGANIZACE PAMĚŤOVÝCH JEDNOTEK ... 122
7.2.1 *Paměť se širokým slovem* ... 123
7.2.2 *Paměť s prokládanými cykly* ... 123
7.2.2.1 Techniky redukce konfliktů ... 124
7.2.3 *Kontrola a opravy chyb dat v pamětech* ... 126
7.3 HIERARCHICKÉ PAMĚŤOVÉ SYSTÉMY ... 128
7.3.1 *Mapovací mechanismy* ... 129
7.3.2 *Dvojúrovňový paměťový systém* ... 130
7.3.2.1 Stránkovaná paměť ... 130
7.3.2.2 Segmentovaná paměť ... 131
7.3.2.3 Stránkovaná segmentová paměť ... 131
7.3.3 *Efektivní přístupová doba* ... 132
7.3.4 *Příklady realizace virtuálního adresování* ... 133
7.3.4.1 Sun SPARC ... 133
7.3.4.2 Intel Pentium ... 135
7.3.5 *Struktura programu a virtuální paměťový systém* ... 137
7.3.6 *Strategie výměny bloku* ... 138
7.3.6.1 Strategie FIFO ... 139
7.3.6.2 Strategie LRU ... 140
7.3.7 *Tříúrovňový paměťový systém* ... 141
7.3.7.1 Vyrovnávací paměť s přímým mapováním ... 143
7.3.7.2 Vyrovnávací paměť s asociativním mapováním ... 143
7.3.7.3 Vyrovnávací paměť s částečně asociativním mapováním ... 144
7.3.8 *Zapisovací strategie* ... 145
7.3.8.1 Strategie současného zápisu ... 146
7.3.8.2 Strategie zpožděného zápisu ... 146
7.3.8.3 Strategie předběžného zápisu ... 147
7.3.8.4 Porovnání strategií zápisu ... 147
7.3.9 *Redukce ztrát výkonu v hierarchických paměťových systémech* ... 147
7.3.9.1 Snížení počtu výpadků bloku ... 147
7.3.9.2 Redukce ztrát při výpadku bloku ... 148
7.3.9.3 Redukce času potřebného k nalezení shody ... 149
7.3.10 *Virtuální a fyzické vyrovnávací paměti* ... 149
8 PROPOJOVACÍ SUBSYSTÉMY ... 151
8.1 SMĚROVÉ ORGANIZACE ... 152
8.1.1 *Základní pojmy* ... 152
8.1.2 *Statické sítě* ... 153
8.1.2.1 Lineární síť ... 153
8.1.2.2 Stromové sítě a hvězda ... 153
8.1.2.3 Kruhová síť ... 154
8.1.2.4 Planární topologie ... 155
8.1.2.5 Hyperkrychle ... 155
8.1.2.6 Komunikační strategie ve statických sítích ... 155
8.1.3 *Dynamické sítě* ... 156
8.1.3.1 Křížové přepínače ... 156
8.1.3.2 Propojovací sítě s výměnou ... 157

8.1.3.3 Víceúrovňové přepínací sítě ... 158
8.1.3.4 Strategie řízení dynamické sítě ... 159
8.2 SBĚRNICOVÉ ORGANIZACE ... 160
8.2.1 Sběrnicové hierarchie ... 161
8.2.2 Synchronní a asynchronní přenos ... 162
8.2.3 Arbitráž požadavků o sběrnici ... 164
8.2.4 Příklady sběrnic v komerčních systémech ... 166
8.2.4.1 Paralelní sběrnice AGP ... 166
8.2.4.2 Sériová sběrnice USB ... 167
8.3 ZPŮSOBY PROGRAMOVÉ OBSLUHY PROPOJENÍ ... 167
8.3.1 Programová obsluha ... 168
8.3.2 Obsluha s využitím přerušení ... 168
8.3.3 Blokové přenosy - DMA ... 169
8.3.3.1 Přímý přístup s odpojením procesoru ... 170
8.3.3.2 Přímý přístup s "kradením" cyklů ... 171
8.3.3.3 Multiplexovaný přístup ... 171
8.3.4 Samostatné procesory ... 171
9 TAXONOMIE VÝPOČETNÍCH SYSTÉMŮ ... 173
9.1 FLYNNOVA TAXONOMIE ... 173
9.2 PROCESORY TYPU SISD ... 173
9.3 PROCESORY TYPU SIMD ... 174
9.4 PROCESORY MISD ... 174
9.5 PROCESORY MIMD ... 175
9.6 DALŠÍ POUŽÍVANÉ TAXONOMIE ... 175
10 POČÍTAČOVÉ SYSTÉMY S VYSOKÝM VÝKONEM ... 177
10.1 PŘEHLED ORGANIZACÍ S VYSOKÝM VÝKONEM ... 178
10.1.1 Multiprocesory MIMD se sdílenou pamětí ... 179
10.1.2 Multiprocesorové vektorové stroje ... 180
10.1.3 Systémy MIMD s distribuovanými pamětmi ... 181
10.1.4 Stroje SIMD ... 182
10.1.5 Maticové počítače ... 183
10.1.6 Distribuované paměti ... 184
10.1.7 Porovnání systémů SIMD a MIMD ... 186
10.2 PARALELNÍ SYSTÉMY ... 186
10.2.1 Podmínky pro paralelní provádění výpočtu ... 186
10.2.2 Granularita ... 187
10.2.3 Způsoby paralelního zpracování ... 187
10.2.4 Výkon paralelního systému ... 188
10.2.5 Účinnost paralelního zpracování ... 192
10.2.6 Účinnost paralelních algoritmů ... 193
11 NETRADIČNÍ STROJE ... 195
11.1 ASOCIATIVNÍ PROCESORY ... 195
11.2 POČÍTAČE ŘÍZENÉ UDÁLOSTMI ... 196
11.2.1 Počítač řízený tokem dat ... 196
11.2.1.1 Jednoúčelové počítače řízené daty ... 197
11.2.1.2 Statické a dynamické univerzální počítače ... 198
11.2.2 Aplikační charakteristiky počítačů řízených tokem dat ... 198
11.2.3 Redukční počítače ... 199
11.3 SYSTOLICKÁ POLE ... 199
11.4 NEURONOVÉ SÍTĚ ... 200
12 PERSPEKTIVY DALŠÍHO TECHNOLOGICKÉHO ROZVOJE ... 203
12.1 MECHANICKÉ NANOPOČÍTAČE ... 205

12.2 CHEMICKÉ NANOPOČÍTAČE ...
12.2.1 Biologické systémy ... 207
12.3 KVANTOVÉ NANOPOČÍTAČE ... 207
12.4 ELEKTRONICKÉ NANOPOČÍTAČE ... 210
13 MĚŘENÍ VÝKONNOSTI ... 213
14 SPOLEHLIVOST POČÍTAČOVÝCH SYSTÉMŮ ... 217
14.1 ZÁKLADNÍ POJMY ... 217
14.1.1 Spolehlivost R(t) ... 217
14.1.2 Pravděpodobnost poruchy Q(t) ... 218
14.1.3 Hustota rozložení poruch q(t) ... 218
14.1.4 Střední doba bezporuchového provozu - MTBF ... 218
14.2 HODNOCENÍ SPOLEHLIVOSTI ... 219
14.3 ZÁLOHOVÁNÍ ... 220
14.4 SPOLEHLIVOST PROGRAMOVÉHO VYBAVENÍ ... 221
15 PŘÍLOHY ... 225
15.1 HISTORICKÝ PŘEHLED VÝZNAMNÝCH TYPŮ POČÍTAČŮ DO ROKU 1978 ... 226
15.2 KÓDOVACÍ TABULKA EBCDIC ... 228
15.3 KÓDOVACÍ TABULKA ASCII ... 229
15.4 KÓDOVACÍ TABULKA LATIN 2 ... 230
15.5 SEZNAM NEJDŮLEŽITĚJŠÍCH ANGLICKÝCH ZKRATEK Z VÝPOČETNÍ TECHNIKY ... 232
16 LITERATURA ... 237

Seznam vyobrazení

Obr. 1 Čínský Abakus ... 3
Obr. 2 Blaise Pascal ... 4
Obr. 3 Část repliky Analytical Engine ... 5
Obr. 4 Jednoduchý Turingův stroj ... 9
Obr. 5 Počítač ENIAC ... 10
Obr. 6 Struktura počítače dle von Neumanna ... 11
Obr. 7 Část registru elektronkového počítače ... 15
Obr. 8 Počítač IBM360 – centrální jednotka a pult operátora ... 16
Obr. 9 První československý počítač SAPO ... 17
Obr. 10 Procesorová jednotka počítače Altair ... 20
Obr. 11 První počítač IBM PC ... 20
Obr. 12 Nárůst výkonnosti mikroprocesorů firmy Intel ... 21
Obr. 13 Velikosti základních datových struktur ... 24
Obr. 14 Příklad datové cesty šířky 32 bitů ... 26
Obr. 15 Vývoj počtu transistorů v mikroprocesorech Intel ... 29
Obr. 16 Schéma struktury počítače se vstupy a výstupy prostřednictvím akumulátoru ... 32
Obr. 17 Modifikovaná struktura počítače s jedním rychlý a jedním pomalým kanálem vstupu a výstupu ... 33
Obr. 18 Struktura počítače s vícenásobným přístupem do paměti ... 34
Obr. 19 Zjednodušený PMS diagram počítače Apple ... 35
Obr. 20 Průběh chyby vzniklé zavedením stupnice ... 42
Obr. 21 Průběh chyby vzniklé zaokrouhlením nebo odseknutím ... 43
Obr. 22 Konstrukce váhového binárního kódu ... 44
Obr. 23 Nejčastěji používané formáty záznamu čísel v pevné řádové čárce ... 44
Obr. 24 Způsob uložení čísla v kódu BCD bez znaménka ... 46
Obr. 25 Přímý záznam čísla v plovoucí řádové čárce ... 47
Obr. 26 Nejčastěji používané záznamy čísla v plovoucí řádové čárce ... 48
Obr. 27 Množina binárních reálných čísel - velikost mantisy určuje interval mezi dvěma možnými reprezentacemi ... 50
Obr. 28 Rozložení reprezentace reálných čísel podle číselné osy ... 53
Obr. 29 Zobrazení reálných nul ... 53
Obr. 30 Normalizované a denormalizované číselné hodnoty ... 53
Obr. 31 Reprezentace nekonečna se znaménkem ... 54
Obr. 32 Reprezentace NaN, tichého a signálního ... 54
Obr. 33 Formáty zobrazení reálných čísel v procesorech Intel ... 55
Obr. 34 Reprezentace komplexního číslo ... 56
Obr. 35 Zlomková reprezentace čísla ... 56
Obr. 36 Buňka tagované paměti ... 57
Obr. 37 Blok deskriptorů s paměťovými odkazy ... 57
Obr. 38 Přímé adresování registrů ... 71
Obr. 39 Přímé adresování paměti, THERE=93 ... 71
Obr. 40 Nepřímé adresování obsahem registru ... 71
Obr. 41 Nepřímé adresování obsahem paměti, THERE=93 ... 72
Obr. 42 Varianty nepřímého adresování registrem ... 73
Obr. 43 Relativní adresování vzhledem k bázové adrese ... 73
Obr. 44 Relativní adresování vzhledem k programovému čítači ... 74
Obr. 45 Relativní adresování s hodnotou v operandu ... 74
Obr. 46 Logický součet realizovaný kontakty relé ... 77
Obr. 47 Realizace poloscítacího obvodu z logických hradel ... 78
Obr. 48 Realizace klopného obvodu RS z kombinačních hradel NAND ... 78
Obr. 49 Generování impulsů pro časování procesoru ... 79
Obr. 50 Direktivní řadič a jeho stavový diagram ... 80
Obr. 51 Obecné schéma zpětnovazebního řadiče ... 80
Obr. 52 Jednoadresový procesor ... 81
Obr. 53 Obecné schéma mikroprogramového řadiče ... 82
Obr. 54 Wilkinsovo schéma mikrořadiče ... 82
Obr. 55 Obsahy pamětí mikroinstrukce při horizontálním a vertikálním mikroprogramování ... 83
Obr. 56 Velikost paměti mikroprogramu a nanoprogramu ... 84
Obr. 57 Organizace virtuálního procesoru ... 85
Obr. 58 Gantovy diagramy pro hypotetickou sčítačku s proudovým zpracováním ... 88
Obr. 59 Řetězení vektorů ... 91
Obr. 60 Stavový diagram přechodů pro dvoubitovou predikci skoků. ... 95
Obr. 61 Příklad instrukčních formátů procesoru třídy RISC ... 99
Obr. 62 Parametry procesoru SPARC ... 102
Obr. 63 Parametry procesoru RS/6000 ... 103
Obr. 64 Parametry procesoru Intel 80860 ... 104
Obr. 65 Obecná struktura procesoru třídy post-RISC ... 105
Obr. 66 Typická položka v předsunuté paměti instrukcí (I-cache). ... 106
Obr. 67 Přiřazení příznaků k jednotlivým instrukcím ... 106
Obr. 68 Pořadí provádění instrukcí ... 107
Obr. 69 Postup odkládání instrukcí ... 108
Obr. 70 Linka proudového zpracování v procesorech třídy post-RISC ... 110
Obr. 71 Vnitřní struktura procesoru DEC Alpha 21164 ... 111
Obr. 72 Bloková vnitřní struktura procesoru PowerPC 604 ... 112
Obr. 73 Třícestný model procesoru Pentium ... 113
Obr. 74 Uspřádání čipů v pouzdře procesoru Pentium ... 113
Obr. 75 Vnitřní bloková struktura procesoru Intel Pentium P6 ... 114
Obr. 76 Model paměťové jednotky počítače ... 117
Obr. 77 Hierarchie pamětí v paměťovém systému ... 121
Obr. 78 Vztah ceny za uloženou jednotku informace a rychlosti přístupu k informaci ... 122
Obr. 79 PMS diagram paměti s prokládanými cykly pro čtyři bloky ... 123
Obr. 80 Snímek četnosti přístupů do paměti ... 128
Obr. 81 Hierarchické uspořádání paměťového systému ... 128
Obr. 82 Příklad mechanismu výměny paměťových oblastí (stránek) v hierarchickém systému ... 129
Obr. 83 Dvojúrovňový hierarchický paměťový systém ... 130
Obr. 84 Mapování adres s použitím tabulky stránek ... 131
Obr. 85 Mapovací mechanismus pro segmentovanou stránkovanou paměť ... 132
Obr. 86 Blokové schéma systému s reálnou MMU ... 134
Obr. 87 Tříúrovňové mapování adresových odkazů v překladových tabulkách v architektuře SPARC ... 134
Obr. 88 Segmentace a stránkování v procesoru Intel Pentium ... 136
Obr. 89 Grafické znázornění změn v hlavní paměti. ... 137
Obr. 90 Příklad chování programu při přístupu ke stránkám paměti ... 137
Obr. 91 Průběh funkce výměny stránek k diagramu v textu ... 141
Obr. 92 Tříúrovňový paměťový systém ... 142
Obr. 93 Blokové schéma vyrovnávací paměti s přímým mapováním ... 143
Obr. 94 Vyrovnávací paměť s plně asociativním mapováním ... 144
Obr. 95 Vyrovnávací paměť s částečnou asociativitou stupně 2 ... 145
Obr. 96 Hierarchické propojení počítačového systému ... 151
Obr. 97 Základní topologie sítí ... 152
Obr. 98 Stromová a hvězdicová topologie ... 154
Obr. 99 Propojovací sítě s kruhovou topologií ... 154
Obr. 100 Příklady planárních sítí ... 155
Obr. 101 Postup konstrukce čtyřdimenzionální krychle (hyperkrychle) ... 156
Obr. 102 Křížový přepínač ... 157
Obr. 103 Elementární přepínač 2 x 2. ... 157
Obr. 104 Konfigurace sítě s dokonalým promícháním ... 158
Obr. 105 Konfigurace sítě s dokonalým oddělením ... 158
Obr. 106 Příklad sítě typu Omega pro n=8 ... 159
Obr. 107 Jednoduchý sběrnicový systém propojující jednotky J1 až Jn ... 160
Obr. 108 Dvouúrovňový hierarchický systém sběrnic ... 161
Obr. 109 Hierarchický systém sběrnic personálního počítače ... 162
Obr. 110 Časový diagram práce synchronní sběrnice při čtení a zápisu ... 163
Obr. 111 Čtení na synchronní sběrnici s vkládáním čekacích cyklů ... 163
Obr. 112 Řízení asynchronní sběrnice - jednostranné ... 164
Obr. 113 Řízení asynchronní sběrnice - oboustranné ... 164
Obr. 114 Centrální řízení s výzvami ... 165
Obr. 115 Centrální vyzývání s nezávislými vodiči ... 165
Obr. 116 Přidělování prioritního přístupu na sběrnici - metoda daisy chain ... 165
Obr. 117 Princip získání adresy vektoru přerušení ... 168
Obr. 118 Zapojení řadiče sběrnice a procesoru pro přímý přístup do paměti ... 169
Obr. 119 Přímý přístup do paměti s odpojením procesoru – mikroprocesor MC6800 ... 170
Obr. 120 Přímý přístup do paměti – metoda kradení cyklů ... 170
Obr. 121 Přímý přístup do paměti - multiplexace ... 171
Obr. 122 PMS diagram stroje typu SISD ... 173

Obr. 123 PMS diagram stroje typu SIMD 174
Obr. 124 PMS diagram stroje typu MISD 175
Obr. 125 PMS diagram stroje typu MIMD 175
Obr. 126 PMS diagram systému c.mmp 179
Obr. 127 Organizace systému KSR-1 180
Obr. 128 Profily komunikace ve struktuře "fat-tree" 181
Obr. 129 PMS diagram propojovací sítě stroje CM-5 181
Obr. 130 PMS diagram uzlů stroje CM-5 182
Obr. 131 Procesor počítače CM-2 183
Obr. 132 Organizace maticového počítače s lokálními pamětmi 183
Obr. 133 Organizace maticového procesoru se sdílenou pamětí 184
Obr. 134 PMS diagram paralelního procesor s distribuovanou pamětí 185
Obr. 135 Redukční paralelismus 188
Obr. 136 Paralelismus datových toků, schéma provádění výpočtu 188
Obr. 137 Grafické vyjádření Amdahlova zákona 191
Obr. 138 Struktura počítače STARAN 196
Obr. 139 Struktura statického počítače řízeného tokem dat 199
Obr. 140 Lineární systolická síť 200
Obr. 141 Model formálního neuronu a jeho přechodová funkce 201
Obr. 142 Schema Hopfieldovy sítě 202
Obr. 143 Vývoj směřující k nanotechnologiím v porovnání s Moorovým zákonem 203
Obr. 144 Postup manipulace s jednotlivými atomy pomocí "tunelové mikroskopie". 204
Obr. 145 Vývoj nanomechanických a nanoelektronických technologií vzhledem k jejich uplatnění na trhu. 205
Obr. 146 Princip mechanického logického hradla 206
Obr. 147 Ekvivalence elektronické a biologické logiky 207
Obr. 148 Vznik interferencí po průchodu světla štěrbinou 209
Obr. 149 Průběh jednoho maxima vodivosti rezonanční tunelové diody 211
Obr. 150 Princip atomového relé 212
Obr. 151 Spojení prvků systému z hlediska spolehlivosti 219
Obr. 152 Vanovitá křivka průběhu intenzity poruch po dobu životnosti programu 222

1 Úvod

Předmět "Principy počítačů" byl koncipován jako přehledový, s cílem zpracovat základní oblasti počítačové architektury, organizace a implementace digitálních počítačů tak, aby budoucí absolventi oboru informatika mohli tyto znalosti použít při optimalizaci navrhovaných programů a pro snadnější výběr cílového stroje. Tento úkol, který si před sebe klade předkládaná publikace, je nelehký. Problém spočívá nejenom v překotném vývoji oboru, a z toho plynoucího nezměrného rozsahu mnohdy protichůdných informací, ale i v poměrně jednostranném pohledu na počítač, který si přinášejí noví studenti ze středních škol. Obtížnou orientaci v oboru ještě komplikují nejrůznější "odborné" a marketingové informace obchodních firem, jejichž nepřesnost je spíše projevem profesionální nevyzrálosti či nedostatečné teoretické a odborné přípravy, než úmyslem.

Rozsah problematiky a důraz na přehlednost základních pojmů a koncepcí mne vedly k záměru soustředit se zejména na objasnění významu termínů, základní charakteristiky příslušných oblastí a popis typických řešení. Uvádění detailních aplikací, spolu s potřebným teoretickým základem, by vedlo nejenom k neúnosnému objemu publikace, ale s ohledem na rozvoj technologií i k neaktuálnosti publikace v okamžiku jejího vydání. Z toho vyplývá i pojetí předmětu, které se soustřeďuje na základní architektonické a organizační principy a zcela úmyslně potlačuje technologické implementační hledisko, kde je v současnosti soustředěn největší nápor vývojových změn. Vytvoření širokého přehledu v těchto oblastech výstavby počítače by se mělo stát základnou pro navazující předměty v rámci specializace. Detailní informace o novinkách v oboru najdou zájemci jak v odborném tisku, tak i na mnoha internetových serverech. Na tomto místě bych chtěl znovu varovat čtenáře - při získávání informací z publikací spíše marketingového nebo obchodního charakteru, kterých je na internetových serverech většina, je nutno je vyhodnocovat velmi pozorně a odpovědně, neboť jejich úroveň mnohdy neodpovídá profesionálním potřebám.

Celá publikace je rozdělena do čtrnácti částí. První z nich se zabývá v krátkém přehledu historií a nejvýznačnějšími postavami oboru. Teprve při bližším seznámení s dějinami výpočetní techniky si uvědomíme, kolik idejí existovalo již před staletími a dodnes, i při stávajícím stavu techniky, jsou obtížně realizovatelné. Následující kapitoly se zabývají postupně základy reprezentace dat a formováním instrukčního kódu, což umožní lepší pochopení principu práce procesorů, jež jsou probírány dále. Nezbytná součást každého počítače – paměť je rozebrána v samostatné kapitole. Pojednání o výstavbě stroje je zakončeno popisem propojovacích systémů s příklady typických konstrukcí. Navazující kapitoly podávají přehled o složitějších počítačových systémech, netradičních strojích a perspektivách dalšího rozvoje. Na závěr jsou uvedeny základní pojmy z oblasti měření výkonnosti počítačových systémů a spolehlivosti jak z hlediska technického, tak i programového vybavení..

Na tomto místě bych chtěl vyjádřit své poděkování recenzentům, Prof. Ing. Jiřímu Bílovi, DrSc. a Mgr. Jakubu Yaghobovi, jejichž připomínky byly cenným přínosem pro zpřesnění a úpravu probírané látky. Doufám, že toto skriptum bude užitečnou pomůckou nejenom pro studenty informatiky matematicko-fyzikální fakulty Univerzity Karlovy pro které je určeno, ale i pro ty, kteří se o danou problematiku hlouběji zajímají.

Praha, prosinec 1999

Václav Jirovský

Historie se neopakuje, opakují se pouze sami historici.
A. Bloch:"Murphy's Law"

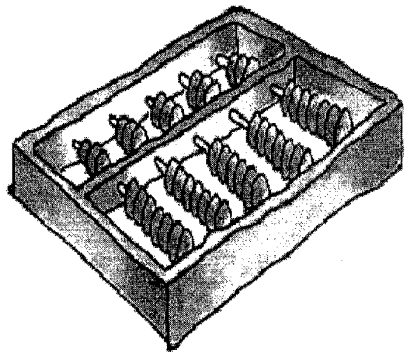
2 Historie

Začátky moderní vědy mohou být vysledovány až do šerého dávnověku, do doby kdy lidé žili v jeskyních nebo v lesích, a kmenová organizace byla základní ochranou proti rozmarům přírody a okolního prostředí. Převážná většina těchto kmenů přijímala živelnou náklonnost k vysvětlování okolních jevů pomocí nadpřirozených sil a tedy i příklon k primitivní víře. V každém kmeni, nejnižší organizované společnosti, byl vždy někdo, kdo se staral o dodržování náboženských obřadů, kdo komunikoval s Bohy jménem kmene. Takový jedinec, kouzelník nebo šaman, určoval období náboženských obřadů a musel tedy být schopen základní **orientace v čase** - počítání dnů nebo období, a **v prostoru** - počítání délek nebo kusů. Tak se zrodily první operace s kvantitativními hodnotami a první **primitivní záznam čísel** - zářezy na holi nebo čárky na stěně.

Při dlouhodobém počítání pomocí čárek nebo zářezů se objevovaly chyby. Rovněž kouzelníci se snažili si svoji práci ulehčit. Objevují se první primitivní pomůcky pro počítání času - kalendáře. Takovým primitivním prehistorickým kalendářem je např. Stonehenge (cca 13 km na sever od Salisbury, Anglie), který byl vybudován přibližně 2800 let před n.l. Nezávisle na tom, že dnes neumíme vysvětlit jakým způsobem byly tyto monumentální stavby realizovány, jde o doklad odvěké touhy člověka počítat a měřit sebe i přírodu kolem.

2.1 Záznam čísel

Čína, jedna z nejrozvinutějších civilizací v Asii, aktivně rozvíjela obchod se svými sousedy - Japonskem, Indií a Koreou. Obchodníci potřebovali vést přesné a objemné účetnictví, na které nestačilo pouhé ukazování na prstech a jednoduché početní úkony. Tak se zrodil „Abakus", první známý prostředek výpočetní techniky. Princip Abakusu na Obr. 1 byl jednoduchý:



Obr. 1 Čínský Abakus

Hodnota přiřazená každé „kuličce" je určena nikoliv jejím tvarem, ale její polohou. Jedna kulička v řádce má hodnotu 1, dvě kuličky v téže řádce dávají hodnotu 2 atd. Kulička na další řádce má váhu vyšší o jeden řád - to znamená, že jedna kulička značí 10 (předpokládáme desítkovou soustavu), dvě kuličky 20, podobně jako v předcházejí řádce s nižším řádem. Každá následující řádka má o jednotku vyšší řád než předcházející. Abakus pracuje tedy na principu „hodnota-poloha", umístění kuličky reprezentuje její hodnotu. Kuličky jsou počítány jejich přesouváním doprava, hodnoty jsou mazány přesouvání kuliček doleva. Abakus tedy byl první mechanickou pomůckou, „externí pamětí" a „aritmeticko-logickou jednotkou", pro osoby provádějící numerické výpočty „zpaměti". Ke skutečným mechanickým kalkulačním strojům však zbývala ještě dlouhá cesta. Ani evropská kultura nezahálela, římské kalkulační pomůcky se od čínského abakusu svým principem příliš nelišily.

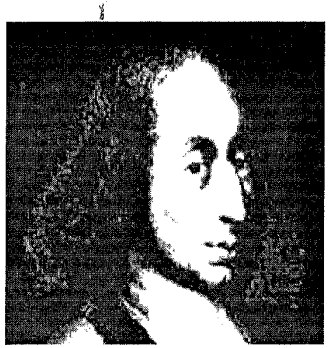
Desítková čísla, která dnes používáme, vznikla v Indii a do Evropy je „přinesli" Arabové zhruba kolem roku 1000. Římské znaky, převzaté ze znaků abecedy, se používaly pro záznam čísel jak vzestupně (např. 9 = VIIII) tak i sestupně (např. 9 = IX ... od 17. století). Přesto, že nejběžnější číselnou soustavou je desítková, udržela se i šedesátková (i když s použitím číslic desítkové soustavy pro záznam) a v souvislosti s římským záznamem číslic se používala ve středověké západní Evropě i dvacítková (např. 80 = IIII^{XX}). Soustava arabských číslic je soustavou polyadickou na rozdíl od římského způsobu, který může být příkladem nepolyadické[1] soustavy.

---

[1] Viz kap. 4.3.2 na straně 41.

2.2 První vynálezci výpočetních pomůcek

Po dobu téměř tisíce let od doby, co číňané sestrojili abakus, nedošlo k podstatnému pokroku v oblasti automatizace výpočtů. Řekové přišli s novými matematickými teorémy, formulovali nové oblasti této exaktní vědy, ale stále jediným prostředkem provádění výpočtů byla "ruční práce". Matematik té doby byl člověk, sedící v zadních prostorách instituce a vybaven psacími potřebami, spolu s ostatními skloněn nad řešením zadaného početního úkonu. Jeho kolega, řešící stejný problém, byl zárukou, že nalezené řešení nebo výsledek budou správné[1]. Trvalo týdny nebo měsíce, než se prokázala správnost ověřovaného teorému nebo se dospělo ke konečnému výsledku ze shromážděných dat. Převážná většina tabulek logaritmů a trigonometrických funkcí byla zpracována tímto postupem a teprve s nástupem kalkulačních strojů schopných zpracovat obdobné výpočty v mnohem kratším čase byly hodnoty uvedené v tabulkách ověřovány.



Obr. 2 Blaise Pascal

Byl to pravděpodobně Blaise Pascal[2], významný myslitel, vědec a matematik kdo první sestrojil mechanický kalkulační stroj. V roce 1642 předvedl první sčítací stroj pracující na principu určování vzdálenosti ujeté vozem, který byl popsán Herem Alexandrijským[3] již v druhém století př. n.l. Jeho mechanický kalkulátor pojmenovaný "Pascalina" používal sadu ozubených kol, přístupných pomocí kliček a upravených tak, že při desetinásobném otočení jednoho kola se jednou otočilo kolo odpovídající vyššímu řádu.

Pascalina měla mnoho nevýhod; mezi největší patřila ta, že jediný, kdo ji uměl opravit byl sám Blaise Pascal. Uvedení tohoto přístroje ve známost spustilo první vlnu "technofobie", vlna pomluv a nenávisti se zdvihla od jeho kolegů - matematiků, kteří se cítili novým strojem ohroženi. Po Pascalině, která uměla jen vytvářet kumulativní součty, bylo navrženo a vyrobeno ještě mnoho dalších mechanických strojů, i dokonalejších, které uměly násobit a dělit. Pascalina však byla první.

2.3 Průmyslová revoluce v Anglii

Anglie na počátku devatenáctého století byla svědkem bouřlivého rozmachu přírodních věd. Je založena British Association for Advancement of Science[4]. Zejména celoevropský pokrok v oblasti matematiky[5] vede řadu vědců a inženýrů k zamyšlení nad možnostmi realizace únavných ručních výpočtů prací mechanického přístroje. V době kdy se pokračovatelé Blaise Pascala snažili o zdokonalení jeho Pascaliny, absolvoval univerzitu v Cambridge mladík, jehož myšlenky v oblasti výstavby počítačů předstihly svoji dobu o téměř sto let. Charles Babbage se narodil v roce 1791 v hrabství Devonshire v rodině bankovního úředníka. Babbage byl na svoji viktoriánskou dobu velmi nekonvenční – miloval železnici, tunely i žaludeční výplachové pumpy, studoval biblické zázraky i odolnost lidského těla vůči vysokým teplotám. Nenáviděl pouliční hudebníky. Svými

[1] Jednalo se o jednoduchý způsob zajištění spolehlivosti práce "aritmeticko-logické jednotky" dodáním redundantních "modulů".

[2] Blaise Pascal (19. 6. 1623 – 19. 8. 1662) francouzský matematik, fyzik a filosof. Jeho hlavní příspěvky v matematice směřovaly do oblasti výpočtů binomických koeficientů a teorie pravděpodobnosti. Z fyziky je znám Pascalův zákon o rovnoměrném šíření tlaku v kapalinách.

[3] Hero Alexandrijský žil okolo roku 150 př. n.l. a byl uznávaným řeckým mechanikem, geometrem a zeměměřičem, autorem četných spisů o mechanice, automatice a válečných strojích.

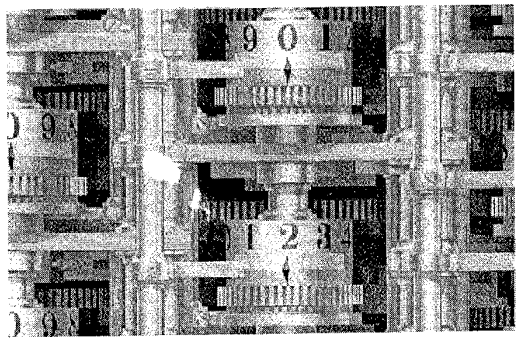
[4] Založena v roce 1831 podle obdobných společností vznikajících v ostatních zemích.

[5] Uveďme jen pro ilustraci – v roce 1823 formuluje Cauchy limitní definici integrálu, v roce 1826 Lobačevskij zveřejňuje svoji neeuklidovskou geometrii a rok 1828 je významným mezníkem v teorii eliptických funkcí.

neobvyklými projevy a zvyky byl mnohdy pro smích svým sousedům, kteří pohoršení pouličními hudebníky, stáli před jeho domem v Dorset Street vyhrávali co nejhlasitěji.

Fascinovali ho data a čísla, snažil se kvantifikovat i nemožné. Ověřování jeho teorie matematické předpovědi dostihových výsledků stály mnoho peněz jeho dlouholetou přítelkyni lady Lovelace. Měřil vše co bylo v dosahu – tep srdce prasete, průměrné množství dřeva pokácené dřevorubcem za 10 hodin i procentní poměr pohlaví u jednotlivých druhů domácí drůbeže. Mezi jeho obdivuhodné publikace patří článek "Relativní četnost příčin rozbití výkladních skříní"[1], ve kterém uvedl 464 případů ve kterých pouze čtrnáct bylo způsobeno "opilými muži ženami nebo dětmi".

Změna britské společnosti z pastorální v letech 1820 na až "brutálně materialistickou" v letech 1840 až 1850 se silně projevila i v myšlení Charlese Babbage. Doba páry, nadšení pro expandující industrializaci a manufaktury přivedla Babbage k vědeckému studiu pracovního procesu. Jeho podrobné analýzy pracovního procesu, kde položil základ operačnímu výzkumu, shrnul roku 1832 v knize "On the Economy of Manufactures". Jednotlivé pasáže z tohoto díla byly citovány předními filosofy a ekonomy ještě dlouhá léta.



Obr. 3 Část repliky Analytical Engine

Při své posedlosti čísly Charles Babbage zjistil dnes již zřejmou skutečnost, že mnoho dlouhých výpočtů se sestává z operací, které se opakují. Teoreticky prokázal, že musí být možné zkonstruovat kalkulační stroj, který by tyto operace prováděl automaticky a z vlastních prostředků sestavil v roce 1822 prototyp stroje. Následovně se obrátil dopisem, ve kterém popsal svůj plán na výstavbu stroje pro výpočty a tisk matematických tabulek, na předsedu Královské společnosti se žádostí o finanční podporu při jeho konečné realizaci. Když se v roce 1823 sešel s pokladníkem společnosti a obdržel potřebné finance, ihned se pustil do stavby stroje, který vešel ve známost jako **"Difference Engine"**. Ten měl být poháněn parním strojem, pracovat zcela automaticky, včetně tisku výstupních sestav a měl být řízen pevným programem. Konečná verze stroje měla zaujímat plochu dnešního fotbalového hřiště. V roce 1828, když celý grant 1500 liber od britské vlády byl spotřebován, financuje další vývoj Charles Babbage z vlastních prostředků. Skupina Babbageových přátel přesvědčila tehdejšího předsedu britské vlády hraběte Wellingtona, aby se seznámil s projektem a v prosinci 1829 Babbage získal dodatečný grant 3000 liber. Svoji výjimečnosti si Charles Babbage získal řadu přátel, ale i ohnivých nepřátel. Reverend Richard Sheepshanks byl jeden z nejzavilejších a jeho stostránkový spis "Letter to the Board of Visitors of Greenwich Royal Observatory, in Reply to the Calumnies of Mr. Babbage" byl argumentem k odmítnutí další podpory realizace Difference Engine v roce 1854. I když stroj nebyl nikdy dokončen, mnoho z jeho myšlenek se použilo při vývoji prvního moderního počítače.

Jedním z důvodů, mimo nedostatek financování, pro které ukončil Charles Babbage stavbu "Difference Engine" bylo i to, že jej napadla myšlenka na stavbu lepšího stroje - **"Analytical Engine"**. Tento analytický stroj měl počítat v decimální soustavě s ciframi o až padesáti místech a měl být schopen si zapamatovat až 1000 takových čísel. Myšlenku řízení stroje převzal z tkalcovského stavu vytvořeného v roce 1801 francouzským tkalcem a později průmyslníkem *Josephem Maria Jacquardem* (7.7. 1752 - 7.8. 1834). Ten použil pro řízení stavu při tkaní složitých vzorů Vaucansonovu myšlenku papírové šablony[2] a upravil ji do tvaru děrných štítků. Instrukce a data pro Babbageův "Analytical Engine" byly uloženy právě na děrných štítcích. Přelomem v tomto programovém přístupu byla řada zabudovaných operací – instrukcí, mezi které patřila i instrukce **podmíněného skoku.**

[1] Uveřejněno v Mechanics Magazine, 1857.

[2] *Jacques de Vaucanson* - francouzský mechanik a vynálezce 24.2 1709 - 21.11. 1782

Historie připisuje autorství instrukce podmíněného skoku nikoliv Babbageovi, ale jeho dlouholeté přítelkyni lady Lovelace, dívčím jménem Ada Byron. Dcera básníka britského romantismu lorda Byrona byla pět týdnů po svém narození na žádost své matky lady Byron separována od svého otce a plně svěřena do její výchovy. Ta ji z obavy, aby z dcery nevyrostla básnířka, podporovala ve studiu matematiky a přírodních věd. Na podzim roku 1841 Charles Babbage odjel přednášet do Turina a synopse jeho přednášek byla publikována ve francouzštině. Ada Adelida de Lovelac byla, pravděpodobně prostřednictvím Mary Somervillové[1], požádána o překlad těchto sylabů. Když jej zasílala Charlesu Babbageovi, doplnila překlad svými vlastními odbornými poznámkami, které byly více než třikrát delší než vlastní článek. To byl začátek jejich přátelství. Lady Lovelac byla osobou, která inspirovala Babbage k napsání prvního počítačové programu – výpočtu Bernoulliho koeficientů, její poznámky, myšlenky a nápady, doplněné a realizované Babbagem, předešly dobu o více než sto let. Největší svoji roli však sehrála jako popularizátorka myšlenek Charlese Babbage. Patřila mezi několik jedinců, kteří do hloubky pochopili myšlenku Analytical Engine a snažila se ji populárně presentovat vlivným osobnostem ve svém okolí, jejichž matematické zázemí bylo téměř zanedbatelné. Její krátký demonstrační program a úvahy o možném použití stroje byly velkým přínosem k popularizaci Babbageových myšlenek, avšak celá její snaha přišla pro Charlese Babbage pozdě. Ada Adelaide de Lovelac, zemřela ve svých 36 letech, zanechajíc po sobě myšlenkový odkaz velkého dosahu. Na její počest byl jejím jménem nazván programovací jazyk Ada, vyvinutý v roce 1979 americkým ministerstvem obrany.

Stejně jako první - Difference Engine – nebyl dokončen ani Analytical Engine. V roce 1851 se Charles Babbage vzdává veškerých nadějí, i když se ještě následující rok pokouší přesvědčit o své myšlence ministra Disraeliho. Odrazem jeho zklamání je sžíravá "Exposition of 1851": "Thus bad names are coined by worse men to destroy honest people, as madness of innocent dogs arises from the cry of insanity raised their villainous pursuers."[2]. Zklamán společností, odchází do ústraní svého "opevněného" domu na Dorset Street. I když až do roku 1868 stále publikuje, v roce 1864 "Passages from Life of the Philosopher", již to není ten Babbage - filosof, politik, vášnivý podporovatel industrializace, misantrop. Zapomenut, umírá v listopadu 1871. Na severním pólu měsíce je kráter. Jeho jméno je Charles Babbage.

2.4 Stroje na konci 19. století

Řízení pomocí otvorů nebo výstupků, signalizující přítomnost nebo nepřítomnost příslušného příkazu se přeneslo do řady elementárních mechanických automatů - hrací strojky, klavifony, hrací skříně. Princip zůstal stejný i když jeho provedení se různilo - jednou to byly otvory v pásce, podruhé výstupky na bubnu atd.

První vykročení směrem k automatizovaným výpočtům s použitím binárního záznamu bylo úspěšné uvedení děrných štítků do spojitosti s počítacímu stroji v roce 1890 ve Spojených státech. Americký inženýr Herman Hollerith, který pracoval pro americký statistický úřad, byl otcem myšlenky použít děrné štítky pro zpracování údajů při sčítání obyvatelstva. Navrhl stroj, nazývaný tabulátor, který třídil děrné štítky podle kódu zaznamenaného ve formě otvorů v příslušných posicích. Na děrných štítcích byly zaznamenány údaje o obyvatelích Spojených států podle odpovědí ano nebo ne v příslušných dotaznících. Realizace jeho myšlenky vedla k výraznému snížení chyb při zpracování výsledků sčítání obyvatelstva v roce 1890 a podstatně zvýšila jeho rychlosti. Navíc, balík děrných štítků byl pamětí s téměř neomezenou kapacitou a různé úlohy mohly být rozděleny do různých "balíků" štítků a zpracovány znovu podle potřeby. Na druhou stranu, Hollerithův stroj byl značně omezený, děrné štítky mohly být použity pouze pro sumační výpočty a složitější postupy stále zůstávaly doménou matematiků - počtářů. Kuriozitou jeho řešení je, že myšlenka použití děrných štítků nepocházela ze znalosti díla Charlese Babbage, ale připadl na ni při pozorování průvodčího štípajícího lístky ve vlaku.

Úspěch tabulátorů při sčítání obyvatelstva ve Spojených státech vedl k jejich obrovskému rozšíření. Jeho tvůrce založil vlastní firmu pro výrobu a prodej počítacích strojů. Firma se stala známou pod názvem *I*nternational *B*usiness *M*achines.

[1] Mary Fairfax Somerville – skotská matematička, populární ve vědeckých kruzích viktoriánské Anglie.

[2] "Tyto pomluvy jsou šířeny špatnými lidmi, jen aby byly zhoubou pro ty čestné, stejně jako pomatení nevinných psů vzniká z řevu jejich ničemných pronásledovatelů."

2.5 První polovina 20. století

Začátek dvacátého století je na obou kontinentech, jak ve Spojených státech tak i v Evropě, ve znam hollerithových tabulátorů. První světová válka soustředila veškerý technický a výrobní potenciál do zbrojní průmyslu, a tak se myšlenky na stavbu nových výpočetních strojů dostaly ke slovu až po jejím konci. V obd mezi dvěma světovými válkami sehráli důležitou roli dva evropané – němec Konrád Zuse a angličan A Turing. Válečné období bylo charakterizováno orientací vývoje matematických strojů na využití pro vojen cíle, ať už v Evropě nebo na americkém kontinentě. Poválečný rozvoj, zejména ve válkou nedotčený Spojených státech, znamenal přesměrování vývojového trendu v této oblasti do komerčních aplikací, ať už hromadném zpracování dat nebo speciálních výpočtech při vývoji nových technologií. Spojené státy získaly této době výrazný náskok a ovládly většinu počítačového trhu poloviny 20. století.

2.5.1 Konrád Zuse

Německý inženýr Konrád Zuse, který v období mezi dvěma válkami zkonstruoval první programovatel počítač určený k řešení složitějších technických výpočtů se narodil v červnu 1910 v Berlíně. Po absolvov berlínské techniky nastoupil na místo statika do letecké firmy Henschel Flugzeugwerke. Obtížné, neustále opakující statické výpočty jej přivedly k myšlence konstrukce stroje, který by tyto výpočty prováděl mí člověka.

V roce 1934 začal pracovat na kalkulačním stroji bez jakýchkoli znalostí současných i předcházejících prac tomto oboru. V době, kdy se počínající "počítačový průmysl" orientoval na výrobu kalkulátorů pracující většinou v desítkové soustavě neznal Zuse ani práce Charlese Babbage. Při svých analýzách a formul problému vycházel ze dvou hledisek

- matematického a logického, které zahrnovalo
  - *metody řízení běhu programu,*
  - *binární systém pro reprezentaci čísel a*
  - *metody pro realizaci aritmetiky v pohyblivé řádové čárce,*
- designérského s orientací na
  - *realizaci plně automatických aritmetických kalkulací,*
  - *paměti s vysokou kapacitou a*
  - *stavebních modulů pracujících na dvoustavovém principu.*

Ve svých prvních úvahách neopouštěl myšlenku sestrojení logických obvodů z tehdy moderní elektromagnetických relé. První stroj Z1[1] (1938) však využíval převážně mechanické principy, zejména proto, čas a náklady potřebné na konstrukci prototypu byly výrazně nižší. Od myšlenky elektromagnetických relé by již jen krok k návrhu speciálního počítače pro protileteckou obranu, který předložil vedení německé armády polovině třicátých let. Jak sám zaznamenal, reakce německého velení na předpokládanou dvouletý plán výstav byla zarážející: "A jak dlouho myslíte, že nám bude trvat než vyhrajeme válku?".

Když se v nejvyšších kruzích setkal s nepořízenou, sestrojil na vlastní náklady malý experimentální str pracující s deseti binárními čísly a obsahující asi 100 elektronek. Na počátku druhé světové války se vrátil zpě myšlence elektromagnetických relé a připravil návrh reléového počítače Z3, když předtím ověřil někte myšlenky na menší experimentální verzi Z2[2] (1939). Plně reléový Z3 měl paměť 64 slov, používal binární syst a aritmetiku v plovoucí řádové čárce s délkou slova 22 bitů (znaménkový bit, 7 bitů pro exponent a 14 bitů p mantisu). Počítač používal 2 400 relé, z toho 600 relé realizovalo aritmeticko logickou jednotku a zbytek b použit pro paměť. Použitá technologie umožňovala operace sčítání, odčítání, násobení, dělení, výpočet dru odmocniny a byly realizovány některé pomocné funkce. Stroj byl řízen děrnou páskou vytvořenou perforovaných použitých filmů a pracoval rychlostí cca 50 operací/s . Použitím přirozeného váhového systému binárním zobrazení čísel se tak Z3 stal prvním počítačem používajícím binární soustavu na místo dos

[1] Replika tohoto stroje byla postavena v roce 1989 a je součástí expozice Museum für Verkher und Technik v Berlíně.

[2] Experimentální počítač Z2 sloužil pouze k ověření některých hypotéz a obsahoval pouze 200 relé.

všeobecně rozšířené decimální. Navržený binární systém prokázal svoji důležitost v dalších návrzích výpočetních strojů, které těžily z výhod použití dvoustavových zařízení jako snímačů děrné pásky nebo štítků nebo dvoustavové elektrické obvody - klopné obvody a hradla realizovaná na nejrůznějších elektronických prvcích (relé, elektronky, tranzistory).

Práce na počítači přerušil Konrád Zuse v roce 1939, kdy byl povolán na vojenskou službu, ale protože pracoval ve vojensky důležitém odvětví, zůstal prakticky na stejném místě jen zaměnil civilní subordinaci za vojenské velení. Veškerý svůj volný čas věnoval dokončení Z3. V roce 1941 byl funkční Z3 předveden německým představitelům leteckého průmyslu, avšak ani zájem německého výzkumného ústavu leteckého nestačil na uvolnění Konráda Zuseho z vojenských povinností. Počítač Z3 byl později zničen při bombardování Berlína, avšak pro jeho historickou důležitost byla o dvacet let později zhotovena replika a umístěna do Deutsches Museum v Mnichově.

V roce 1942 bylo rozhodnuto o výstavbě nového stroje Z4 s větším výkonem, mechanickou pamětí 1024 slov a několika snímači a děrovači děrných štítků. Konstrukce tohoto stroje byla provázena problémy konce války – nedostatkem součástek a neustálými přesuny stroje, aby byl uchráněn před bombardováním. Hotový stroj byl předveden 28. dubna 1945 na ministerstvu letectví v Berlíně, avšak blížící se fronta vedla skupinu kolem Konráda Zuse k přesunu do Harzu, do podzemní továrny na výrobu raketových střel V1 a V2. Zuse odmítl nechat stroj v těchto polních podmínkách a s pomocí Wernera von Brauna se mu podařilo sehnat nákladní vůz a stroj byl postupně přesouván nejprve do vsi Hinterstein v Alpách, později do Rakouska a do Švýcarska. Zde byl instalován na Federálním Polytechnickém Institutu v Curychu, kde pracoval až do roku 1955. Již v roce 1939 založil Konrád Zuse vlastní firmu na výrobu počítačů, která vyráběla počítače řady "Z" až do počátku 60-tých let.

2.5.2 Alan Mathison Turing

Britský matematik Alan Mathison Turing se narodil 23.června 1912 v Londýně. V roce 1931 absolvoval universitu v Cambridge a v akademických kruzích zůstal až do konce svého krátkého života. Alan Turing byl velmi svéráznou osobou s minimálním zájmem o studium prací svých kolegů nebo předchůdců. Jeho práce směřující spíše do oblasti vztahu stroje a přírody se staly základními kameny moderní vědy - umělé inteligence. V roce 1936 publikoval článek nazvaný "On Computable Numbers", ve kterém popsal hypotetické zařízení, dnes známé pod názvem **Turingův stroj**, které teoreticky předznamenalo programovatelné počítací stroje.

Turingův stroj byl navržen k provádění logických operací a mohl číst, zapisovat a mazat znaky zapsané do políček na nekonečné pásce. Tento druh strojů se stal později znám jako "konečné automaty", neboť z každého stavu mohl stroj přejít jenom do konečného počtu dalších možných stavů[1]. Cílem Turingovy práce nebylo navrhnout počítač, ale spíše popsat skupinu problémů, které je možno řešit pomocí logiky. Avšak jeho hypotetický stroj předestřel některé charakteristiky moderních počítačů (např. nekonečná páska = RAM).

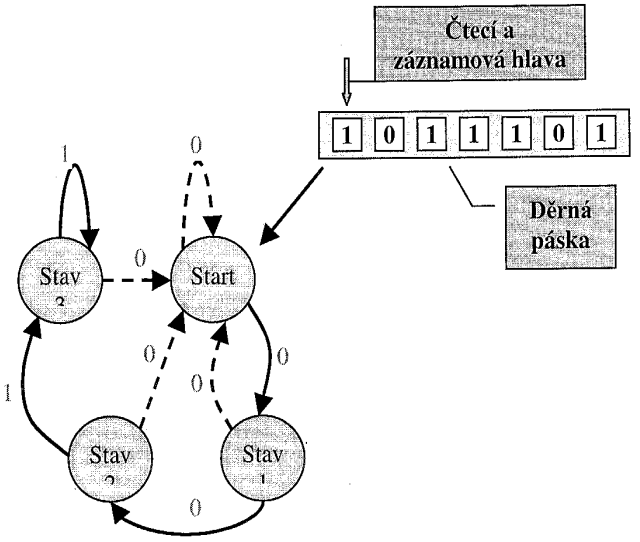
Práce Turingova stroje

Nejjednodušší případ Turingova stroje na Obr. 4 používá čtecí hlavu, která může sejmout údaj z děrné pásky umístěné pod touto hlavou. Podle konstrukce se buď hlava nebo páska mohou pohybovat vždy o jednu pozici vpravo nebo vlevo od okamžité pozice. Stroj řeší následující problém: kdykoliv se na vstupu vyskytnou za sebou tři nebo více jedniček, zobraz na výstupu jedničku, v každém jiném případě zobraz nulu.

Řešení tohoto problému je na uvedeném stavovém diagramu, kde čárkované šipky značí vstup hodnoty nula z pásky a plné šipky odpovídají vstupu hodnoty 1. Výstupy stroje jsou uvedeny u šipek. Vstupní symboly jsou zaznamenány na nekonečné pásce a stroj začíná ze stavu označeného **Start**. Na počátku je pod čtecí hlavou hodnota **1**, stroj změní svůj stav po plné šipce ze stavu **Start** do stavu **Stav 1** a na výstupu zobrazí **0**. Hlava se posune na další pozici a čte hodnotu **0**. Stroj změní svůj stav, tentokrát po čárkované šipce (na vstupu je nula) zpět do stavu **Start** a na výstupu zobrazí opět **0**. Čtecí hlava se posune na další pozici a z ní přečte hodnotu **1**, která navrátí stroj opět do **Stavu 1** po plné šipce a na výstupu zobrazí **0**. Při následujícím kroku však čtecí hlava sejme hodnotu **1**, což způsobí změnu stavu stroje ze **Stavu 1** do **Stavu 2** a zobrazení hodnoty **0** na výstupu (stále nejsou alespoň tři jedničky za sebou) . Posun čtecí hlavy na následující hodnotu **1** způsobí přechod do **Stavu 3** a zobrazení hodnoty **1** na výstupu. Tento postup se neustále opakuje a stroj přechází mezi jednotlivými stavy podle hodnot sejmutých čtecí hlavou z nekonečné pásky.

[1] Při hlubší diferenciaci nemusí být dnes začlenění Turingova stroje mezi konečné automaty již věcně správné.

Koncepce Turingova stroje byla na svou dobu velmi revoluční. Ještě většina počítačů navrhovaných v padesátých letech byla orientována na řešení specifických problémů, zatímco Turingova vize směřovala do oblasti strojů, které by mohly řešit obecně jakýkoli problém. Základním přínosem byla myšlenka řízení počítače, spočívající v rozložení úlohy do řady jednoduchých kroků – instrukcí. Turing věřil, že pomocí algoritmu lze řešit každý problém, záleží jen na schopnosti určit jaké kroky a v jakém pořadí mají být vykonány. Jeho přesvědčení bylo tak hluboké, že předpokládal možnost realizace náhrady lidské mozkové činnosti strojem. V roce 1950 napsal článek, popisující možnost testování inteligence stroje – tzv. **Turingův test**.



Obr. 4 Jednoduchý Turingův stroj

Turingův test

Základem Turingova testu jsou dva dálnopisy, které má k dispozici testující osoba. Jeden z těchto dálnopisů je spojen s jiným dálnopisem, který je obsluhován člověkem, druhý je připojen na výstup a vstup stroje. Testující osoba klade prostřednictvím dálnopisů libovolné otázky. Pokud z odpovědí není možno v přijatelném čase rozlišit, který z dálnopisů je připojen ke stroji a který je obsluhován člověkem, má se za to, že stroj vykazuje vlastnosti hodnocené jako inteligence.

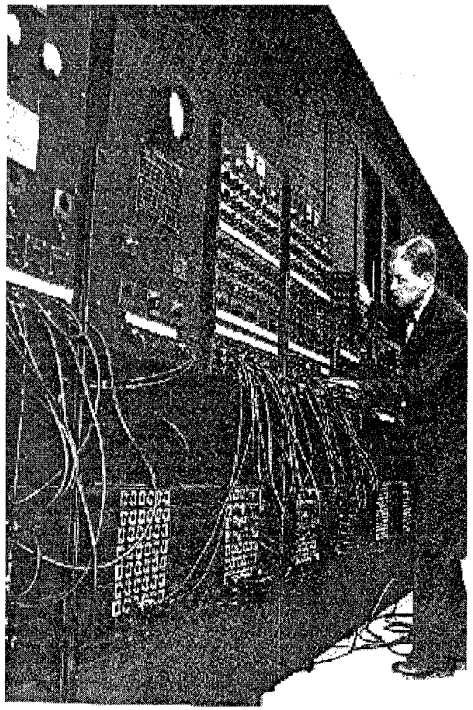
Stejně jako Konráda Zuseho i Alana Turinga poznamenala vojenská služba během druhé světové války. Jeho matematické znalosti přišli vhod britskému Department of Communication, kde se věnoval metodám pro rozluštění německých tajných šifer. Německá armáda používala pro utajení zpráv zvláštní šifrovací stroj nazývaný Enigma, který generoval měnící se kódové sekvence, jež byly v té době běžnými metodami v rozumném čase nerozluštitelné. Skupina ve které Alan Turing pracoval používala k automatizaci některých postupů stroj nazývaný COLOSSUS, který byl sestaven z prvků používaných v té době v telefonních ústřednách - servomotorů, elektromagnetických spínačů. Na konci druhé světové války byl stroj na přímý příkaz Winstona Churchilla rozebrán a dokumentace zničena[1]. Podle jiné verze se dochovaly dva stroje COLOSSUS v provozu až do roku 1960. Oblasti počítačového zpracování informací zůstal věrný i v Národní laboratoři pro fyziku v rámci projektu Automatic Computer Engine (ACE). Odtud v roce 1945 Alan Turing odešel na univerzitu v Manchesteru, kde se zúčastnil prací na vývoji počítacího stroje MADAM (**M**anchester **D**igital **A**utomatic **M**achine).

[1] V roce 1995 se pokusil Tony Sale, na základě dochovaných poznámek a vyprávění pamětníků, postavit repliku stroje COLOSSUS pro museum v Bletchley Park, kde je expozice historie šifrování.

Význam jeho příspěvku ke stavbě počítacích strojů zastřel Turingovy studie v jiných přírodních vědách, zejména v biologii. K té ho vedly zejména myšlenky na stavbu inteligentního stroje a teoreticky se zabýval fyzikálními strukturami živých organismů – proč právě miliony buněk vytvoří danou formu. Alan Mathison Turing zemřel 7. června 1954 na otravu kyanidem ve své laboratoři.

2.6 Rozmach počítačů

Koncem třicátých let se "hollerity", jak se říkalo strojům pracujícím s děrnými štítky, natolik usídlily na americkém i evropském trhu, že se staly neodmyslitelnou a spolehlivou součástí administrativy. Nově se otevírající trh přivedl Howarda Hathawaye Aikena[1] ke spolupráci s techniky firmy IBM a pod jeho vedením byl navržen a zkonstruován velký automatický počítač založený na standardních elektromechanických prvcích. Tento stroj, pojmenovaný **Mark I**[2], byl uveden do provozu v roce 1944 ve Výpočetní laboratoři harvardské univerzity. Pracoval s čísly o 23 desítkových místech, uložených s kódem jedna z deseti, a mohl provádět čtyři základní aritmetické operace. Navíc měl zabudovány speciální podprogramy, které realizovaly logaritmy a trigonometrické funkce. Mark I (1944) byl řízen papírovou děrnou páskou bez možnosti zpětného chodu, což znamenalo nemožnost zpětných skoků v programu. Výstupním zařízením byl děrovač děrných štítků a elektrický psací stroj. Ačkoliv hlavním komponentem počítače Mark I bylo 72 bubnových rotačních čítačů (podobně jako v Pascalině) a 3500 dekadických voličů doplněných o elektromagnetická relé; tento první počítač byl charakterizován jako reléový. Z dnešního hlediska se zdá čas tři až pět sekund na vynásobení dvou čísel jako dlouhý. Ale jednalo se o plně automatický proces, který mohl být opakován a řízen a tak byly umožněny dlouhé výpočty bez zásahu člověka. Howard Aiken by nositelem i dalšího projektu **Mark II** (1947) - plně reléového počítače.



Obr. 5 Počítač ENIAC

S úspěchem počítače **Mark I** se rozběhl konkurenční projekt na pensylvánské univerzitě ve Filadelfii[3], kde **John W. Mauchly** a **J. Presper Eckert** dokončili v roce 1946 počítač **ENIAC** (**E**lectrical **N**umerical **I**ntegrator and **C**alculator). **ENIAC** používal slovo s deseti dekadickými číslicemi (nikoliv binárními), 18 000 elektronek a vážil asi 30 tun. Jeho rozměry (asi 30 metrů délky, 1 metr šířky a 3 metry výšky) a příkon byly úctyhodné. Vstupní jednotkou byl snímač děrných štítků a výstupem děrovač děrných štítků. Jeho aritmeticko-logická jednotka obsahovala m.j. násobičku, děličku, jednotku pro druhou odmocninu a 20 sčítaček s decimálními čítači. Poprvé zde byla použita "rychlá" paměť - sada registrů (přístupová doba 0,2 ms). Instrukce byly zabudovány do zvláštních jednotek, které byly

[1] Americký matematik, průkopník počítačového průmyslu 9.března 1900 - 14.března 1973. V době vzniku Mark I pracoval na Harvardské univerzitě.

[2] Původní označení projektu bylo ASCC – Automatic Sequence Controlled Calculator

[3] University of Pennsylvania, patřící do "evy league" amerických univerzit.

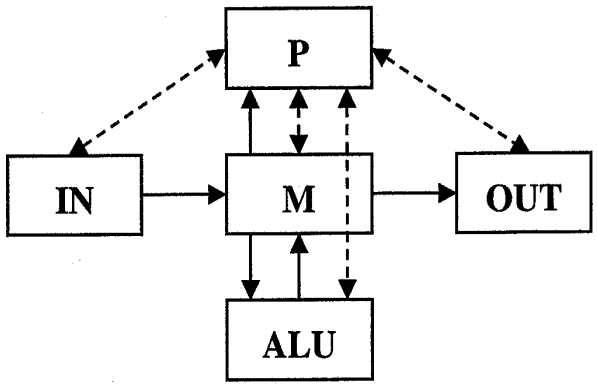
mezi sebou propojovány kabely a tak vytvářely řídící strukturu počítače (tok instrukcí = pořadí jednotek + propojení). S každou novou úlohou bylo nutno tuto část počítače přestavět - je otázka zda můžeme z tohoto hlediska považovat **ENIAC** za programovatelný stroj. Později byl počítač převzat armádou, která ho používal pro svoji balistickou laboratoř, a to až do roku 1955[1]. Autoři počítače se úspěšném ukončení projektu rozhodl založit vlastní firmu pro výrobu a dodávky počítacích strojů.

Historie **ENIAC**u má ještě zajímavou dohru - v roce 1971 napadl patent chránící jeho základní digitáln koncepci fyzik John V. Atanasoff (původem z Bulharska) s odkazem na svůj počítač založený na stejném principu a uvedený do chodu v roce 1930 na Iowa State College, kde tehdy působil. V roce 1973 mu americký nejvyšší soud přiznal právo původnosti.

2.6.1 John Louis von Neumann

John Louis von Neumann[2], americký matematik maďarského původu, je snad nejznámější postavou v historii výpočetní techniky. Studoval matematiku v Budapešti a svůj první odborný článek publikoval v 18 letech Přestože po vstupu na univerzitu měl sklon ke studiu chemie, brzy se vrátil ke svému původnímu tématu : věnoval se teorii množin a algebře. V letech 1927 až 1930 byl hostujícím profesorem na princetonské univerzitě : po založení Institute for Advanced Studies v roce 1933 se stal jedním z jeho prvních šesti profesorů matematiky Zde také zůstal až do konce svého života.

Když Alan Turing navštívil jako postgraduální student v letech 1936 – 1938 princetonskou univerzitu, John von Neumann mu nabídl místo svého asistenta. Avšak Turing odmítl a vrátil se zpět na svoji rodnou anglickou univerzitu v Cambridge. Tato návštěva se uskutečnila krátce potom, co Turing publikoval svůj článek "On Computable Numbers and Application to the Entscheidungs-problem", který obsahoval koncepci logického návrhu univerzálního počítacího stroje. Historie má za to, že John von Neumann musel znát Turingovy výsledky a že je aplikoval do svých objevných návrhů o deset let později [H5]. Rovněž pozice John von Neumanna během druhé světové války, kdy jako expert a konzultant několika vládních komisí byl ve styku s vědci, kteří : důvodu utajení byli od sebe izolováni, přispěla k jeho obrovskému přehledu v trendech vývoje nového oboru Byl to právě on, kdo přivedl k sobě dvě vzájemně velmi odlišné skupiny – atomové fyziky pracující v Los Alamos na projektu nukleární zbraně a skupinu v Moore Schoole of Electrical Engineering připravující počítač ENIAC.



Obr. 6 Struktura počítače dle von Neumanna

V roce 1945 publikoval své závěry, ve kterých dokázal, že počítač může mít jednoduchou pevnou fyzickou strukturu a přesto může být schopen provádět jakékoliv výpočty, řízené programem, aniž by jeho fyzická struktura musel být modifikována. Tato koncepce, s programem uloženým v paměti, se stala základem pro návrh

[1] Část počítače ENIAC je dodnes vystavena u zadního vchodu elektrotechnické fakulty University of Pennsylvania.

[2] Narozen 28. prosince 1903 v Budapešti, zemřel 8. února 1957 ve Washingtonu DC.

Ústřední knihovna

moderních rychlých počítačů[1]. Hlavními myšlenkami koncepce John von Neumanna, která přežila s malými změnami desítky let bouřlivého vývoje, je rozdělení do čtyř vzájemně spolupracujících částí (viz Obr. 6) a stanovení několika základních pravidel

1. Počítač je tvořen řadičem (P), aritmeticko-logickou jednotkou (ALU), pamětí (M), blokem vstupů (IN) a blokem výstupů (OUT).
2. Struktura počítače je neměnná a jeho práce je řízena programem uloženým v paměti.
3. V paměti jsou uložena data společně s instrukcemi programu.
4. Paměť je tvořena jednotlivými paměťovými místy, do kterých se dá zapisovat a ze kterých je možno uloženou informaci přečíst po zadání adresy tohoto místa. Adresa místa je dána pořadovým číslem tohoto místa.
5. Program je tvořen posloupností instrukcí, které určují elementární změnu stavu počítače.
6. Pořadí provádění instrukcí programu je, až na výjimky, sekvenční. Výjimkou jsou instrukce skoku.
7. Instrukce, data a adresy jsou kódovány binárně.

Zavedením nové instrukce, podmíněného skoku v programu, umožňovalo převedení toku programu za jistých podmínek do jiného místa[2]. Uložení instrukcí i programu do stejné jednotky paměti vedlo k absurdní, avšak donedávna programátory vtipně používané, možnosti - modifikovat instrukce podobně jako data. V historii se vyskytují i organizace počítačů, které oddělují data od instrukcí – podle místa svého vzniku se nazývají harvardskou architekturou[3].

Výsledkem jeho práce bylo rychlejší, flexibilnější a účinnější programování s možností použití programových "subroutine" s možností jejich uložení do programových knihoven. První generace počítačů, která aplikovala výhody nového přístupu se objevila v roce 1947. Tyto počítače již používaly operační paměť s náhodným přístupem (RAM) s rozsahem cca 1000 slov, byly kompaktnější než **ENIAC** a měly mnohem menší rozměry (měly zhruba velikost piana a obsahovaly asi 2500 miniaturních elektronek). Vyžadovaly neustálý servis, spolehlivost provozu se pohybovala okolo 80% a programovaly se ve strojovém kódu. Mezi první členy této skupiny patřil **EDVAC** a **UNIVAC**.

EDVAC (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) byl nesmírným vylepšením **ENIAC**u. **John W. Mauchly** a **J. Presper Eckert** začali pracovat na myšlence nového počítače již dva roky před tím, než byl ENIAC spuštěn. Myšlenka počítače s programem uloženým v operační paměti dostala v tomto případě smysl, neboť EDVAC měl mít mnohem větší paměť než všechny jeho předchůdci. Jeho vnitřní paměť byla realizována na rtuťových zpožďovacích linkách a pro zjednodušení konstrukce aritmetických jednotek používal binární aritmetiku.

Aktivita Mauchlyho a Eckerta se postupně soustředila na jejich vlastní firmu Eckert-Mauchly Computer Corporation. Mauchly, který byl více obchodně založen, očekával, že statistický úřad Spojených států bude vhodným zákazníkem v blížícím se sčítání obyvatelstva. Se svoji reputací, jako tvůrci ENIACu, předložili perspektivním zákazníkům návrh na dodávku počítače pro tyto účely a jejich návrh byl akceptován. Začaly práce na počítači UNIVAC, avšak podfinancování projektu brzo přivedlo firmu EMCC do potíží. Teprve s pomocí Remington Rand Corp. byl počítač dokončen a v roce 1951 předán a akceptován Statistickým úřadem Spojených států. Následovně byly objednány a během osmnácti měsíců dodány další dva stroje.

UNIVAC byl podstatným technickým krokem vpřed. Byl založen na koncepci von Neumanna, hlavní paměť měla kapacitu 1000 slov, sekundární paměť používala magnetické pásky a periferní zařízení byla značně zdokonalena. Jako vstupy nebo výstupy byly používány elektrický psací stroj, snímač a děrovač děrných štítků a tiskárna. S hodinovým kmitočtem 2,25 MHz byl počítač daleko rychlejší než jeho předchůdce ENIAC – např. operace dělení trvala 3,89 ms oproti 24 ms na počítači ENIAC. Snížení celkového počtu elektronek na 5 400 z původních 18 000 vedlo i ke zvýšení spolehlivosti celého systému. Počítač UNIVAC byl rovněž použit při odhadu volebních výsledků v prezidentské kampani v roce 1952. Přestože výsledky zpracované počítačem předpověděly Eisenhowerovu výhru, novináři tomu nevěřili a uveřejnili opak. Po zjištění shody odhadů zpracovaných počítačem a skutečných výsledků propukla v mediích oslava nové techniky.

Firma Eckert-Mauchly Computer Corporation, jako divize Remington Rand Corp., slavila úspěch s novou verzí počítače. Avšak nedostatek kvalifikovaných pracovníků, zabývajících se aplikacemi universálních počítačů otevřel novou příležitost vlastního uplatnění pro Johna W. Mauchlyho, který EMCC v roce 1959 opustil a založil si vlastní konzultační firmu.

2.6.2 Počítač Mark I

V letech 1947 – 1948 byl v Anglii, na univerzitě v Manchesteru, připraven prototyp počítače, jehož úkolem bylo prověřit nové možnosti zrychlení a zvýšení spolehlivosti matematických strojů. V projektu Small Scale Experimental Machine, známém spíše pod přezdívkou "The Baby", použili jeho autoři F.C. Williams a Tom Kilburn nový prvek pro uložení informací – paměťovou elektronku[1]. Základní parametry počítače byly

- 32-bitové slovo,
- sériová binární aritmetika, používající pro záznam čísel jejich dvojkový doplněk,
- hlavní paměť 32 slov s náhodným přístupem, rozšiřitelna až na 8192 slov,
- rychlost výpočtu asi 1,2 ms na jednu instrukci.

Počítač obsahoval tři registry – akumulátor, čítač instrukcí a registr právě prováděné instrukce, které byly součástí hlavní paměti realizované pomocí paměťové elektronky. Poměrně zvláštním způsobem zvolená sada sedmi instrukcí obsahovala i instrukci podmíněného skoku a zastavení programu. První programy pro demonstraci prototypu byly napsány Alanem Turingem.

The Baby se stalo předchůdcem komerčního stroje Mark I, jehož specifikace byla předložena na podzim 1949, avšak své finální podoby se dočkal až ve spolupráci s firmou Ferranti jako "Ferranti Mark I". Základní charakteristiky stroje se dají shrnout následovně:

- Paměť byla organizována jako 20-ti bitové adresovatelné řádky, kde instrukce měla délku jedné a data dvou řádek.
- Aritmetika 40-ti bitová, obsahující hardwarovou sčítačku a násobičku (akumulátor měl dvojnásobnou délku) a logické instrukce.
- Sada osmi speciálních registrů používaných pro modifikace adres v instrukcích, se zvláštní aritmetikou a testovacími instrukcemi.
- Hardware mohl provádět padesát jednoadresových instrukcí se standardním časem provádění instrukce 1,2 ms; jediná instrukce násobení trvala 2,16 ms.
- Paměť s náhodným přístupem rozdělená na osm stránek, jedna stránka definována jako 64 řádků po dvaceti bitech.
- Sekundární paměť na magnetických bubnech obsahovala 512 stránek, dvě stránky na jednu stopu s přístupovým časem 30 ms.
- Instrukce pro operace vstupu a výstupu zahrnovaly čtení a děrování pětistopé děrné pásky, a přenos jedné stránky mezi primární a sekundární pamětí.

Výstupy na děrnou pásku mohly být stejně jako vstupy přesměrovány na dálnopis. Právě pětistopý záznam byl zdrojem z dnešního hlediska přímo "hororového" zápisu programu. Bylo téměř nemyslitelné napsat program bez intuitivní práce ve dvaatřicítkové soustavě (pětibitový záznam). Celá čísla byla zaznamenána jako osm znaků, instrukce se sestávaly ze čtyř a adresy ze dvou znaků.

---

[1] Tato koncepce je často nazývána architekturou von Neumanna. Jak uvidíme dále, toto označení není zcela přesné.

[2] Tato idea byla uvedena již v návrhu Analytical Engine Charlese Babbage.

[3] Harvardská architektura byla použita např. v počítači MARK-I a jeho následovnících, avšak stává se populární i v posledních verzích výkonných mikroprocesorů, např. rodina výkonných mikroprocesorů SHARK firmy Analog Devices.

[1] V anglosaské literatuře se často tento prvek označuje jménem autora jako William's tube.

Program na stroji Ferranti Mark I

V té době byly používány tzv. pětistopé děrné pásky, kde každý z pěti otvorů v příčně uložené řádce měl svoji binární váhu. Program se zaznamenával jako pětibitové ekvivalenty binárních čísel do děrné pásky, takže např. čtyřicetibitové číslo

10001 11011 10100 01001 10001 11001 01010 10110

bylo do programu zapsáno posloupností odpovídajících abecedně-číslicových znaků v pětibitové reprezentaci, tedy jako

Z"SLZWRF.

Podobně byly zaznamenány i instrukce a tak celý program vypadal jako změť naprosto nic neříkajících znaků, např.

/E@A:SIUFDRJNFCKTYLWHZPQOBG:MXVQ

Jistým ulehčením bylo, když v roce 1954 publikoval Tony Brooker Mark I Autocode, první "programovací jazyk". Tom Kilburn a Geoff Tootil, autoři a programátoři stroje Mark I, se pokusili v roce 1996 zrekonstruovat první program, který byl pro Mark I napsán, ze svých poznámek. Tento experimentální program určoval největší činitele, jejichž součin bude v rozsahu 131 072 až 262 143[1]. Běh tohoto programu trval na počítači Mark I 52 minut během kterých bylo provedeno 2,1 milionu instrukcí a přes 3,5 milionu přístupu do paměti.

2.6.3 Počítač EDSAC

Nedaleko od Manchesteru, na starobylé anglické univerzitě v Cambridge se připravoval pod vedením Maurice V Wilkese projekt samočinného počítače s názvem EDSAC (Electronic Delay Storage Automatic Calculator). Založen na myšlenkách prvního stroje ENIAC, v univerzitní matematické laboratoři vyrůstal gigant obsahující 3 500 elektronek. Ač se to zdá na tehdejší dobu neočekávané, s počítačem byly propojeny tři obrazovkové monitory, na kterých byl svítícími body zobrazen obsah paměti počítače. Ta byla vybudována na bázi zpožďovacích linek, kde se informace uchovávala pomocí rychle cirkulujících pulsů[2]. Periferním zařízením byl snímač děrné pásky a elektrický psací stroj.

Programování počítače EDSAC rovněž nebylo jednoduché i když mnohem snadnější než u počítače Mark I. Premiéra počítače EDSAC se odehrála 6. května 1949, kdy po zavedení děrné pásky se zkušebním programem začal elektrický psací stroj vypisovat řadu čísel 1, 4, 9, 16, 25, 36 ..., mocninnou posloupnost. V roce 1980 objevil Maurice Wilkes, když opouštěl svou pracovnu na univerzitě a odcházel do důchodu, v rohu zásuvky papírovou pásku s programem pro EDSAC. Zaslal ji Martinu Campbell-Kellymu na univerzitě ve Warwicku, který se specializoval na počítačovou historii. Cambell-Kelly program nejen zrekonstruoval, ale na základě dalších informací připravil v roce 1992 i počítačový simulátor, repliku EDSACu, pro personální počítače[3].

Programování počítače EDSAC

Bylo navrženo celkem 18 dostupných kódů pro jednotlivé operace. Tyto kódy odpovídaly kódům alfanumerických znaků, každý byl reprezentován jedním znakem, a v mnoha případech odpovídaly i mnemonicky. Např. zápis **A45** znamenal "přičti (ADD) obsah adresy 45 k obsahu akumulátoru.. Podobně zápis **S** (SUBTRACT) znamenal odčítání, **I** (INPUT) vstup následujícího znaku ze snímače děrné pásky a **O** (OUTPUT) vytištění jednoho znaku na elektrickém psacím stroji.

[1] Tedy 2^{17} až 2^{18}-1.

[2] Paměti tvořené zpožďovacími linkami zůstaly v oblibě poměrně dlouho, zejména tam, kde sériový záznam informace mohl být přirozeně využit. Ještě v roce 1976 dodávala anglická firma ICL obrazovkové displeje ICL7181 vybavené pamětí na zpožďovacích linkách.

[3] Verze V5.0 tohoto programu z roku 1998 je dostupná pro Windows95 a WindowsNT.

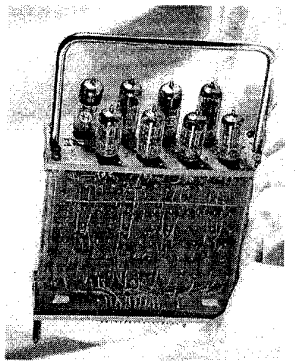
2.7 Počítačový průmysl

Přechod mezi pionýrskou dobou budování prvních počítačů na univerzitách nebo pro potřebu armády době druhé světové války, kdy existovaly pouze ojedinělé exempláře, a nástupem průmyslové výroby b historickou nutností a došlo k němu s rychlostí až neobvyklou. Nové firmy, které vznikaly většinou počítačových nadšenců, kteří byly bezprostředně účastni vývoje na univerzitě, neměly obvykle dostatečný kapit a jejich projekty byly finančně podceněny. Tak se obvykle, po strmém nástupu, dostaly do finančních problémů skončily v lůně některé z velkých a zavedených firem. Avšak právě tyto, obchodně zdatné a zavedené firmy uvědomovaly, že se pro ně otevírá nový trh. A tak velmi často investovaly právě do rozvoje perspektivníh odvětví s očekávání bohatých zisků v nedaleké budoucnosti.

Historie rozvoje počítačového průmyslu není historií lidí nebo firem. Jak lidé tak firmy na trh přicházeli odcházeli. Je historií strojů a lidí nebo firem za těmito stroji. Jako obvykle však existuje několik výjimek, mezi ně patří dnes již legendární International Business Machines, neboli IBM, založená Hermanem Hollerithen Úspěch jeho "holeritů" přinesl firmě počáteční kapitál a díky dlouhodobé obchodní a technické strategii je dosu jedním z největších dodavatelů počítačů na světě.

První reléový počítač Mark I (viz kap. 2.6) byl postaven ve spolupráci s Harvardskou univerzitou v roce 1944. Ten byl předzvěstí sériově vyráběných elektronkových[1] počítačů

- IBM604 (v roce 1948), elektronkový počítač s registry,
- IBM701 (v roce 1952), elektronkový počítač s pamětí založenou na paměťové elektronce,
- IBM650 (v roce 1954), elektronkový počítač s magnetickou bubnovou pamětí a
- IBM704 (v roce 1956), který měl paměť na feritových jádrech.



Obr. 7 Část registru elektronkovéhc počítače

Tranzistorové počítače firmy IBM přišly velice rychle po objevu těchto součástek a první počítač vybavený polovodičovou technologií IBM7090 (v roce 1960) byl pětkrát rychlejší než jeho předchůdci s elektronkami. Zlomem v historii počítačové firmy byl rok 1964, kdy byl uveden na trh počítač IBM360. Jeho význam nespočívá ani tak v tom, že byl postaven na monolitických a hybridních integrovaných obvodech, jako v jeho nové architektuře a organizaci, které se s malými změnami dochovaly až dodnes.

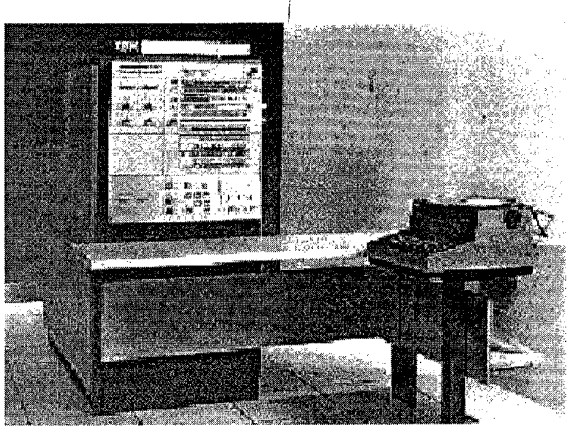
Počítač IBM360 měl stavebnicovou strukturu hlavní paměti, jednotnou strukturu dat a instrukcí a jednotn způsob připojování periferií. Adresování hlavní paměti probíhalo po bytech a délka slova se vyjadřovala násobcích bytů. Aritmeticko-logická jednotka byla schopna provádět 170 instrukcí, včetně operací s čísly pohyblivou řádovou čárkou. Řídící paměť byla rozdělena na permanentní a semipermanentní, což umožňovalo emulaci instrukcí jiných počítačů. Adresovací systém pracující s bázovými registry umožňoval snadné přemístěn programu v paměti a připojování velkokapacitních pamětí. Ochrana dat v hlavní paměti proti neoprávněném přístupu byla zajišťována v blocích 2 kB, přičemž chybné čtení z paměti bylo signalizováno pomocí přerušení Stav stroje byl indikován zvláštním stavovým slovem, které obsahovalo údaje vztahující se k poslední provedené instrukci. Periferie byly připojovány prostřednictvím dvou standardních kanálů – multiplexním a selektorovým které byly řízeny pomocí řídících slov a pracovaly autonomně. Počítač IBM360 byl vybaven operačním systémem OS/360, který již umožňoval vzdálené vstupy z terminálů a sdílení času mezi programy[2]. Bohatý sortiment překladačů Asembleru, Algolu, Fortranu, Cobolu, RPG a PL/1doplňoval tento komerční

[1] Na Obr. 7 je část registru elektronkového počítače, která byla použita k uložení jedné dekadické cifry, tedy čísla 0 až 9. Není divu, že počítače v té době dosahovaly obřích rozměrů.

[2] Systém OS/360 umožňoval současný běh maximálně 15 úloh.

nejuspesnejsi produkt firmy. Architekturu IBM360 prevzaly nejenom jeho nasledovnici, zejmena IBM370, ale stala se vzorem i pro stroje jiných firem[1]. Tato koncepce byla zachována i v pozdějších strojích, např. IBM4300 (v roce 1979) nebo IBM3081 (v roce 1981).

Velmi často se stávalo, že někteří přední pracovníci velkých firem založili vlastní firmu k realizaci svých myšlenek a od velké firmy odešli. Jejich výrobky pak byly zcela nové, nebo levněji kopírovaly mateřské výrobky s poněkud jiným designem, avšak zachovávajíc kompatibilitu. To byl příklad firmy Amdahl, kterou založil v roce 1970 inženýr firmy IBM. Počítače Amdahl byly plně slučitelné s počítači IBM370, avšak levnější. Jejich výhodou bylo, mimo ceny, i velmi účinné využívání virtuální paměti. Mezi nejznámější typy této firmy patří např. systém 470 V/8 z roku 1978 s výkonem 6 MIPS nebo 580-5880 s výkonem až 24 MIPS.



Obr. 8 Počítač IBM360 – centrální jednotka a pult operátora

Pravděpodobně nejstarší původní počítačovou firmou na evropském kontinentě založil Konrád Zuse v roce 1949. Vyráběla reléové počítače dle jeho návrhů a pod označením Z5 (v roce 1953), Z11 (v roce 1957) a elektronkový počítač Z22 (v roce 1958). Jeho zlepšená transistorová verze přišla na trh v roce 1960 a měla feritovou operační paměť s kapacitou 10 000 slov.

Průnik amerických firem na evropský trh se projevil i jejich majetkovou účastí u zavedených evropských výrobců elektronických zařízení. Příkladem může být francouzská firma CII-Honeywell-Bull (Companie Internationale pour l'Informatique – Honeywell – Companie des machines Bull), která se v osmdesátých letech řadila, podle objemu výroby počítačů, na druhé místo na světě. Od roku 1968 vyráběla počítače Iris 50 a Iris 80. Nejvyšší typy této řady, zavedené do výroby v roce 1974, měly možnost osadit až 15 MB operační paměti a dvěma kilobyty paměti cache předsunuté každému z maximálně čtyř procesorů. V licenci této firmy se vyráběl v sedmdesátých letech v Čechách počítač Tesla 200.

2.8 Poválečné období v Čechách

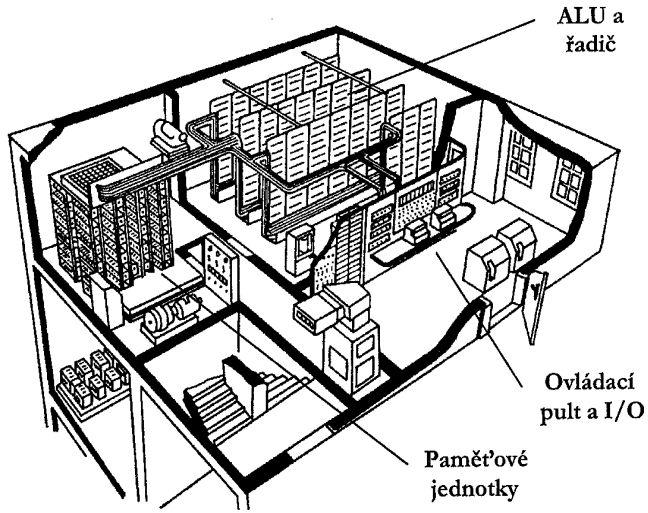
Poválečné období v Čechách sledovalo tendence k rozvoji odvětví těžkého průmyslu a strojírenství, tak jak vyžadovaly potřeby poválečného východního bloku. Oblast elektroniky a výpočetní techniky patřila do okrajových sfér státního zájmu a to i hlediska ideologického, neboť jak pojem kybernetika, tak i tento obor byl považován za imperialistickou pavědu. Nicméně i v těchto relativně omezených podmínkách vznikaly a realizovaly se nové myšlenky a nápady, rodily se osobnosti nové vědy.

První návrh samočinného počítače byl představen v roce 1947 na půdě tehdejšího Badatelského ústavu matematického České akademie věd a umění a obsahoval některé myšlenky, které byly později realizovány v projektu **SAPO**. Modely částí počítacích strojů, jako např. elektronková násobička nebo reléový kalkulátor, byly vyrobeny v tehdejším národním podniku Zbrojovka Praha, později Aritma n.p., a sloužily k předvádění možností elektronických počítačů a získání podpory pro rozvoj oboru. V srpnu 1950 bylo při Ústředním ústavu matematickém založeno oddělení matematických strojů, jehož prvním projektem byl návrh a realizace jednoúčelového počítače **M1**, jehož základní parametry vycházely z požadavků fyzikálního ústavu. Stroj měl vestavěný algoritmus na provádění trojrozměrné Fourierovy transformace a byl určen k výzkumu krystalových struktur. Pracoval v binárním kódu s rychlostí asi 40 operací za sekundu a byl spuštěn v roce 1952.

[1] Např. firma ICL založila svoji řadu počítačů System 4 právě na architektuře IBM360. Jeden z nejvýkonnějších počítačů této řady ICL 4-72 (začátek výroby v roce 1972) byl instalován v roce 1975 v Oblastním výpočetním centru vysokých škol při ČVUT v Praze.

Od počátku vzniku oddělení byl připravován návrh architektury prvního československého samočinného počítače - SAPO[1], který se na tehdejší dobu a omezené technologické možnosti vyznačoval progresivními architektonickými nápady. Aritmetika počítače byla binární s pohyblivou řádovou čárkou a délkou slova 32 bitů a měl magnetickou bubnovou paměť o kapacitě 1024 slov. Instrukce byly pětiadresové a počítač pracoval s rychlostí 5 op/s. Kromě dvou adres operandů a adresy místa uložení výsledku do hlavní paměti obsahovala dvě adresy následujících instrukcí, mezi nimiž se rozhodovalo podle znaménka výsledku.

Zajímavým prvkem byla automatická oprava nahodilých chyb ve výsledcích aritmetických operací bez přerušení výpočtu. Aritmetická jednotka byla trojnásobná a každá její část prováděla stejné operace nezávisle na druhé. Vyhodnocovací obvody řadiče vybraly vždy výsledek, který byl alespoň ve dvou jednotkách stejný[2]. Tyto vyhodnocovací obvody prověřovaly současně i svoji vlastní činnost. Přesuny informací uvnitř stroje byly zajištěny paritou, kontrolním čtením nebo trojnásobným přesunem informace.



Obr. 9 První československý počítač SAPO

S ohledem na relativně vysokou rychlost paměti bylo možno navrhnout vnitřní zapojení počítače do maximální míry paralelní. Operační jednotka obsahovala 15 střadačů, každý o délce jednoho slova, v nichž se při násobení a dělení vytvářely násobky operandů. Algoritmy ostatních operací byly navrženy tak, aby co nejvíce využívaly těchto obvodů ke zkrácení operačních časů. Např. při sčítání v pohyblivé řádové čárce se ve střadačích současně vytvořila celá řada možných výsledků s navzájem různě posunutými operandy. Z nich se pak vybral správný výsledek na základě rozdílu hodnot exponentů obou operandů. Paralelnost systému se při tehdejší součástkové základně odrazila v rozměrech a nákladnosti celého zařízení, které obsahovalo na 7 000 relé a 400 elektronek. Počítač SAPO, který byl předán do provozu v únoru 1958, sloužil k vědeckotechnickým výpočtům pro různé podniky až do února 1960, kdy vznikl požár v reléové části stroje a zcela ji zničil. Po požáru stroj nebyl opraven a do dnešní doby se z něho zachovalo pouze několik drobných částí. Ve srovnání se světovou historií počítačů vznikl počítač SAPO s malým zpožděním, v Evropě však patřil mezi špičkové stroje.

Souběžně s dokončovacími pracemi na projektu SAPO probíhaly i přípravy projektu EPOS, kterému předcházely dvě experimentální verze pojmenované **E1a** a **E1b**. Počítač E1a pracoval v dekadické soustavě s pevnou řádovou čárkou s osmimístnými nebo šestnáctimístnými čísly. Doby jednotlivých operací se pohybovaly v rozmezí od 200 ms do 3600 ms. Stroj měl 10 osmimístných reléových pamětí mezivýsledků. Vstupními jednotkami byly stránkový dálnopis a dva snímače děrné pásky, z nichž jeden byl určen ke vkládání vstupních dat a druhý k automatickému řízení počítače. Výsledky tiskl tentýž stránkový dálnopis, který sloužil k ručnímu ovládání. Celý stroj, včetně reléových pamětí, obsahoval přibližně 1500 miniaturních dvoukontaktových polarizovaných relé.

[1] Zkratka z názvu **SA**močinný **PO**čítač.

[2] Tento způsob zabezpečení proti chybám se nazývá systém TMR (Triple Modular Redundancy) a dosud se často používá tam kde na spolehlivosti výsledku nebo operace závisí např. lidské životy.

amočinný počítač E1b pracoval v dekadické soustavě s desetimístnými čísly s pohyblivou řádovou čárkou sahu 30 dekadických řádů. Doba trvání operací se pohybovala v rozmezí 180 ms až 1200 ms. Magnetická ová paměť o kapacitě 990 slov sloužila k ukládání dat instrukcí i textu. Instrukce byly tříadresové. Vstupní nace se vkládaly z děrné pásky nebo z klávesnice určené k ručnímu vkládání dat i instrukcí. Sadou ních tlačítek bylo možno přerušit probíhající program a spustit výpočtové rutiny předem vložené do ové paměti. Tak bylo možno sdílet počítač mezi několika pracovišti. Na výstupu počítače byl připojen cký psací stroj a děrovač pásky. Počítač obsahoval asi 1000 čtyřkontaktových polarizovaných relé a přibliž-0 elektronek.

využitím zkušeností nabytých při konstrukci strojů E1a a E1b vycházel návrh počítače **EPOS1**, který byl n do provozu v roce 1963. Ten obsahoval asi 8 000 elektronek, a používal feritovou[1] paměť o kapacitě slov s délkou 65 bitů. Zpracovával dvanáctimístná desítková čísla v kódu dva z pěti, který umožňoval u jedné chyby bez zastavení, s rychlostí až 20 000 operací/s.

Při návrhu počítače EPOS1 se kladl hlavní důraz na maximální využití strojového času – tedy času soru. Byl to požadavek v té době neobvyklý, proto se také koncepce počítače od tehdy běžných návrhů mně odchylovala. Na tehdejší dobu unikátní myšlenka sdílení času procesoru se do oblasti návrhu počítačů dila až mnohem později. Základní filosofie sdílení času rozlišovala tzv. vnější sdílení a vnitřní sdílení času. **ní sdílení času**, umožňovalo řešit až pět navzájem nezávislých programů a bylo založeno na principu í aritmetické jednotky ke zpracování instrukcí jiného programu v době, kdy původní program čekal na čení operace přídavného zařízení. Sled využití aritmetické jednotky byl řízen organizátorem (hardwarovým ním, jednotkou). Potřebný rozsah paměti každému programu přidělil přepínacím polem, umístěným na acím pultě operátor nebo programátor, přičemž jednotkou přidělování bylo 100 nebo později 1000 slov. nací pole plnilo současně funkci mapovacího i ochranného mechanismu, tj. provedlo transformaci do čných (fyzických) adres a kontrolovalo užití paměti přidělené jednotlivým programům. Při překročení se hlásila chyba a program se zastavil. Část paměti přidělená programu nemusela být souvislá. **Vnitřní ní času** umožňovalo, aby při dlouhých operacích násobení a dělení se současně prováděla dlouhá operace a lně s ní i operace po ní následující. Pochopitelně, že operandem následující operace nemohl být výsledek hozí operace. Vnější sdílení času bylo možno programově využít dvojím způsobem:

- k paralelnímu provádění až pěti programů na sobě nezávislých,
- pro řešení jedné úlohy rozdělené na části, které mohly být zpracovány paralelně jako různé programy. Vzájemná vazba byla řízena jedním z probíhajících programů (obdoba Dijkstrových semaforů)[2].

Významnou vlastností počítače EPOS1 byla jeho **stavebnicovost**. K základnímu počítači (feritová paměť, lní jednotka, ovládací pult) bylo možno připojit více periferních zařízení téhož typu v různé kombinaci. zařízení závisel na zpracovávaných úlohách (maximálně deset jednotek od každého typu zařízení). na periferní zařízení byla připojena jednotným způsobem. Každé přídavné zařízení bylo vybaveno ími řídícími obvody, které podle instrukcí předaných základním počítačem řídily práci přídavného zařízení a kanál" pracoval autonomně. Spojení se základní jednotkou se vyžadovalo jenom tehdy, bylo-li nutno předat lata nebo instrukce.

Spolehlivost počítače byla zajištěna použitím samoopravných kódů a detekčních kódů. Operace s m zařízením, čtení nebo zápis, se prováděly po blocích pěti slov. Dekadická soustava byla použita pro entaci operačního kódu stejně jako pro zobrazení čísel i znaků, kde znak byl reprezentován dvěma ickými číslicemi. Počítač byl vybaven českou i slovenskou abecedou. Rovněž sada univerzálních registrů, telná pro urychlení operací, byla unikátní myšlenkou.

vé paměti využívaly fyzikálního jevu magnetizace a demagnetizace miniaturních paměťových toroidních jader (o průměru mm) s pravoúhlou hysterezní křivkou průchodem elektrického proudu. Ty se sestavovaly do matic v nichž bylo zapojeno množství feritových jader.

p sdílení času byl v té době unikátním řešením a u zahraničních počítačů se objevil až později, většinou řešen pomocí e operačního systému. Pro tento přístup ke zpracování úlohy byl návrh počítače EPOS1 považován ve světě za špičkový.

Počítač EPOS1, na svoji dobu nesmírně pokrokový, přivedl celou řadu nových myšlenek, které byly komerčně realizovány v počítačích i daleko pozdějších generací. Měl děrnoštítkové periferní jednotky, magnetickou bubnovou paměť, magnetické páskové paměti a řádkovou tiskárnu. S ohledem na rychlý vývoj v oblasti polovodičů byl tento elektronkový počítač nahrazen tranzistorovou verzí **EPOS2** v následujícím roce (1964), která se později vyráběla a dodávala pod označením **ZPA601**.

S pokračující integrací zemí východního bloku do mezinárodní organizace RVHP[1] byl další rozvoj počítačových prvků a technologií velmi omezen a zaměřen na podporu výroby v oblasti definované programy SMEP[2] a JSEP[3]. Tehdejšímu Československu připadla výroba středních počítačů EC1021, EC1025, EC1027, malých počítačů SM3–20, SM4–20, SM52/11, SM50/50, SM50/40 a některých periferních zařízení. Základní jednotky středních počítačů se dodávaly ve třech podružných řadách R1, R2 a R3, které se odlišovaly výkonem, použitou technologií, kapacitou paměti a dalšími parametry. Např. počítač EC1025 byl vystavěn na bázi integrovaných obvodů střední a velké integrace s polovodičovými pamětmi. Měl modulovou strukturu a obsahoval několik specializovaných bloků pro zpracování dat a pro připojení vnějších zařízení. Jeho výkon byl 38 000 op/s a maximální instalovaná operační paměť 256 kB. Nejnižší typy počítačů řady SMEP používaly mikroprocesory typu Intel 8080 popř. 8086, avšak jejich architektura se od dnešních, mikroprocesory řízených PC, lišila.

2.8.1 Antonín Svoboda

Prof. Dr. Ing. Antonín Svoboda je označován za zakladatele české výpočetní techniky. Jeho Svobodovy mapy, používané pro minimalizaci logických funkcí při obvodovém návrhu jednotek počítačů se používaly a vyučovaly na celém světě. Antonín Svoboda[4] absolvoval strojní a elektrotechnické inženýrství na ČVUT a přírodovědeckou fakultu Univerzity Karlovy. Když byl roku 1936 jmenován doktorem technických věd, spolupracoval s armádou na zkouškách a vývoji zaměřovačů pro řízení protiletecké obrany. Po německé okupaci Československa nabídl Antonín Svoboda své služby Francii, kam zakrátko emigroval a pokračoval ve své práci u společnosti Société d'Applications Générales d'Electricité et Mécanique v Paříži. Po porážce Francie v červnu 1940 emigroval spolu se svou rodinou přes Maroko, Španělsko, Portugalsko a Kubu do USA, kde se brzy stal vedoucím inženýrem a vicepresidentem společnosti ABAX v New Yorku, která se měla zabývat výrobou zaměřovačů jeho systému. Výpočet polohy letounu ze zjištěných vstupních dat realizoval analogovým počítačem s mechanickými pákovými operačními prvky, jejichž teorii pro tento účel vypracoval a později též publikoval. Uvedené zařízení bylo chráněno i několika americkými patenty. Jelikož zakázka výroby zaměřovačů u firmy ABAX byla zrušena, odešel v roce 1943 do Radiation Laboratory of Massachusetts Institute of Technology a pokračoval zde ve vývoji analogových výpočetních zařízení pro zaměřovače k řízení palby. Zde byl návrh jeho zaměřovače dokončen, stroj postaven a Antonín Svoboda obdržel v roce 1946 vyznamenání ministerstva námořnictví USA za zásluhy o vítězství spojenců.

Do Československa se opět vrátil v roce 1946, byl opět jmenován asistentem matematického ústavu ČVUT a roku 1948 docentem. V rámci své pedagogické činnosti pořádal specializované přednášky o strojích na zpracování informací pro studenty strojní a elektrotechnické fakulty. Zhruba současně založil výzkumnou laboratoř v pražské pobočce Zbrojovky Brno a zahájil vývoj několika funkčních modelů reléových stolních kalkulátorů i složitějších matematických strojů. Za projekt kalkulačního děrovače mu byla roku 1955 udělena státní cena. V roce 1950 byl založen Ústřední ústav matematický a Antonín Svoboda byl jmenován vedoucím

[1] RVHP – Rada vzájemné hospodářské pomoci – mezinárodní ekonomická organizace států východního bloku, vznikla 8.ledna 1949.

[2] SMEP – systém malých elektronických počítačů, jednotná řada osmi a šestnáctibitových počítačů na bázi technologie integrovaných obvodů, vyráběná v rámci mezinárodních dohod RVHP. Většina typů této řady vycházela z v té době komerčně dodávané řady PDP–11 firmy Digital Equipment a v mnohém s ní byla kompatibilní.

[3] JSEP – jednotný systém elektronických počítačů, řada počítačů na bázi technologie integrovaných obvodů, vyráběná v rámci mezinárodní spolupráce RVHP. Jednotlivé typy vycházely z konstrukcí komerčně dodávané řady IBM360 a byly s ní plně kompatibilní.

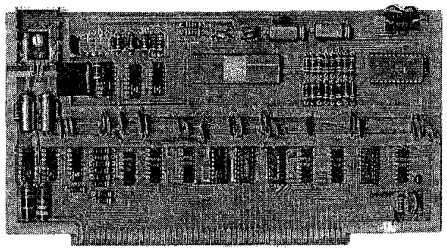
[4] Narozen 14. 10. 1907 v Praze, zemřel 18. května 1980 v Milwaukee, Oregon, USA.

oratoře matematických strojů. Zde začal pracovat na projektu velkého reléového samočinného počítače PO. Z původní Laboratoře matematických strojů vznikl roku 1958 Výzkumný ústav matematických strojů, Antonín Svoboda vedl oddělení výzkumu. Jeho koncepčním dílem byly velké samočinné počítače EPOS, ající elektronkových a tranzistorových logických prvků. V roce 1964 byl Antonín Svoboda nucen z itických důvodů opět opustit svou vlast a po příjezdu do USA pracoval ve výzkumném středisku firmy neral Electric v arizonském Phoenixu.

Po dvou letech byl povolán na kalifornskou univerzitu v Los Angeles, kde byl roku 1968 jmenován fesorem. Přednášel logický návrh, architekturu počítačů a počítačovou aritmetiku. V témž roce byl za své e v oboru samočinných počítačů poctěn čestným členstvím v Institute of Electrical and Electronic gineers (IEEE). Roku 1975 u příležitosti přednáškového turné v Paříži krátce a naposledy navštívil Prahu. agogické činnosti zanechal a odešel do důchodu v roce 1975.

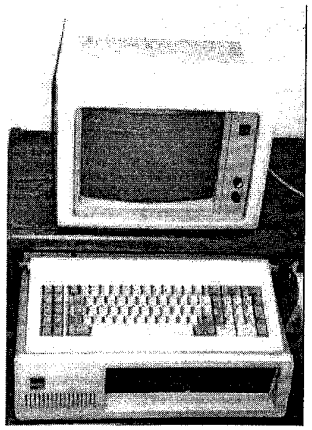
) Rozvoj mikroprocesorů a personálních počítačů

Do té doby byly počítače, většinou sálového typu, dostupné pouze armádě, univerzitám a velkým firmám, a k pro svoji cenu, tak pro nutnost nákladné údržby. V roce 1975 uveřejnil časopis "Popular Electronics" velký ek o "první stavebnici počítače, který soupeří s komerčně dodávanými typy - Altair 8800". Altair 8800 běný firmou Micro Instrumentation and mentry Systems (MITS) stál v té době asi 400 rů a tak byl relativně dostupný i amatérům. ak sestavení a spuštění Altairu nebylo zase tak oduché. Nový vlastník si jej musel sestavit, at první program – v té době nebyly žádné gramy komerčně dostupné. Program nesměl ročit velikost paměti počítače (256 bytů) a at do paměti počítače se musel ve strojovém u pomocí přepínačů na čelním panelu, které nastavovány na odpovídající kombinace pro ukce, data a operandy. Počítač Altair byl éze doplněn přídavnými jednotkami paměti, umožňovaly rozšíření až na 16 kB, připojovací jednotkou pro záznam dat na kazetový magnetofon a mi prvky.



Obr. 10 Procesorová jednotka počítače Altair

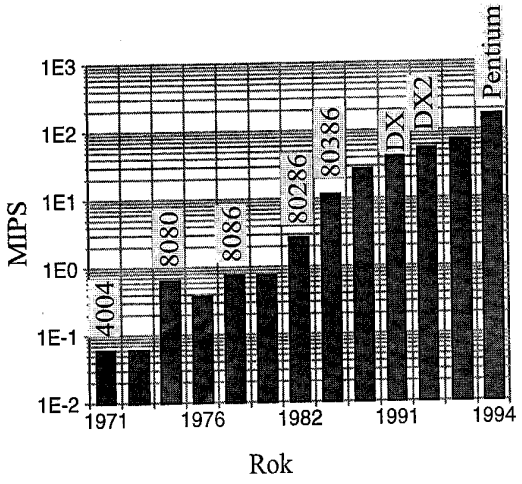
Uvedení stavebnice Altair spustilo trh personálních počítačů. V 1977 předvedl na veletrhu v San Franciscu Steve Wozniak a Jobs počítač Apple II se zabudovaným interpreterem Cu, barevnou grafikou a 4.1 kB paměti (1300 USD) a s ností uložení programu a dat na běžném magnetofonu. Síť ronických prodejen Radio Shack přispěchala s modelem TRS- terý ve svém inovovaném pozdějším provedení měl již 64 kB ěti. Jak TRS-80 tak i Apple II v té době byly již vybaveny vou pamětí.



Obr. 11 První počítač IBM PC

Firma IBM, která do té doby vyráběla pouze velké a střední ače se rozhodla vstoupit na trh malých počítačů s verzí Acorn, se později stala známou pod názvem IBM PC. Jednoduchá ální výstavba personálního počítače a příznivá cena přispěly k rychlému rozšíření. V době prvního uvedení na trh měl ční paměť 16 kB, klávesnici z elektrického psacího stroje IBM pojení ke kazetovému magnetofonu, použitému jako vnější ť (1300 USD). Současně s uvedením tohoto počítače uvolnilo i zdrojový kód BIOSU, který omylem nebo úmyslně nebyl tově chráněn. Takový návod k výstavbě jednoduchého ače nezůstal dalšími firmami opomenut a tak se objevila celá řada klonů – napodobenin, či kopií nálních počítačů. V roce 1984 přišly IBM a Apple s novými modely. Apple představil Macintosh, první l s grafickým uživatelským interface (GUI), který zjednodušil a zpřístupnil ovládání. IBM zase nabídl verzi 286-AT s aplikací Lotus 1-2-3 (tabulkový procesor) a Microsoft Word (textový procesor). Tím se stal favoritem pro administrativu i domácnost. Další vývoj v této oblasti byl již směrován dodavateli komponent – zejména dodavatelem mikroprocesorů firmou Intel. Vedoucí úloha v přípravě software se pomalu přesunula na firmu Microsoft, založenou jedním z dvojice autorů prvního interpreteru Basicu pro personální počítač Altair - Williamem Gatesem.

Postupné zvyšování nároků na personální počítače bylo od té doby řešeno převážně v oblasti nových technologií s drobnými úpravami organizace počítače. Nárůst výkonu mikroprocesorů od prvního reprezentanta této skupiny byl téměř exponenciální, jak je vidět z obrázku, který postihuje vývoj výkonnosti mikroprocesorů firmy Intel od prvního typu 4004 do uvedení procesorů Pentium (viz. Obr. 12). Současné mikroprocesory dosahují násobku výkonnosti sálových počítačů sedmdesátých let. Je to hlavně zásluhou rozvoje technologie, nových materiálů a postupů, které umožňují tak vysokou integraci na čipu, že rozměry jednotlivých součástek jsou srovnatelné s vlnovou délkou běžného světla a pro výroby těchto integrovaných obvodů je nutno používat speciální postupy. Uvedená strmá křivka vývoje však postihuje rozvoj v letech, kdy rozvoj technologií zaznamenával největší úspěchy. Se zmenšováním rozměrů jednotlivých elementů na čipu však narůstají výrobní problémy a je nutno hledat jiné cesty pro dodržení kontinuálního nárůstu vývoje výkonnosti prvků pro výstavbu počítačů.



Obr. 12 Nárůst výkonnosti mikroprocesorů firmy Intel

Novy system generuje nove problémy.
A. Bloch: "Murphy's Law"

3 Architektura, organizace a implementace počítače

Východiskem pro návrh počítače je celá řada poznatků z oblasti přírodních i humanitních věd. Můžeme je rozdělit na ty, které se návrhu přímo účastní - tedy vědy základní, mezi které můžeme zařadit filozofii, matematiku, logiku a elektroniku, a vědy, které jsou vodítky při koncových aplikacích nebo podporují výzkum nových technologií, tedy vědy podpůrné, kam patří např. ekonomika nebo biologie. Z tohoto hlediska vědy matematicko-fyzikální či v širším kontextu vědy přírodní, patří větší měrou mezi vědy základní. Z tohoto principu se může vymykat přínos věd podpůrných, které mohou být velmi důležité při objevování nových principů záznamu a zpracování informací a tak se v budoucnu mohou stát i podstatnou složkou při návrhu nových technologických konceptů počítačů.

Z hlediska postupu při návrhu počítače můžeme rozdělit celý proces do tří základních oblastí, z nichž každá z nich má svá specifika:

1. **Architekturu počítače**, která zahrnuje prvotní úvahy související s návrhem počítače a definuje vlastnosti počítače, které mají vliv na jeho programový model. Zjednodušeně se jedná o pohled na počítač z úrovně programátora.
2. **Organizaci počítače**, jež specifikuje vlastnosti počítače, které ovlivní potenciál navržené architektury. Opět můžeme zjednodušeně chápat tuto oblast jako popis logického seskupení a vzájemných vztahů mezi různými subsystémy navrhovaného stroje.
3. **Implementaci počítače**, zabývající se volbou vhodné technologie pro podporu navržené architektury a organizace. Implementace bude např. při realizaci počítače pomocí elektronických prvků zahrnovat i obvodový návrh.

Jednotlivé oblasti spolu vzájemně souvisí a na sebe navazují a výsledný návrh vzniká teprve jejich interakcí. Výše uvedený postup jako celek, všechny vyjmenované oblasti, obvykle nazýváme souhrnným termínem **návrh počítače - design**. Vzhledem k tomu, že první dvě uvedené oblasti se velmi intenzivně prolínají, považují se mnohdy termíny *organizace počítače* a *architektura počítače* za synonyma a označuje se jimi souhrnná oblast přípravy návrhu zahrnující jak návrh architektury, tak i návrh organizace. V tomto smyslu je nutno chápat i běžně používané termíny "architektura von Neumanna" nebo "hardwardská" či "princetonská architektura", které označují spíše organizaci navržených strojů než jejich architekturu ve výše uvedeném pojetí.

3.1 Architektura počítače

Jak již bylo uvedeno, architektura počítače spočívá ve specifikaci jeho programového modelu, nebo spíše prvků, ze kterých tento programový model bude vystaven. Jedním z nejdůležitějších bodů je prvotní definice způsobu **reprezentace dat** a **velikosti základních datových struktur**. Pod reprezentací dat přitom můžeme zjednodušeně chápat způsob nebo formu, jak budou člověkem vnímané entity uchovávány ve strukturách stroje. Vzhledem k potřebě minimalizovat potřebný prostor a tedy i omezit možnost atomizace jednotlivých dat je při stanovení jejich reprezentace rovněž definována minimální a maximální celistvá adresovatelná jednotka datové struktury.

Při stanovení reprezentace dat bude důležité hledisko realizace reprezentované entity a operace s touto entitou. Zvolíme-li např. jinou než binární soustavu pro reprezentaci čísel, pak budeme muset definovat jakým způsobem budou reprezentovány stavy v implementaci této soustavy – budou transformovány do binární logiky a realizovány jednotkovými elektrickými signály nebo použijeme víceúrovňový systém pro reprezentaci čísel[1] a navrhneme operátory pracující přímo v tomto víceúrovňovém systému? Svoji roli tedy sehrají i zvolená kódovací schémata a očekávaný způsob implementace navrhované architektury. Dnešní stroje jsou navrhovány s představou implementace v binární logice a základní metodou přístupu do prostoru, ve kterém jsou uloženy

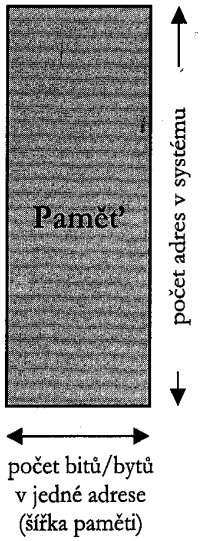
[1] Z hlediska teorie informace není dvojková soustava optimálním řešením. Nejefektivnější zápis čísla může být dosažen v soustavě se základem e (Eulerovo číslo = 2,718 ...).

formace, je **adresování**. Stanovení velikosti základních datových struktur se bude tedy soustřeďovat na novení násobku nejmenší jednotky – **bitu**[1], který bude samostatně přístupný v prostoru uložení informace – vykle paměti. V této souvislosti byla obecně přijata pouze jedna vyšší jednotka – **byte** – který je sloučením mi bitů. Vyšší používané jednotky jsou obvykle násobky bytů a označují se jako **slova** nebo jejich části či sobky – půlslova, dvojitá slova[2] apod. Slova a jejich násobky nejsou standardizovány a můžeme se setkat velikostí slova např. 12 bitů (Digital Equipment PDP-6), 16 bitů (Intel 8086), 32 bitů (Intel Pentium), 24 bitů urroughs B1700) nebo i 80 bitů (číslo v pohyblivé řádové čárce podle standardu IEEE 854). Uspořádaná nožina adresovatelných jednotek bude tvořit **adresový prostor**.

V této části návrhu se bude muset architekt systému rovněž zhodnout o způsobu jak bude k jednotlivým datovým ukturám přistupovat – o tzv. **adresovacích konvencích**, což očívá v definici položky v paměti počítače jako **adresy** a ve novení pravidel pro formátování adresy. Základem při tom je vrh formátu příkazů pro řízení počítače, kdy ke změně stavu chází na základě provedení elementární operace, – **instrukce**. instrukce je možno nahlížet jako na operátory zpracování dat, ré jsou rovněž uloženy v paměti (ať již stejné jako data nebo dělené paměti) a při předpokladu, že základní tvar instrukce de

OPERACE <operand 1> <operand 2> ...

nezbytné určit např. kolik bitů v instrukci bude věnováno na kódování příslušné operace, jaká část bude věnována na adresy, lik adres bude instrukce mít a jak budou formátovány. Zároveň de nutno zajistit aby adresy uvedené v poli operandů byly ikátní, nebo definovat jakým způsobem budou zpracovány, kud unikátní nebudou. Součástí popisu instrukce rchitektonickém modelu je, mimo formátu, i definice stavu oje před provedením instrukce a po jejím provedení.



Obr. 13 Velikosti základních datových struktur

Množina takto definovaných instrukcí bude tvořit instrukční abor, který bude obsahovat základní realizované operace. Doplněním o specifikaci interakcí mezi ováděnými operacemi, druhy adresování a reprezentací zpracovávaných dat vznikne architektonický model čítače. Ze způsobu konstrukce modelu a z jeho obsahu budou vyplývat různá omezení daná modelem, a to ména na

- instrukce a reprezentace dat,
- instrukce a adresovatelné jednotky.

o jednodušší popis modelu se často používá registrů, reprezentujících alternativní adresový prostor, výrazně nší než počet adres v hlavní paměti[3]. Rozeznáváme dva základní typy registrových modelů

- model používající pro popis architektury dvojici explicitní registr / implicitní registr,
- model používající pro popis architektury dvojici řídící registr / registr operandu.

V prvém případě registrového modelu jsou použité registry voleny podle toho, jakým způsobem k nim ize být přistupováno z úrovně programu. Ten registr, jehož obsah se mění jako důsledek akce programu nazýváme **explicitní registr**, zatímco registr, jehož obsah se mění během provádění operace a nemůže být přímo ovlivněn zásahem programátora označujeme jako **implicitní registr**. Druhý případ rozlišuje registry použité v modelu podle jejich funkce v systému. Registry, které budou podporovat základní funkce systému prohlásíme za **řídící registry** a registry sloužící jako pracovní registry programu pojmenujeme **registry operandu**.

Nezbytnou složkou návrhu architektury budou i úvahy o způsobu řízení práce systému a jeho chování v závislosti na okolí. Do této kategorie spadá definice mechanismů **přerušení** a spolupráce procesoru, paměti a vstupně výstupních zařízení. Pod přerušovacími mechanismy přitom chápeme reakci procesoru nebo jiné řídící jednotky na událost vzniklou mimo tuto řídící jednotku. Z hlediska způsobu vzniku přerušení můžeme rozlišit mezi

- **trapem**, což je přerušení vzniklé jako následek provádění sekvence instrukcí,
- **asynchronním přerušením**, nebo též vnějším přerušením, způsobeným jako důsledek příchodu specifického signálu z okolí procesoru.

Návrh architektury určuje zda bude použito přerušovacích mechanismů, kolik a jaké tyto mechanismy budou, které události způsobí trap a zda bude systém reagovat odlišně na trap a na vnější přerušení. Protože přerušení, jak již název sám upozorňuje, naruší kontinuitu programu požadavkem na přednostní vykonání nějaké funkce, bude rovněž nutno určit, jakým způsobem se provede návrat stroje do stavu před přerušením[1].

Samostatnou kapitolu návrhu tvoří **architektura vstupu a výstupu**, která determinuje metody a způsoby přenosu dat mezi procesorem a okolním prostředím. V této fázi je nutno stanovit zejména

- definice datových struktur přenášených mezi jednotlivými jednotkami (resp. mezi procesorem a okolím),
- způsob a formu identifikace vstupně/výstupních jednotek včetně stanovení případné podmnožiny operací, které se promítnou do příslušných instrukčních kódů,
- způsob a formu identifikace zdroje a cíle dat jakož i metodu arbitrace mezi požadavky z více zdrojů,
- umístění dat na jednotlivých paměťových zařízeních a
- způsob reakce na vzniklé chyby.

Architektura I/O[2] formuje předpoklady pro vytváření výkonných komunikačních struktur a je určující pro kompatibilitu s jinými systémy. Na této úrovni je nutno také položit základy pro budování případných paralelizmů ať už mezi procesorem a I/O nebo přímo v řídících strukturách procesoru.

3.2 Organizace počítače

Organizace počítačového systému je mapováním architektury do funkčních jednotek. Potenciál, který byl vytvořen při návrhu architektury, je nutno zužitkovat maximálně efektivním návrhem organizace stroje. V této fázi většinou vstupuje do hry též měřítko ekonomické, kdy s ohledem na aplikaci je posuzován parametr cena/výkon[3]. Aplikace, které budou nákladově citlivé mohou stejnou architekturu realizovat v organizaci a následně implementaci, která výrazně sníží výslednou cenu, zatímco parametrově citlivé aplikace mohou tutéž architekturu realizovat v organizaci rapidně zvyšující požadovaný parametr.

ákladní jednotka informace, odpovídá nejmenší možné změně. Název jednotky pochází z anglického "**bi**nary dig**it**".

anglické terminologii, která je obecně přejímaným názvoslovím celosvětově používaným, se označují jako např. **word**, **half-rd** nebo **double-word**.

ento přístup popsal poprvé J. von Neumann společně s A. Burksem a H. Goldsteinem v práci "Preliminary Discussion of the gical of an Electronic Computing Instrument" (1946)

[1] V rámci návrhu architektury obvykle definujeme různé stavy stroje, které odlišují způsob jeho práce a každý stav slouží k provádění dané množiny operací.

[2] Zkratka I/O odpovídá běžně užívanému anglickému termínu INPUT/OUTPUT (vstup/výstup). V české literatuře se také často používá odpovídající zkratka V/V. Obě tyto zkratky budeme nadále používat jako synonyma.

[3] Tento parametr se v anglosaské literatuře označuje **price/performance**.

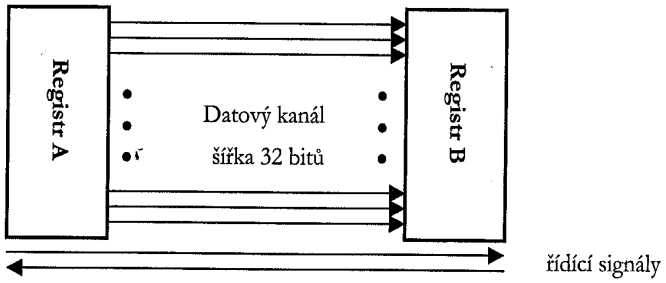
Poznámka

Při návrhu procesoru je použita architektura, jež akceptuje operace s čísly v pohyblivé řádové čárce. V cenově citlivé aplikaci určené pro běžné kancelářské zpracování malých objemů dat můžeme rozhodnout, že hardware procesoru bude přímo realizovat pouze operace s celými čísly a operace s čísly v pohyblivé řádové čárce budou emulovány prostřednictvím speciálních knihovních modulů, čímž se cena hardware prudce sníží. Naopak, bude-li se jednat o aplikaci, kde na prvním místě stojí rychlost výpočtu používajícího aritmetiku v pohyblivé řádové čárce, můžeme v organizaci navrhnout speciální jednotku, která bude realizovat potřebné operace přímo v hardware.

Vzájemné provázání organizace a implementace je zejména zřejmé v případě, kdy uvažujeme o budoucích změnách implementace. Z tohoto důvodu musí být organizace stroje dostatečně flexibilní, aby při změnách technologie nebo aplikačního zaměření docházelo pouze k minimálním změnám v organizaci počítačového systému a nevznikaly výrazné implementační problémy. Návrh organizace počítače zahrnuje mimo jiné

- návrh šíře datových cest,
- stupeň sdílení jednotlivých částí systému,
- definici specializovaných jednotek,
- návrh paralelizmů, popř. návrh realizace a podpory paralelizmů navržených v architektuře,
- organizaci paměti, vyrovnávacích pamětí a systému vstupu/výstupu,
- implementaci cílových funkcí a navržený stupeň predikce.

Je obvykle navrhován s předpokladem použití rámcově definované technologie při implementaci. Definuje proto tzv. **technologické plató**, což je obecný souhrn vlastností technologie, která bude při implementaci připravena pro použití. Důležité je, aby toto technologické plató bylo v použitelných mezích neměnné - tzv. interval stabilní technologie - a nedocházelo k oscilacím mezi návrhem organizace a implementace. Tedy výše uvedená flexibilita návrhu musí přiměřeně zaručovat absorbci pokroku v technologiích. Tím dojde k soustředění investic do výroby a ušetří se na nákladech souvisejících s častou změnou návrhu.



Obr. 14 Příklad datové cesty šířky 32 bitů

Šířkou datových cest rozumíme počet bitů, které je schopen komunikační kanál přenášet v jednom okamžiku. Nejmenší možná šířka datové cesty je tedy jeden bit. Šířka datové cesty obvykle v sobě nese i nároky na řízení přenosu dat, které je zajišťováno pomocí separátních řídících signálů. Ty jsou režijním nákladem přenosu a rozšiřují přenosové médium o nezbytné řídící kanály. Při výstavbě počítačů se širší přenosové cesty realizují jako uspořádané soubory přenosových médií, obvykle elektrických vodičů, které označujeme **sběrnice**. Datová **šířka sběrnice** udává celkový počet přenášených bitů v jednom okamžiku. **Kapacita sběrnice** je dána její fyzickou implementací, závisí na použité technologii a udává se v počtu jednotek informace, které je schopna sběrnice přenést za jednu sekundu (obvykle v byte/s). **Rychlost sběrnice** je kapacita sběrnice normovaná na jednotce informace, obvykle bytu.

Poznámka

Mnozí výrobci udávají místo rychlosti sběrnice její kapacitu, která je při širších sběrnicích větší a tak psychologicky vypadá nabízená sběrnice jako kvalitnější. Předpokládejme např. sběrnici se šířkou datových cest 32 bitů, která má kapacitu 120 MB/s. Znamená to, že sběrnice má rychlost 30 MB/s a při rozšíření na 64 bitů bude mít kapacitu 240 MB/s. Bude-li změnou technologie zvýšena rychlost sběrnice o 10 MB/s, zvýší se kapacita sběrnice na 160 MB/s.

Při hledání cest k minimální ceně se organizace počítače orientuje na vyhledávání možností sdílení těch částí počítače, které by byly realizovány několikanásobně, avšak jen velmi zřídka by byly využity současně. Přitom sdílení může probíhat na několika úrovních, které lze charakterizovat jako

- **sdílení na úrovni obvodů systému**, např. sdílení obvodů procesoru a obvodů I/O při nízkém poměru využití obvodů I/O
- **sdílení na úrovni organizačních jednotek**, např. sdílení registrů při zpracování instrukce.

Toto sdílení je časté zejména u specializovaných jednotek, např. mikroprocesor 3150 firmy Motorola používá jednu aritmeticko-logickou jednotku, která je sdílena třemi procesory – řadiči. Maximálního využití je dosaženo vhodnou volbou fáze hodinových pulsů, kterými jsou jednotlivé procesory řízeny.

Od sdílení částí počítače již není daleko k návrhu paralelizmů, procesů, které probíhají v počítači současně[1]. Z hlediska organizace počítače je důležitá **hloubka paralelního zpracování**, což je vlastně stupeň rozkladu úlohy na paralelně zpracovatelné části. Tato vlastnost úlohy, tedy možnost rozložit úlohu na menší úkoly se nazývá **granularita**. Úloha má větší granularitu, je-li možno ji rozdělit na více částí zpracovatelných paralelně. V zásadě lze hloubku paralelního zpracování chápat jako rozdělení na samostatné zpracování

- nezávislých úloh a programů,
- částí úloh a programů,
- podprogramů,
- cyklů a iterací a
- jednotlivých instrukcí.

Přitom je nutno si uvědomit, že úlohy, skupiny úloh nebo instrukce mohou být zpracovány paralelně jenom tehdy, pokud nejsou topologicky uspořádány. Paralelní zpracování pak může probíhat na různých typech specializovaných jednotek nebo na stejných typech jednotek. Velmi časté jsou kombinace paralelní práce specializovaných a obecných jednotek – např. procesoru a koprocesoru.

Organizace pamětí a vyrovnávacích pamětí spolu s přístupem procesoru a I/O je jeden z určujících parametrů výkonnosti počítačového systému. Hlavním úkolem vyrovnávací paměti mezi procesorem, pamětí a I/O, která se nazývá **cache**, bude vyrovnání rychlosti procesoru s připojenými jednotkami. Organizace musí nejenom stanovit, zda, kde a jak bude vyrovnávací paměť použita, ale určit i jaké budou stupně **paměťové hierarchie**, jakým způsobem bude probíhat předávání dat mezi jednotlivými paměťovými bloky a jak bude jejich spolupráce probíhat. Pro vlastní paměť pak bude důležité specifikovat odpovídající velikost paměti pro daný výkon procesoru, navrhnout rozložení paměti do fyzických jednotek, přidělit adresové prostory a stanovit topologii propojení pamětí a procesorů. Rovněž charakter paměti, způsob přístupu k uložené informaci a případné další charakteristiky předurčující použitou implementaci budou muset být v této fázi návrhu definovány.

Způsob jak budou organizována připojení vstupně/výstupních zařízení, jak budou rozděleny funkce mezi jednotlivé komponenty vstupně/výstupního systému a kolik může být takových zařízení připojeno je součástí návrhu organizace počítače. Pokud vstupně/výstupní zařízení budou vyžadovat vytvoření zvláštního adresového

[1] V tomto kontextu není zřejmý rozdíl mezi paralelním a distribuovaným zpracováním, který vyvstane teprve tehdy, když posuzovaný systém opatříme mírou rozlišitelnosti jednotek. Takto i dva procesy probíhající současně ve dvou linkách proudového zpracování instrukcí procesoru Pentium mohou být při vhodném měřítku rozlišení chápány jako distribuované procesy. Precizní rozlišení distribuovaného a paralelního zpracování a bližší analýza těchto procesů je však mimo rámec tohoto skripta.

rostoru, pak tak musí být učiněno již v návrhu architektury systému. Tato skutečnost vyplývá z nutnosti návrhu vláštních instrukcí, které budou manipulovat s daty právě v tomto adresovém prostoru, dedikovaném pro ařízení vstupu a výstupu. Procesor, který má vytvořen speciální adresový prostor pro práci s obvody ařízeními) vstupu a výstupu, přenáší data mezi zařízením vstupu/ výstupu tak, že zvláštní instrukce obsahuje dresu zařízení v tomto adresovém prostoru i adresu uložení informace v paměťovém prostoru procesoru – gistr, místo v paměti. Druhou možností je, že paměťový prostor bude pouze jeden a procesor bude anipulovat se vstupně/výstupním zařízením stejným způsobem jako při práci s položkou dat uloženou paměti. Tento způsob, kdy se jednotlivá zařízení vstupu/výstupu jeví jako součást hlavní paměti počítače ázýváme **mapováním vstupně/výstupního adresového prostoru** do hlavní paměti.

Predikcí nazýváme vlastnost jednotek počítačového systému stanovit nejpravděpodobnější další postup nnosti a učinit tak v předstihu kroky, které nejsou přímou součástí následující operace, ale připraví pro tuto peraci potřebné vstupní hodnoty. Tento proces se uplatňuje v moderních systémech a slouží ke zvýšení výkonu dnotek, které ji používají. Uplatnit se může nejenom při přístupu procesoru do paměti, ale také během racování instrukcí vstupu a výstupu. Většinou se používá tzv. **heuristická predikce**, což je způsob predikce isledujícího kroku založený na odhadu pravděpodobnosti vycházejícího ze statistiky podobných operací minulosti. Pro optimalizaci parametru výkon/cena bude nutno určit optimální stupeň a složitost predikce, kdy růst ceny při implementaci predikčních mechanismů bude vzhledem ke koncovým parametrům systému ještě ijatelný.

Organizace počítače navržená ve smyslu předcházejících odstavců bude očekávat, že na úrovni hardware ude vytvořena tzv. **cílová architektura** systému, tedy architektura, kterou bude systém poskytovat úrovni mpilátoru. Za tímto účelem bude možné ještě dále rozdělit proces vytváření cílové architektury a buď ji zcela alizovat prostřednictvím hardware[1] nebo posunout hranici mezi hardware a software níže a ponechat úrovni rdware pouze elementární funkce. Cílová architektura pak bude emulována pomocí speciálního software, tzv. **ikrokódu**. V tomto pojetí je na cílový program nahlíženo z hlediska mikroprogramu jako na data určená ke racování mikrokódem. Rozhodnutí o způsobu realizace cílové architektury stroje a správné nastavení dělící ry mezi hardware a software patří mezi stěžejní úlohy návrhu nového počítače a na něm mnohdy závisí mcové parametry a cena realizovaného systému. K úspěšnému rozhodnutí je nutno zasahovat do všech tří blastí návrhu stroje a velmi citlivě regulovat případné změny, aby sekundárně neovlivnily zcela jiné parametry. efinice technologického plató a tedy i způsobu implementace navržené architektury a organizace může tento oces velmi ovlivnit.

.3 Implementace počítače

Implementační zázemí pro realizaci navržených architektur se mění závratnou rychlostí. Podle některých hadů dojde ke změně technologického plató každých 18 měsíců, kdy se zdvojnásobí hustota transistorů nístěných na jednom čipu[2] (viz graf na obr. 15). Prudký nárůst hustoty integrace a výkonu, který je zřejmý z edeného grafu je poněkud zkreslující, neboť tak, jak se rozměry součástek blíží vlnovým délkám světla, echází vývojová křivka do ploché části s výrazně zpomaleným nárůstem parametrů součástek. Současné stroje alizované na bázi elektronických součástek jsou však stále schopny poskytovat vyšší výkony, i když časový růst výkonu je pomalejší. Přesto, nebo spíše právě proto, se v laboratořích předních univerzit a světových obců připravují technologie, které by měly výrazně změnit tvář počítačů v budoucnosti.

Pokud předpokládáme elektronickou realizaci systému (nikoliv např. kvantovou nebo biologickou), pak žeme odlišit historicky několik základních úrovní stavebních prvků, rozdělených podle počtu základních ementů např. tranzistorů:

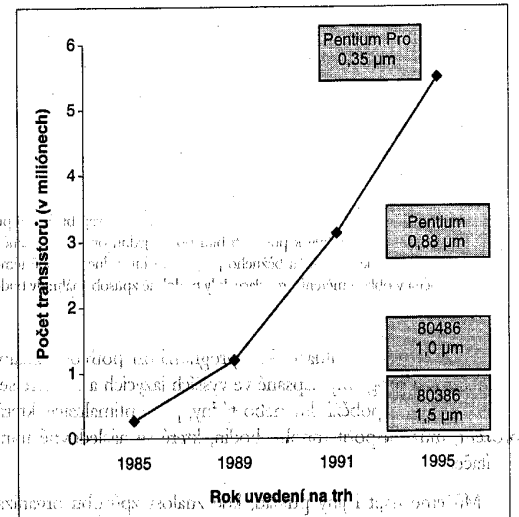
- **hradla** - základní a nejjednodušší stavební jednotka v technologii integrovaných obvodů. Obsahují řádově desítky transistorů.
- **kombinace hradel** - obsahují nejrůznější kombinace jednoduchých hradel např. registry, kodéry, multiplexery apod. Tyto stavební jednotky mohou obsahovat až tisíce transistorů.
- **funkční jednotky** – sestavy obvodů integrovaných v jednom čipu, na jednom substrátu či jedné podložce, vykazující samostatnou funkci uvnitř systému a s minimálními nároky na okolní podporu. Obsahují stovky tisíc až milióny tranzistorů a patří mezi ně např. specializované koprocesory nebo řadiče.

Toto dělení nijak nevystihuje historický postup integrace a je velmi všeobecné. Můžeme očekávat, že další, stále výkonnější prvky, se budou stále objevovat na trhu. Přestože mnozí výrobci ještě před několika lety předpokládali nemožnost realizace submikronových technologií, dnešní mikroprocesory tuto hranici snadno překonaly[1].

V budoucnu lze očekávat realizaci ultrarychlých procesorových jednotek tzv. kvantových počítačů, jejichž základní jednotky - kvantová hradla, byly již sestrojeny hned v několika laboratořích. Logika kvantových počítačů však je výrazně odlišná od klasické binární logiky současných konstrukcí. Vzhledem k tomu, že poloha částice je udávána spíše jako pravděpodobnost její polohy než neměnná souřadnice, stavový prostor kvantového počítače je daleko rozměrnější než stavový prostor klasického elektronického nebo mechanického stroje. Kvantový počítač může existovat ve více stavech v jednom okamžiku, což rozvíjí obrovskou příležitost pro realizaci paralelních architektur. Poznatky z laboratoří však vedou k tomu, že chování kvantových počítačů bude spíše směřovat k "analogové simulaci" problému, tedy k jeho spojitému řešení na bázi modelování kvantových systémů, než k jeho řešení digitální aproximací.

Rozvoj v oblasti paměťových médií již zaznamenal zvýšený pohyb směrem k používání optických nosičů. Nastoupený směr signalizuje možnost realizací holografických pamětí, které využijí pro uložení informací celou hmotu paměťového média; teoreticky předpokládaná možnost je hustota až 1 TB/cm³. Výhodou je, že taková média neobsahují pohyblivé části a přístup k datům může probíhat paralelně. Rychlost přenosu řádu 1 Gbit/s z jednoho záznamového média, může přispět i k použití holografických pamětí při realizaci hlavních pamětí počítačů.

Nejenom fyzikální principy jsou však středem zájmu. Biologické látky, zejména proteiny jsou předmětem intensivního studia s ohledem na jejich použití jako paměťová media i jako prvky pro výstavbu procesorů. Pokusy zatím ukázaly teoretickou možnost hustoty záznamu až 0,33 TB/cm³ s rychlostí přístupu až 80 Mbit/s.



Obr. 15 Vývoj počtu transistorů v mikroprocesorech Intel

Studie technologických možností ukazují, že jejich další rozvoj může mít vliv nejenom na zvětšení výpočetní kapacity nových počítačů, ale i na způsoby a metody programování takových strojů. Použití kvantových počítačů, vzhledem k jejich odlišnému způsobu práce by znamenalo generační změnu v architekturách a stávajících přístupech k software. Rozšíření holografických pamětí by mohlo urychlit nástup nových metod uchovávání a publikace informací, což by se v důsledku projevilo i ve vynucených změnách přístupu ke jejich třídění a distribuci.

Obvykle je označována jako "hardwired architecture".

Tento jev je někdy označován jako Mooreovo pravidlo podle Gordona Moore, který tento trend poprvé vysledoval na počátku ndesátých let.

[1] Jednalo se o technologie používající fotolitografických metod, které používaly fotocitlivých materiálů k přenosu masky na vlastní čip. Rozměry prvků masky začaly dosahovat rozměrů srovnatelných s vlnovou délkou světla, což znemožňovalo další vývoj této technologie. Nové technologie však umožnily překonat tuto obtíž a např. procesory Pentium používají technologie 0,35 mikronu a na trhu se objevují první součástky s technologiemi 0,25 a 0,18 mikronu.

.4 Programování

Za praotce počítačů moderní doby se považuje počítač ENIAC, jehož programování se provádělo řepojováním kabelů mezi jednotlivými jednotkami. S postupem času se ukázalo jako výhodné umístit program o paměti a mít tak snadnou možnost modifikace algoritmu, který byl pomocí programu předán stroji. leobyčejně těžkopádné programování pomocí zápisu strojových instrukcí vedlo k vývoji programů, jež dokázaly řevést symbolické zápisy instrukcí ve srozumitelné formě převést do binárního obrazu zaveditelného do paměti roje. Tyto programy, dnes známé pod názvem asemblery, byly spolu s rozvojem v oblasti operačních systému ikladem k přechodu k sofistikovanějším řešením – vyšším programovacím jazykům.

Výhody programování ve vyšších programovacích jazycích zahrnují zejména abstraktnost a přenositelnost. **bstraktnost** znamená, že programátor, který popisuje algoritmus ve vyšším programovacím jazyku není závislý a tom, jaké detailní údaje bude potřebovat pro svoji práci počítač, který bude tento algoritmus vykonávat. **řenositelnost** je vedlejším produktem této abstrakce, který umožňuje, aby stejný program mohl být spuštěn na ızných počítačích, pokud zde budou existovat kompilátory, které přeloží program pro každý tento stroj.

Pro mnoho situací, se kterými se můžeme setkat při programování ve vyšších jazycích, je realita velmi blízká eálu. Kompilátory pro vyšší jazyky generují efektivní a přenositelné kódy pro typické počítačové systémy, :ogramátoři mohou používat zápisy ve vyšších programovacích jazycích a předpokládat, že jejich programy ›běží stejně efektivně na téměř jakémkoliv stroji. V dnešní době lze těžko očekávat zápis celého programu asembleru pro složité víceprocesorové stroje, avšak jistě bude několik významných míst např. v programech dících automatů, které se programátor rozhodne zapsat v asembleru pro zvýšení efektivnosti práce programu. zřejmé, že v případě, kdy část nebo celý program bude zapsán v asembleru, dochází k výrazné odchylce od ›strakce, protože asembler je jazykem stroje, nikoliv jazykem aplikace a bude velmi obtížně přenositelný z dnoho stroje na druhý.

Poznámka

Jako jednoduchý příklad použití podprogramu v asembleru předpokládejme úsek programu, který násobí hodnotu **x** dvěma. Zápis této operace např. v jazyce C bude **2*x**. Avšak programátor, který zpracovává algoritmus, může vědět, že konkrétní stroj má velmi pomalou instrukci násobení. Zároveň je mu známo, že celá čísla jsou v tomto stroji reprezentovány v binárním obraze. Pro urychlení práce programu pak může použít výrazu pro operaci binárního posunu vlevo **x << 1**[1], který je v binární aritmetice ekvivalentní násobení dvěma. Výsledný program pak bude méně abstraktní než původní, protože jedna z jeho operací je definována na úrovni strojového výrazu (posunutí binární hodnoty doleva o jeden bit) místo matematického výrazu (násobení). Celý program bude tedy přenositelný jen velmi omezeně, protože jej bude možno použít pouze na počítačích, které používají binární reprezentaci celých čísel a bude efektivní pouze na strojích, které potřebují k posunutí bitů pouze jednu operaci, výrazně kratší než operace násobení celých čísel. I když tento příklad je z hlediska běžného programování v dnešní době téměř nesmyslný, přesto se mohou nalézt aplikace, zejména v oblasti měření a regulace, kdy podobné způsoby náhrady budou použitelné.

Toto je typická situace, kde programátoři potřebují znalost počítačového systému k tomu, aby mohli ›timalizovat programy napsané ve vyšších jazycích a ve které se velice často nacházejí systémoví programátoři. stliže program poběží dny nebo týdny, pak optimalizace, která zrychlí provádění programu pouze o několik ocent, může uspořit mnoho hodin, které se následovně transformují do skutečné úspory strojového času ›čítače.

Můžeme najít i jiný příklad, kde znalost způsobu organizace a architektury systému může být důležitá. 'edstavme si počítač s velmi vysokým výkonem, který používá několik procesorů. V tomto případě nemusí být ›mpilátor vždy připraven efektivně přeložit předložený program bez alespoň malé pomoci od programátora. ıková situace může nastat např. u cyklu zapsaného v jazyce FORTRAN, který, když je napsán se znalostí stému, může být přeložen do jedné instrukce vektorového procesoru[2].

'osunutí o jedno binární místo doleva.

'ráce vektorových procesorů je podrobně popsána v kapitole 6.5.2.

Poznámka

K tomu, aby kompilátor byl efektivně napsán, je potřeba řada zkušeností s daným strojem tak, aby autor kompilátoru byl schopen využít všech charakteristik stroje a díky této znalosti vybudovat optimální verzi kompilátoru. Například techniky pro optimální mapování nezávislých částí paralelních programů, které mohou být prováděny současně v různých uzlech paralelního procesoru, jsou jistě metody vedoucí k efektivnímu způsobu výpočtu, ale bez znalosti, jakým způsobem jsou jednotlivé procesory propojeny a jakým způsobem jsou předávány zprávy mezi nimi, nebude možno zajistit kvalitní a efektivně pracující návrh kompilátoru.

3.5 Základní prvky počítače

Základními komponenty typického počítačového systému jsou **procesor, paměť, zařízení vstupu a výstupu, a komunikační kanály**, které tyto jednotky propojují. Řídícím prvkem celého systému je **procesor**, který realizuje zadané úlohy tak, že provádí aritmetické nebo logické operace s daty. Je to jediný element, který vytváří novou informaci kombinací stávajících informací nebo jejich modifikací. V typickém systému bývá pouze jeden procesor známý jako centrální procesorová jednotka (CPU). Moderní výkonné systémy, např. vektorové procesory nebo paralelní procesory, mívají často více než jeden procesor. Systémy, které mají jenom jeden procesor pro řízení své činnosti, se nazývají **sériové procesory**, nebo jsou spíše známy pod názvem **skalární procesory**.

Paměť je pasivním komponentem systému, který zaznamenává a ukládá informaci do té doby, než je požadována jinou částí systému. Informace je v paměti uspořádána a je k ní přistupováno prostřednictvím adresy této informace. Při normálním provozu čte procesor instrukce z paměti, nebo ji používá pro zápis a uložení dat. Na druhé straně může být paměť zdrojem nebo cílem dat přenášených vstupními a výstupními zařízeními. Z hlediska programovacích jazyků bychom mohli na paměť nahlížet jako na jednorozměrné pole. Požadavek procesoru na paměť pak může být formulován jako "pošli instrukci z místa **M[1000]**" nebo požadavek řadiče diskové paměti "ulož následující blok dat do míst počínaje adresou **M[0]** až **M[255]**".

Vstupní a výstupní zařízení přenášejí informaci bez její změny[1] mezi okolím a jedním nebo více vnitřními komponenty systému. Mezi vstupní nebo výstupní zařízení mohou patřit např. sekundární paměti jako disky nebo pásky, popřípadě zařízení používané pro přímou komunikaci s uživatelem - jako displeje, klávesnice nebo myši.

Komunikační kanály, které vzájemně propojují jednotlivé jednotky systému, mohou být buď jednoduchá spojení mezi dvěma zařízeními, nebo velmi komplikované přepínače, propojující několik komponent systému a umožňující kterékoliv dvojici zřízení komunikovat v kterémkoliv čase. Konfigurace těchto přepínačů jsou mnohdy velmi složité a uplatňují se u velmi výkonných multiprocesorových systémů.

3.6 Formální popis počítačového systému

Snaha popsat počítačový systém formálním zápisem vyústila do formulace popisného systému známého jako **PMS** (**P**rocesor **M**emory **S**witch). Tento systém je dosti přesný a přehledný a umožňuje formalizaci až na úroveň popisu vztahu instrukce a procesoru. Při záznamu jsou používány následující označení:

M = pro paměť, která slouží k uložení informace. Je to pasivní zařízení, které netransformuje, ani nemění informaci. Adresovací systém umožňuje přístup k základním jednotkám paměti, které mohou provádět dvě možné operace - uložení a vyzdvižení informace.

L = propojení, které přenáší informaci z jednoho místa na druhé, aniž by došlo k jakékoliv změně této informace s výjimkou toho, že se změní její umístění. Každý link je ukončen bránou pro vstup nebo výstup informace. Stejně tak každý link má definovanou svoji šířku, která udává, jak velké množství informace může být přeneseno najednou.

[1] Je nutno chápat jako "bez změny charakteru přenášené informace", nikoliv jako "bez změny její reprezentace" - tedy např. změna polohy se přenáší ve významu změny polohy.

K = řízení způsobující změny stavu ostatních komponentů v systému. Protože s výjimkou procesoru všechny komponenty jsou pasivní, je vyžadován aktivní agent (K), aby došlo ke spuštění jejich definovaných aktivit.

S = přepínač buď umožní nebo naopak zabrání propojení datových cest mezi dvěma připojenými komponenty systému. Má dva možné stavy, buď je cesta spojena nebo rozpojena.

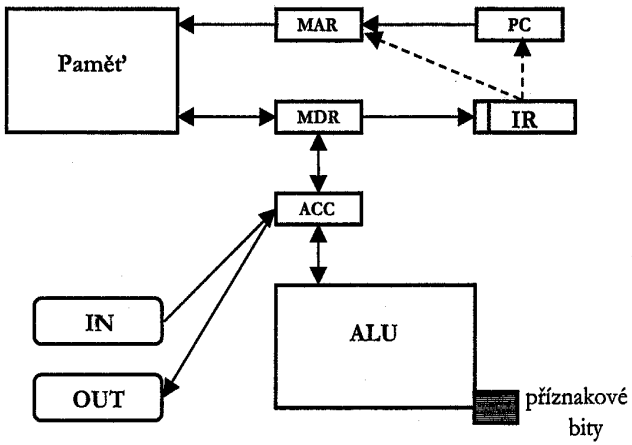
T = transdukter mění reprezentaci informace bez toho, aby došlo ke změně jejího významu. Je důležité si uvědomit, že transdukter zachovává význam, čili sémantiku informace, avšak nezbytně nemusí zachovávat tuto informaci v její původní formě. Příkladem může být převod sériových bitů na bity paralelní.

D = operace s daty vytváří informaci, která má nový význam. Příkladem mohou být aritmetické operace, posuvy nebo logické operace. V kontrastu s předcházejícím transdukterem, při operacích s daty dochází ke vzniku nové informace, zatímco transdukter pouze mění způsob kódování významu informace.

P = procesor je prvek, který je schopný interpretovat program tak, že vykonává sekvence operací, tj. že převádí systém nebo subsystém z jednoho stavu do jiného stavu. K tomu je potřebná množina operací a nezbytné řízení, aby instrukce mohly být přečteny z paměti a interpretovány. Poznamenejme, že **P** se skládá z **K** a **D**, popř. **T**, a že tam, kde je použit **P**, nebude nutno specifikovat **K** a **D**.

C = pro náš účel postačí, když symbol **C** bude označovat program uložený v počítači.

X = okolní prostředí, tedy např. displej počítače, klávesnici, tiskárnu, disk a jiné.



Obr. 16 Schéma struktury počítače se vstupy a výstupy prostřednictvím akumulátoru

Pro základní orientaci ve formálních záznamech PMS uveďme několik příkladů. Na obr. 16 je zobrazena uktura počítače, jehož vstup a výstup je realizován prostřednictvím akumulátoru. V tomto případě není ožné, aby současně byl realizován výpočet a operace vstupu nebo výstupu. Tato struktura může být v notaci MS zaznamenána jako

C:= M — K — D — T — X

Pro označení prvků ve schématu jsou použity následující zkratky, které se budou v následujícím textu skytovat i nadále:

MAR – (**M**emory **A**ddress **R**egister) registr, do kterého se ukládá adresa na kterou se má zapsat nebo ze které se má číst

MDR – (**M**emory **D**ata **R**egister) registr, ve kterém jsou uložena data, která jsou z paměti přečtena nebo která se mají do paměti uložit

PC – (**P**rogram **C**ounter) programový čítač, registr ve kterém je uložena adresa následující instrukce

IR – (**I**nstruction **R**egister) instrukční registr ve kterém je uložena právě zpracovávaná instrukce

IN – (**I**nput) brána vstupního zařízení

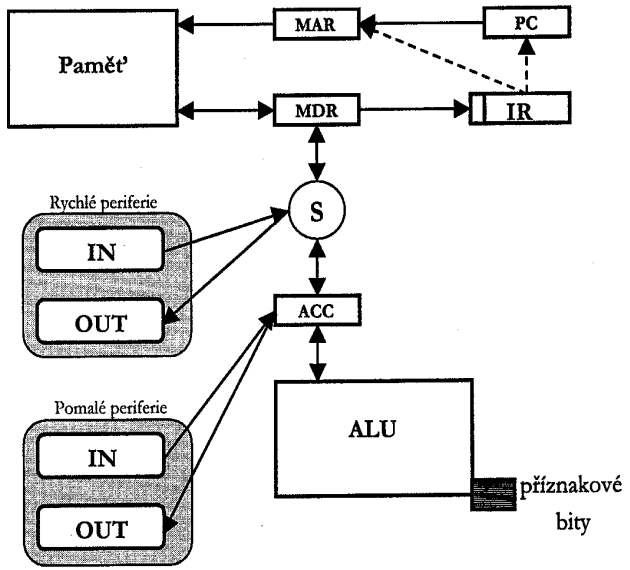
OUT – (**O**utput) brána výstupního zařízení

Protože procesor se skládá z komponentů **K** a **D**, je možno výše uvedený nápis zápis zkrátit do tvaru

C := M_p — P_c — T — X

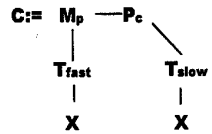
kde P_c označuje centrální procesorovou jednotku a M_p primární paměť.

Modifikujeme-li schéma struktury tak, že vstupy a výstupy jsou připojeny přímo do paměti a nikoliv přes akumulátor, obdržíme blokové schéma dle obr. 17, kde zkratka S označuje "switch" - přepínač. V této struktuře je možno provádět pouze omezené současné operace výpočtu a vstupu a výstupu, neboť obě jednotky, jak vstup a výstup, tak i aritmeticko logická jednotka používají stejné registry přístupu do paměti. Toto je úzké místo této organizace, neboť pro akvizici nezbytné informace z paměti nebo pro uložení potřebných informací do paměti existuje pouze jedna cesta a obě jednotky, jak ALU tak I/O používají stejné registry MAR a MDR. Pomalá zařízení vstupu a výstupu jsou obsluhována stejně jako v předchozím případě prostřednictvím akumulátoru.



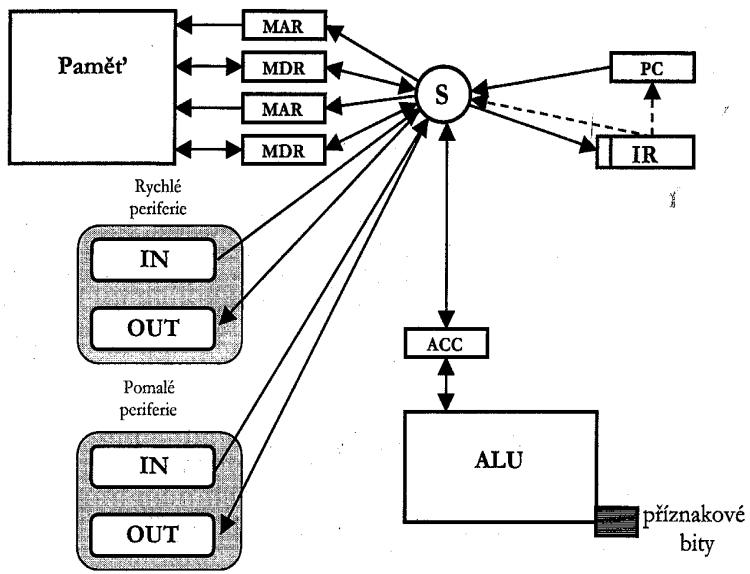
Obr. 17 Modifikovaná struktura počítače s jedním rychlý a jedním pomalým kanálem vstupu a výstupu

Uvedenou strukturu můžeme v notaci PMS zapsat jako



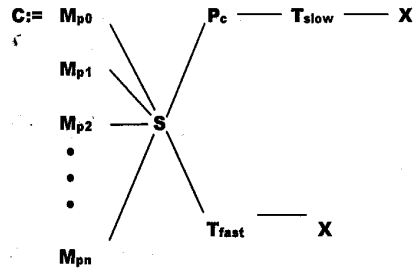
Abychom odstranili problém jediného přístupu k operační paměti, opravíme předcházející strukturu doplněním dvojice paměťových registrů a obdržíme složitější diagram dle obr. 18. V tomto případě navržená

organizace umožňuje současné provádění výpočtů i vstupně výstupních operací, avšak je třeba použít specifické prostředky, aby nedošlo k současnému přístupu k téže buňce paměti. To bude znamenat, že obsah dvou adresových registrů nesmí být současně stejný. Jedna možnost praktického řešení tohoto problému je rozdělení pamětí do částí, kde každá tato část bude obsahovat svůj vlastní pár MAR/MDR.

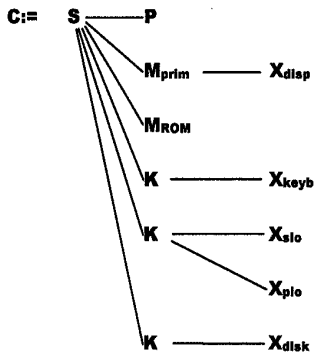


Obr. 18 Struktura počítače s vícenásobným přístupem do paměti

Zaznamenáme-li v notaci PMS strukturu tohoto počítače s rozdělenou pamětí, který umožňuje simultánní práci ALU a vstupně-výstupních zařízení, pak dojdeme k zápisu



Dekompozicí jednotlivých částí systému můžeme získat popis pro prakticky jakoukoliv počítačovou strukturu. Jako příklad lze uvést zjednodušený PMS diagram známého stolního počítače Apple Mackintosh. První, co můžeme zaznamenat na tomto diagramu je jednoduchý komunikační kanál, známý jako sběrnice, který propojuje všechny hlavní komponenty systému, viz obr. 19. Protože sběrnice má charakter přepínače, pouze dvě komponenty systému mohou spolu komunikovat v daném čase. Je-li přepínač konfigurován pro přenos ze vstupně-výstupního zařízení, např. z disku do hlavní paměti, procesor nebude schopen vyzvednout data nebo instrukce a musí zůstat ve stavu nečinnosti. Tato organizace je typická pro personální počítače nebo pracovní stanice. Výkonné počítače nebo superpočítače většinou mají daleko bohatší struktury, propojující vstupně-výstupní zařízení a procesory k paměti, které umožňují procesorům bez omezení pokračovat v práci souběžn s prací vstupně/výstupních zařízení.



Obr. 19 Zjednodušený PMS diagram počítače Apple

Nerozhodnost je předpokladem přizpůsobivosti.
A. Bloch: "Murphy's Law"

4 Reprezentace dat

Informace v počítači je uložena v paměti, směrována a přenášena pomocí komunikačních kanálů transformována procesorem. Pro všechny tyto operace musí existovat nějaká fyzikální entita, která bude mít pr stroj odpovídající význam. Pro člověka je taková entita reprezentována jeho nehmotnou představou neb symboly, které pro záznam těchto představ používá. Pro stroj však bude nutno zavést takovou symboliku, kter bude

- srozumitelná pro stroj,
- snadno reprezentovatelná pomocí fyzikálních veličin,
- vhodně převoditelná do symbolů srozumitelných člověku a
- dostatečně obecná pro záznam i složitých nehmotných představ.

Tato symbolika neboli reprezentace dat patří k nejdůležitějším oblastem návrhu architektury stroje. Každ použitý symbol musí být odpovídajícím způsobem kódován tak, aby jeho kód reprezentoval logický význa symbolu a tento význam byl jednoznačný. A protože tato informace ve formě symbolů je počítače manipulována a ukládána do paměti, musí existovat uvnitř počítače nějaká instance, kterou pojmenujeme **datov typ**. Tento datový typ lze popsat jako

1. buňku dat, ve které bitový obraz kódu jednoznačně kóduje entitu, která je zde uložena (přísluší vlastnostem **syntaxe** datového typu),
2. omezení možných hodnot, které mohou být v buňce uloženy (přísluší k **sémantickým** vlastnoste datového typu),
3. soubor pravidel určujících jaké transformace jsou s tímto datovým typem povoleny s ohledem n výsledek těchto transformací a
4. soubor pravidel definujících způsoby a oprávnění přístupu k těmto datům.

Každý datový typ musí explicitně nebo implicitně obsahovat veškeré výše uvedené charakteristiky. Podle typ stroje a druhu programového vybavení se na jedné straně mohou vyskytnout datové typy, které explicitn deklarují všechny potřebné informace, zatímco jiný typ stroje deklaruje pouze první vlastnost a zbývající jso akceptovány implicitně.

Historicky i z implementačních důvodů však ne všechny stroje akceptují u datových typů soubor pravid popisující způsoby a oprávnění přístupu k datům stejně jako množinu povolených transformací. Takto je ješ stále možno provádět ilegální operace s daty, jejichž obsah by to nedovoloval, např. aritmetické operace se znak nebo použít datum jako ukazatel do paměti.

Pro reprezentaci dat ukládaných do paměti budeme tedy používat kódy, které se budou lišit podle toh které hodnoty a jak budou tímto kódem reprezentovány. V zásadě lze oddělit dvě velké skupiny reprezentovan kódy - literály, kam patří např. logické hodnoty, znaky abecedy nebo grafické symboly a čísla.

4.1 Logické hodnoty

Logická hodnota, obsahující binární údaj, je většinou reprezentována jako číslo. Záleží pak na návrhu logi stroje i kompilátoru, zda bude jedna logická proměnná uložena v jedné datové položce, např. bytu, nebo zc bude reprezentována pouze jedním bitem v datové položce. Ve druhém případě lze zbylé bity buď zanedb nebo je použít pro uložení hodnot ostatních logických proměnných. Extrakce logické proměnné z dato položky v paměti je však obvykle časově náročná a proto se dává přednost úspoře času uložením logický hodnot do nejmenšího datového typu, před úsporou místa v paměti. U současných strojů se logická hodno obvykle ukládá jako jeden byte, kde jakákoli nenulová hodnota bytu odpovídá pravdivé logické hodno (**TRUE**) a nulová hodnota je reprezentantem opaku (**FALSE**). Pro logické hodnoty je obvykle definová množina booleovských operátorů.

4.2 Znaky

Reprezentace alfanumerických – znakových, položek se po původní značné roztříštěnosti v kódech standardizovala a zúžila na několik nejčastěji používaných kódů. Každý takový kód je omezen délkou datového typu, např. použijeme-li šest bitů, můžeme reprezentovat maximálně 64 různých znaků. Během času se ustálila délka skupiny osmi bitů - byte, což dává možnost uložení 256 znaků. V současnosti existují dva nejrozšířenější standardy

EBCDIC – Extended Binary Coded Decimal Interchange Code

ASCII – American Standard Code for Information Interchange

Základem kódu **EBCDIC** je kód **BCD** (Binary Coded Decimal), což je čtyřbitový váhový kód vyjadřující desítkové číslice. Jeho rozšířením na 8 bitů vznikl kód **EBCDIC** (Extended Binary Coded Decimal Interchange Code), který obsahuje skupiny znaků, jako desítkové číslice, malá a velká písmena anglické abecedy, pomocné znaky (**[ ] $ & *** atd.) a řídící znaky např. **CR** - Carriage Return (návrat vozíku psacího stroje), **FF** - Feed Form (nová stránka), **VT** - Vertical Tabulation (vertikální tabelátor), stejně jako oddělovače dat např. **SOH** - Start of Heading (začátek záhlaví), **STX** - Start of Text (začátek textu) nebo **ETX** - End of Text (konec textu).

Kód **ASCII** (American Standard Code for Information Interchange) obsahuje téměř stejné skupiny znaků jako **EBCDIC**, ale používá pouze sedmibitového prostoru, který je však zcela využit. Přidáním osmého bitu lze rozšířit počet kódů na 256 a tak vznikají různé znakové sady. Pro českou abecedu jsou se tak můžeme setkat např. kódy označenými KOI8-čs, EAST 8, ICL, PC std., kód Kamenických, Latin-2, nebo 1250 Microsoft Windows.

Pro Českou republiku je závazný kód Latin 2 jak jej definovaly mezinárodní standardizační organizace ISO (International Organization for Standardization) a IEC (International Electrotechnical Commission) jako jednu z norem pro mezinárodní abecedy[1]. Pro oblast Evropy je definována Latin Alphabet No.1 pro západní Evropu a Latin Alphabet No.2 pro východní Evropu. Modifikace Latin 2, která se však od standardu mírně liší, je známa jako kódová stránka 1250 pro Microsoft Windows. Firma Microsoft iniciovala vznik šestnáctibitového kódu Unicode, čímž se významně rozšířil počet uložitelných znaků na 64k. Kódovací tabulky pro některé z výše uvedených standardů jsou uvedeny v přílohách.

4.3 Číselné soustavy

Potřebujeme-li označit nějaké množství nebo provést aritmetickou operaci, obvykle zvolíme automaticky dekadickou soustavu. Je téměř jisté, že prapůvodním důvodem je existence deseti prstů na lidské ruce, prvotní početní pomůcky. Avšak někdy můžeme zaznamenat i používání jiné soustavy než dekadické. Příkladem může být např. dvanáctková a dvacítková soustava pro původní anglická platidla - šilink a penci, nebo šedesátková soustava používaná pro záznam minut a sekund. Pro zobrazení číslic v počítači se používají nejrůznější kódy, jejichž konstrukcí a studiem se zabývá teorie kódů.

Zcela zvláštní postavení v teorii kódů mají **číselné soustavy**. Čísla v číselných soustavách zaznamenáváme pomocí abecedy znaků - **číslic**, z nichž sestavujeme **číselné obrazy**. Pro hodnocení číselných soustav v souvislosti s jejich použitím v počítači je důležité, jak odolné jsou proti chybám, jak obtížné jsou algoritmy pro realizaci aritmetických operací s příslušnými číselnými obrazy a jak pracně se číselné obrazy jedné číselné soustavy převádějí na odpovídající číselné obrazy druhé číselné soustavy. Příkladem převodu číselného obrazu může být konverze binárního čísla, použitého v aritmetické operaci prováděné procesorem počítače na desítkové číslo určené k výstupu na uživatelský terminál. Pro výstavbu počítačů se nejčastěji používá dělení číselných soustav na soustavy **polyadické** a soustavy **nepolyadické**. Mezi polyadické soustavy patří např. desítková soustava, z nepolyadických soustav je pro aplikace ve výpočetních systémech důležitá soustava zbytkových tříd.

[1] Jedná se o normu ISO/IEC 8859:"Information Processing – 8-bit single byte coded graphic character set"

4.3.1 Polyadické číselné soustavy

Polyadická číselná soustava[2] má definovaný jeden základ nebo více základů **z**. Může tedy jít i o soustavu s smíšenými základy. Z polyadických soustav s jedním základem je nejznámější soustava desítková. Čís v desítkové soustavě vyjadřujeme jako součet mocnin deseti vynásobených součiniteli, které mohou nabýv hodnot 0,1,2,3, ... ,9 jež nazýváme číslicemi.

V polyadické soustavě s jedním základem definujeme **z**-adické číslice a_i , $0 \le a_i < z$, kde nejvyšší čísli $a_{max} = z - 1$. Obrazem čísla A

$$A = \sum_{i=-m}^{n-1} a_i \, z^i \qquad \text{R - 4-1}$$

je tedy uspořádaná k-tice $a_{n-1}, a_{n-2} \ldots a_1, a_0, a_{-1}, a_{-2} \ldots a_{-m}$. Váhový součet **z**-adických číslic a_i , kde váhy jsc příslušné mocniny základu **z**, nazýváme **polyáda**. Podíl dvou sousedních vah, kde větší váha je dělenec je rove **z**. U soustavy s více základy je polyáda definována podobně, ale s tím rozdílem, že podíl sousedních vah ne konstantní, ale je roven z_i . Běžně používané základy jsou 2, 8, 10 a 16. Z hlediska teorie informace je vša optimální základ e = 2,71828182..., kterému je nejbližší základ 3. V tomto případě je minimální redundanc záznamu čísla[3].

Výše uvedeným způsobem můžeme vyjádřit většinu čísel racionálních, resp. ta racionální čísla, která l vyjádřit přesně součtem konečné řady. Připustíme-li, že sumační meze budou růst do nekonečna, pa zobrazované hodnoty mohou být i iracionální, avšak to nebude případ čísla uloženého v paměti počítače, která vždy omezená. Výše uvedený vztah by bylo možno dále studovat pro případy např. záporných čísel, záporn základ, nebo jeho nejednoznačnost v případě použití záporných číslic.

4.3.2 Nepolyadické číselné soustavy

Příkladem nepolyadické soustavy je soustava římských číslic, která je však pro počítání nevhodn Použitelná nepolyadická soustava je např. soustava zbytkových tříd, označovaná též jako kód zbytkových tří Soustava je definovaná pomocí uspořádané k-tice vzájemně různých prvočíselných základů[4] $z_0, z_1 \ldots z$ Obrazem čísla A je uspořádaná k-tice celých čísel $a_0, a_{-1}, a_{-2} \ldots a_{k-1}$ pro která platí $a_i < z_i$, kde $a_i = A \bmod z_i$.

Příklad

Máme zadány základy 2, 3, 5.

Zbytkové třídy pak jsou:

0	0	0
1	1	1
	2	2
		3
		4

Např. číslo 5 je možno vyjádřit trojicí zbytků po dělení danými základy, tedy 120.

Jednoznačně lze vyjádřit pouze číslo A, pro které platí

$$A < \prod_i z_i \qquad \text{R - 4-2}$$

pro všechna i, tedy pouze číslo, které je menší než tzv. perioda neboli interval zobrazení. V našem příkladu je t 2*3*5 = 30. Je zřejmé, že velikost čísla A nezískáme polyadickým součtem. Soustava zbytkových tříd umožňuj

[2] Anglicky "radix number system".

[3] V roce 1958 byl sestrojen Nikolajem Brusentovem na Moskevské univerzitě počítač SETUN, který pracoval v trojkové soustavě Později bylo vyrobeno těchto počítačů asi 40.

rychlé operace sčítání, odčítání a násobení, protože se neuplatňují přenosy mezi jednotlivými řády. Operace dělení není jednoznačně definovaná a rovněž porovnávání velikostí čísel je obtížné. Navíc doba převodu do a zpět ze soustavy zbytkových tříd může často pohltit časovou úsporu získanou na rychlých aritmetických operacích. Proto se tento číselný systém používá v případech, kdy uvedené problematické operace nejsou zapotřebí, např. číslicová filtrace signálů v oblasti signálových procesorů.

4.3.3 Konverze mezi polyadickými soustavami o různém základu

Váha každé číslice v polyadické soustavě se lineárně zvyšuje s pozicí číslice v číselném obraze, reprezentujícím číslo. Jestliže označíme jednotlivé číslice zprava od *0* do *n-1*, pak pro libovolné číslo *x* můžeme zapsat výraz reprezentující toto číslo jako

$$x = a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \ldots + a_0 r^0$$
$$= a_0 + r.(a_1 + r.(a_2 + \ldots + r.(a_{n-2} + r.a_{n-1}))\ldots) . \qquad \text{R - 4-3}$$

Tento způsob zápisu nabízí řešení pro algoritmus konverze mezi dvěma číselnými soustavami. Jestliže totéž číslo zapíšeme v soustavě se základem *t* pak jeho číslice $b_0 \ldots b_{n-1}$ a základ budou opět odpovídat rovnici

$$x = b_0 + t.(b_1 + t.(b_2 + \ldots + t.(b_{n-2} + t.b_{n-1}))\ldots) \qquad \text{R - 4-4}$$

Při dělení rovnice základem *t* dostaneme polynom ve tvaru

$$\frac{x}{t} = Q + R \qquad \text{R - 4-5}$$

kde

$$Q = b_1 + t.(b_2 + t.(b_3 \ldots . t.b_{m-1}))\ldots)$$
$$R = b_0$$

Dalším dělením získáme koeficienty b_1, b_2, což můžeme ukázat na příkladu.

Příklad

Převeďte desítkové číslo 355 (základ 10) do dvojkové soustavy (základ 2).

V tomto případě je tedy r = 2 a t = 10. Postupné dělení bude vypadat takto

	355		
:2	177	zbytek = 1	b_0
:2	88	zbytek = 1	b_1
:2	44	zbytek = 0	b_2
:2	22	zbytek = 0	b_3
:2	11	zbytek = 0	b_4
:2	5	zbytek = 1	b_5
:2	2	zbytek = 1	b_6
:2	1	zbytek = 0	b_7
:2	0	zbytek = 1	b_8

Hledaný binární obraz čísla je $b_8\, b_7\, b_6\, b_5\, b_4\, b_3\, b_2\, b_1\, b_0 = 101100011$. Pro kontrolu můžeme provést součet decimálních váhových koeficientů s číslicemi 1, tedy

$$1 + 2 + 32 + 64 + 256 = 355 \text{ (decimálně)}$$

Poznamenejme, že

- původní číslo dělíme novým základem,
- aritmetika dělení je realizována v původním základu,
- první zbytek je číslice s nejnižší vahou.

Počtářské problémy nastanou, pokud jsou oba číselné základy méně běžné. To můžeme ukázat na jiném příkla konverze celých čísel.

Příklad

Převeďte devítkové číslo 134 (základ 9) do sedmičkové soustavy (základ 7).

V tomto případě je tedy r = 7 a t = 9. Postupné dělení bude vypadat takto

	134		
:7	17	zbytek = 0	b_0
:7	2	zbytek = 2	b_1
:7	0	zbytek = 2	b_2

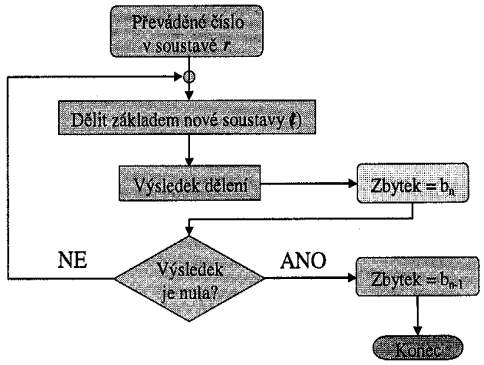
Hledaný sedmičkový obraz čísla je $b_2\, b_1\, b_0 = 220_7$.

Pro kontrolu můžeme provést součty decimálních váhových koeficientů s nenulovými číslicemi, tedy

$134_9 = (4 . 9^0) + (3 . 9^1) + (1 . 9^2) = 4 + 27 + 81 = 112$ *(decimálně)*

$220_7 = (0 . 7^0) + (2 . 7^1) + (2 . 7^2) = 0 + 14 + 98 = 112$ *(decimálně)*

Při převodu jsou čísla zpracovávána vždy v původní aritmetice, takže např. u výše uvedeného příkladu jsc desítkové ekvivalenty devítkových čísel $134_9 = 112_{10}$, $17_9 = 16_{10}$ atd. Je je zřejmé, výše uvedený způsob převoc lze použít při převodu celých čísel (integer). Postup při převodu celého čísla mezi dvěma soustavami o různé základu lze zjednodušeně zaznamenat takto



Složitější případ nastane při převodu čísla, které obsahuje nenulovou část za řádovou čárkou. Nakol proces výše uvedené konverze celého čísla sleduje dělení hodnoty zřejmým základem, konverze zlomku proces, který hledá násobky původní hodnoty, aby se zjistily číslice váhy t^m. Během konverze se generuje ce číslo délky n číslic (v případě dvojkové soustavy bitů), kde číslice s největší váhou je ta, která je výsledkem prv operace násobení. Abychom obdrželi správnou formu zápisu, musí tato číslice být první číslicí za řádovc čárkou. Ukažme si nejdříve tento převod na příkladu.

Příklad

Proveďte konverzi desetinného čísla 0,656 do binární soustavy.

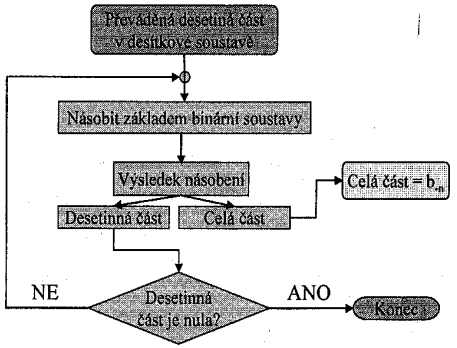
		0,656	
	.2 =	1,312	$b^{-1} = 1$
zbytek		0,312	
	.2 =	0,624	$b^{-2} = 0$
zbytek		0,624	
	.2 =	1,248	$b^{-3} = 1$

. 2 = 0,496 $b^{-1} = 0$

. . . . . atd.

Výsledek je binární číslo 0,1010 ..... .

Podobně jako u konverze celých čísel, každá číslice je vytvořena odtržením celočíselné části a použitím této hodnoty jako výsledné zlomkové číslice. Jakmile je oddělena, stává se součástí výsledku konverze a nezahrnuje se do dalších kalkulací. Převod desetinného čísla do jeho binárního obrazu je možno opět zjednodušeně znázornit takto:

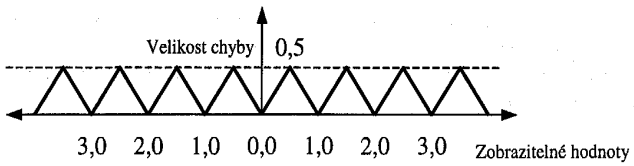


Uvědomme si, že pro každou konverzi celého čísla v soustavě o základu r bude existovat jeho přesná reprezentace v soustavě o základu t. Totéž se však zřejmě nevztahuje na zlomkové části. Pokud čtenář bude pokračovat ve výše uvedeném převodu čísla 0,656 zjistí, že nelze najít reprezentaci tohoto čísla v binární soustavě, která by měla konečný počet bitů. V tomto příkladě bude tedy vždy existovat nenulové residuum a je vidět, že i některá elementární desetinná čísla nemají odpovídající jednoduchou reprezentaci v binárním obrazu.

4.3.4 Chyby při konverzi

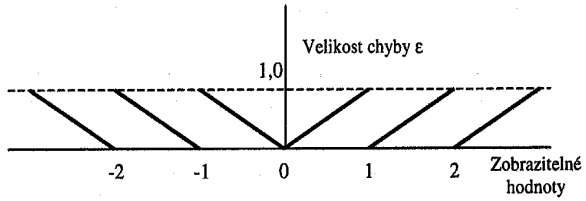
Při pořizování záznamu čísel v počítači může výsledná chyba v zobrazení čísla obecně vznikat ze čtyř příčin:

1. chyba vznikající při pořizování čísla (např. omezený rozsah přesnosti při převodu analogových veličin)
2. chyba způsobená zavedením stupnice (angl. *scaling error*) - tato chyba vzniká tak, že na rozdíl od analogové stupnice je digitální stupnice omezena konečným počtem zobrazitelných hodnot,
3. chyba zanedbáním části čísla (angl. *trunkation error*), kdy pro nedostatek míst pro zobrazení čísla se bity nejnižších vah odseknou,
4. chyba vznikající zaokrouhlením (angl. *rounding error*).



Obr. 20 Průběh chyby vzniklé zavedením stupnice

První dva případy jsou dány zařazením výpočetního systému do širšího kontextu návrhu systému a vlastní návrh počítače je nemůže ovlivnit. Pokud se pokusíme o analýzu chyby vznikající zavedením stupnice, pak zjistíme, že velikost této chyby se s ohledem na odečítanou hodnotu pohybuje po křivce ve tvaru pily (viz obr. 20). Případ, kdy chyba vzniká zanedbáním nebo zaokrouhlením při převodu desetinné části na její binární "ekvivalent" je otázkou návrhu architektury počítače. Je nutno si uvědomit, že tento převod nemusí mít konečný počet číslic a vzhledem ke konečnému počtu bitů v binárním obraze pak bude vždy existovat nejistota v platnosti posledního bitu. Chyba převodu desetinného čísla do binárního obrazu s délkou n bitů bude vždy ležet v intervalu $0 \leq \varepsilon \leq$ 2^{-n} . Tato chyba je však nesymetricky rozložena a tak se při konverzi a současném zpracování více čísel bu agregovat - dojde ke kumulaci chyby převodu.



Obr. 21 Průběh chyby vzniklé zaokrouhlením nebo odseknutím

Příklad

Je-li konvertováno 100 decimálních čísel a pak sečteno, očekávaná chyba bude

100 x (očekávaná chyba při jedné konverzi) = 100 x $(0{,}5 \cdot 2^{-n}) = 50 \cdot 2^{-n}$

Protože kumulace vzniká nesymetrickým rozložením chyby, jedna z cest jak zajistit minimalizaci této chyby docílit, aby chyba byla rovnoměrně rozložena kolem nuly pro statisticky nezávislé hodnoty. Toho se dosahu použitím konverzních algoritmů pro převod do binárního obrazu s délkou $n+1$ bitů a zaokrouhlením výsledl k nejbližší n-bitové hodnotě. Běžné zaokrouhlovací algoritmy nastaví hodnotu posledního bitu na 0, je-li $n+$ bit nulový a je-li $n+1$ bit jedničkový pak k výsledku přičtou jedničku a ponechají pouze n požadovaných bit Postup při zaokrouhlování si můžeme ukázat na příkladu.

Příklad

Převeďte desetinné číslo 0,467 do jeho binárního obrazu se třemi platnými ciframi za řádovou čárkou. Převod proveďte (a) bez zaokrouhlení (b) se zaokrouhlením.

Konverze na tři bity vede k výsledku $0{,}011_2$, což je 0,37510 . Chyba $\varepsilon = 0{,}467 - 0{,}375 = 0{,}092$.

Při konverzi na čtyři bity (nezaokrouhlené) dostaneme $0{,}0111_2$. Poslední bit je jedničkový, tedy přičteme 1 a obdržíme $0{,}100_2$. Chyba $\varepsilon = 0{,}467 - 0{,}5 = 0{,}033$.

Zaokrouhlování má dvě výhody:

- Chyba bude rozložena v intervalu $-2^{-(n+1)} \leq \varepsilon < 2^{-(n+1)}$, to znamená, že maximální chyba bu poloviční.
- Statistická distribuce chyby bude symetrická okolo nuly což znamená, že chyba se nebude kumulov lineárně, ale s druhou odmocninou.

Příklad

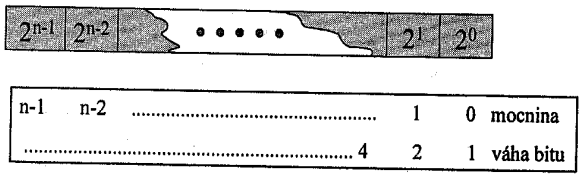
Je-li provedena konverze 100 čísel s rozložením chyby v intervalu $-2^{-(n+1)} \leq \varepsilon < 2^{-(n+1)}$, pak maximální chyba bude $\pm 25 \cdot 2^{-n}$ s nulovou střední hodnotou. Rozložení chyby je symetrické okolo nuly.

Bude-li však zaokrouhlovaná hodnota ležet vždy přesně uprostřed mezi zobrazitelnými body, doj k maximalizaci chyby, která bude mít vždy velikost 0,5. V praxi se proto často používá tzv. statistic zaokrouhlení, kdy se hodnota ležící přesně uprostřed zaokrouhluje jednou směrem nahoru a podruhé směre dolů. Toho se obvykle dosahuje zaokrouhlením na sudé nebo liché číslo.

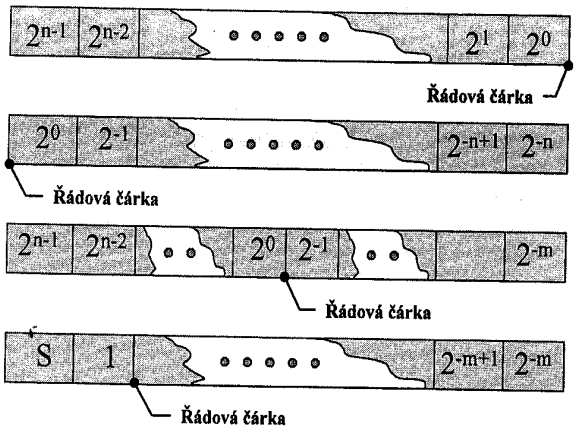
4.4 Čísla v pevné řádové čárce

Většina počítačem realizovaných výpočtů používá pro záznam číselných hodnot jednoduchých binárních zápisů. Výběr mezi různými způsoby použitého záznamu je závislý na konečném určení počítače a musí brát v úvahu zejména

- zda budou používány pouze kladné hodnoty (tedy bude možno použít záznam čísla bez znaménka), nebo kladné i záporné hodnoty (v tom případě bude záznam čísla obsahovat i znaménko),
- dopad volby použitého zobrazení na rychlost provádění aritmetických operací a na cenu aritmetické jednotky počítače. V tomto případě je nutno mít na zřeteli případná kompromisní rozhodnutí, vycházející ze znalosti nebo odhadu četnosti používaných aritmetických operací a jejich typu,
- jakým způsobem bude zobrazení čísel dále zpracováváno, čili jaké další jednotky budou s tímto zobrazením pracovat.



Obr. 22 Konstrukce váhového binárního kódu



Obr. 23 Nejčastěji používané formáty záznamu čísel v pevné řádové čárce

Nejčastěji jsou používány jednoduché váhové binární kódy, vzhledem k jednoduchosti jejich konstrukce a snadnosti realizace pomocí elektronických obvodů. Každé číslo je pak zobrazeno kombinací bitů, kde každému bitu je přiřazena jeho váha, jak ukazuje obr. 22. V tomto zobrazení mohou být mocninné koeficienty zvoleny podle potřeby a tak i řádová čárka může být umístěna kdekoliv podle zvolené architektury počítače, např.:

- zcela vpravo pro celá čísla v rozsahu $0 \le x < 2^{k-1}$,
- zcela vlevo pro zlomky v rozsahu $0 \le x < 1$,
- kdekoliv mezi bity, pro smíšený typ zobrazení,
- dále vlevo (mimo rozsah zobrazení), pokud požadovaná čísla jsou velmi malá ($x << 1$),
- dále vpravo (mimo rozsah zobrazení), pokud požadovaná čísla jsou velmi velká ($x >> 1$), v tomto případě vzdálenost mezi dvěma sousedními čísly bude větší než jeden bit.

Základní používané formáty čísel v pevné řádové čárce jsou na obr. 23. Formáty obsahují n nebo $n+$ bitů, popř. $m+2$ bity. Nejčastěji je používán formát s řádovou čárkou zcela vpravo. První tři formáty mohou b upraveny pro implicitní jedničku mimo rozsah čísla. Při operacích s čísly může dojít, vzhledem k jeji omezenému rozsahu, k tzv. *přetečení* ($2^{k-1} \rightarrow 0$) nebo *podtečení* ($0 \rightarrow 2^{k-1}$) čísla. Oba případy, jak přetečení tal podtečení musí být v systému identifikovány, protože obvykle vedou k závažným chybám.

Přetečení a podtečení

Předpokládejme, že na počítači s délkou slova 7 bitů, který používá binární kódování čísel bez znaménka, máme vynásobit číslo $0011110 = 30_{10}$ osmi. Tuto operaci lze v binární soustavě snadno provést posunutím binárního čísla o tři místa vlevo (jeden posuv reprezentuje vynásobení dvěma). Po prvním i druhém posuvu obdržíme postupně výsledky $0111100 = 60_{10}$ a $1111000 = 120_{10}$. Následující posuv však vzhledem k použité délce zobrazení binárního čísla povede ke ztrátě nejvyššího bitu, který by měl být přesunut na neexistující místo. Obdržíme tedy chybnou hodnotu $1110000 = 112_{10}$, která vznikla přetečením nejvyššího bitu.

V druhém případě můžeme předpokládat, že od hodnoty v registru $0000011 = 3_{10}$ budeme postupně odečítat binární jedničku. Po třech odečteních bude v registru uložena nula a po následujícím kroku bude registr obsahovat chybný údaj $1111111 = 127_{10}$ - způsobený podtečením.

4.4.1 Binární čísla se znaménkem

Digitální systémy musí velmi často operovat jak s kladnými tak i se zápornými čísly. Rozšíříme-li zobraze n-bitového čísla o další bit, rozsah zobrazitelných čísel se dvakrát zvětší z $0 \le x \le 2^n - 1$ na interval $0 \le x \le 2^{n+1}$- Když místo rozšíření intervalu kladných hodnot rozšíříme odpovídající interval do záporných čísel, mů přidaný bit představovat **znaménkový bit**. Toto je základní způsob vytvoření záporné hodnoty čísla přidání znaménkového bitu na posici s nejvyšší vahou; zobrazení čísla bude obsahovat tzv. znaménkový bit a příslušnc hodnotu. Např. zobrazení čísla 7 bude odpovídat 00...0111 a číslo -7 bude mít změněný nejvyšší bit, tec 100...0111. Přestože zobrazení používá nejvyšší bit pro uložení znaménka, tento bit nemá přiřazenu žádnou váh nebo hodnotu stejně jako ostatní bity v zobrazení čísla, je pouze nenumerickým indikátorem znaménka. Ačkc je tento způsob zobrazení celých čísel se znaménkem velmi často používán, přináší dva problémy

- existují dvě nuly (kladná a záporná)
- algoritmy sčítání a odčítání vyžadují řadu časově náročných rozhodnutí.

Druhý způsob využívá přičtení positivní konstanty a nazývá se **zobrazení s posunem**. Pro číslo s n bity je 2^n kombinací negativních a 2^{n-1} kombinací positivních např. záporná čísla v rozsahu -1 až -2^{n-1} a kladná v rozsahu až 2^{n-1}. Tento typ záznamu eliminuje předchozí nevýhody a pro svoji jednoduchost se často používá jak výsledný kód převáděné hodnoty v analogově-číslicových převodnících.

Příklad

Máme zobrazit uložení čísel okolo hodnoty 50_{10}, která je použitou hodnotou posunu. Zobrazíme tři hodnoty na každou stranu v dekadické soustavě.

...		...
53	⇒	3
52	⇒	2
51	⇒	1
50	⇒	**0**
49	⇒	-1
48	⇒	-2
47	⇒	-3
...		...

4.4.2 Dvojkový doplněk

Mezi nejznámější metodu záznamu negativních binárních čísel patří dvojkový doplněk (komplement), zřejmě pro svoje výhodné vlastnosti při aritmetických operacích. Nezáporná čísla se v tomto kódování zobrazí beze změny, kdežto místo záporných čísel se pracuje s jejich obrazy. Pro každé číslo A platí, že jeho doplněk $\{A\}$ je možno vytvořit podle vztahu

$$\{A\} = A + M = -|A| + M \qquad \text{R - 4-6}$$

kde $\{A\}$ je doplněk čísla do modulu M. Modulem je při použití polyadické soustavy celistvá mocnina základu, z^n. Vytvoření doplňku např. v desítkové soustavě znamená dočítání např. do čísla 100, 1000 atd. podle rozsahu čísla. Při záznamu s použitím doplňku ve dvojkové soustavě obvykle používáme negaci zobrazovaného čísla ke kterému přičteme jedničku. V takové soustavě je nula kódována jako 000...000 a nejkladnější hodnota jako 0111...111. Záporná jednička je ve tvaru 111...111.

Příklad

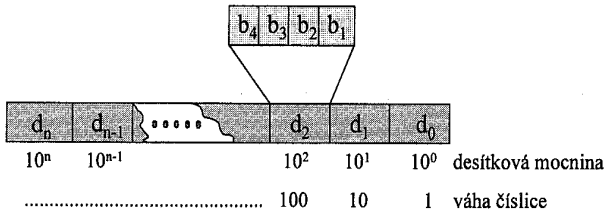
Čísla 37_{10} a -8_{10} mají být sečtena. Nejprve vytvoříme dvojkové obrazy obou čísel, kde záporné číslo je zobrazeno ve dvojkovém doplňku. Následovně provedeme součet.

37_{10}	$= 0100101_2$	Součet 0100101	37_{10}
8_{10}	$= 0001000_2$	1111000	-8_{10}
-8_{10}	$= 1110111_{2inv}$	Vzniklý přenos se zanedbává 10011101	29_{10}
-8_{10}	$= 1111000_{2dop}$		

Někdy se používá čistá negace, tedy doplněk do čísla z^n-1. V tom případě se však už jedná o kód inverzní. Tento kód se vytváří v desítkové soustavě záměnou čísla **9** za číslo **0**, **8** za **1**, **7** za **2** atd. Ve dvojkové soustavě prostým negováním bitů.

4.4.3 Čísla v kódu BCD

Desítkové zobrazení čísel je pro člověka nejpřirozenější. Proto také první počítače byly ovládány snahou reprezentovat čísla v desítkovém záznamu. Při takovém zobrazení odpadá nutnost konverze a tím i její chyby a nepřesnosti. V počítačové aritmetice se používá přímý binární kód BCD (**B**inary **C**oded **D**ecimal), kde jsou dekadické číslice ukládány do čtveřic bitů, které representují vždy právě jednu dekadickou číslici, tak jak je ukázáno na obr. 24.



Obr. 24 Způsob uložení čísla v kódu BCD bez znaménka

Pro reprezentaci čísla v binárním záznamu je potřeba

$$n > \log_2 x$$

bitů. Pro reprezentaci téhož čísla v BCD je třeba

$$m > \log_{10} x$$

bitů. Jsou-li použity čtyři bity pro záznam jedné dekadické číslice, pak celkový počet bitů pro záznam této čísl bude zapotřebí $4.\log_{10} x$ bitů. Celková redundance záznamu bude

$$\frac{4.\log_{10} x}{\log_2 x} = 4.\log_{10} 2 = 1{,}204..$$

To znamená, že záznam v kódu BCD bude vykazovat asi 20% redundanci což platí v případě porovná reprezentací velkých čísel. Pro záznam jednotlivé číslice bude redundance udána počten nadbytečnýc kombinací ve čtveřici, t.j.

$$\frac{6}{16} = 37{,}5\%$$

Pro zobrazení kladných i záporných čísel v kódu BCD můžeme použít, podobně jako v binárních obrazec bud' zvláštní část čísla pro uložení znaménka nebo desítkový doplněk. Kódování znaménka, které se rovně ukládá do čtveřice bitů, může být určeno při návrhu libovolným způsobem. Používají se neobsazené kódov obrazy, např. 1100 pro kladné a 1101 pro záporné znaménko.

Desítkový komplement je BCD verzí dvojkového komplementu a pro čísla složená ze čtveřic bitů j reprezentovatelná hodnota $-5.10^{n-1} \le x < 5.10^{n-1}$.

Příklad

Máme odečíst dvě záporná čísla -39 a -42 s použitím desítkových doplňků. Nejprve připravíme desítkové doplňky

(-39) = 100 - 39 = 61

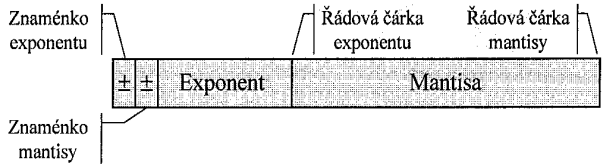
(-42) = 100 - 42 = 58

Rozdíl (-39) - (-42) je pak ekvivalentní

61 - 58 = **3**

4.5 Čísla v plovoucí řádové čárce

Při zpracování některých úloh může dojít k situaci, kdy veličiny dosáhnout hodnot tak velkých, že j nemožné je zobrazit do vyhrazeného místa - dojde k tzv. přeplnění. Bude-li mít programátor plnou kontrolu na procesem, může zvolit vhodné měřítko, kterým veličinu vynásobí a tak umožní její uložení do vyhrazeném místa. Obvykle se měřítko volí takové, aby bylo zobrazeno v užité číselné soustavě jedinou jedničkou a řado nul, což zjednoduší úpravu čísla - místo násobení se provádějí posuvy. Bude výhodné při uložení označ společně s uloženým číslem i jeho měřítko celým číslem, udávajícím potřebný počet posunutí.



Obr. 25 Přímý záznam čísla v plovoucí řádové čárce

Příklad

Číslo 965 137 má být uloženo do rozsahu tří cifer. Zvolíme měřítko 10^{-3}, tedy posuneme číslo o tři řády **doprava** (označení řádu je **-3**) . V novém měřítku budeme tedy zobrazovat (nepřesně) číslo 965 v měřítku 10^{-3}. Uložíme-li i měřítko jako hodnotu udávající potřebný počet posuvů **doleva**, abychom získali skutečnou hodnotu, pak záznam výše uvedeného čísla v paměti může vypadat 965;3.

Toto je základní myšlenka uložení čísla v plovoucí řádové čárce. Obecně se číslo X v plovoucí řádové čárce vyjadřuje dvojicí M ;e, kde M se nazývá mantisa a e je exponent a platí $X = M \cdot z^e$.

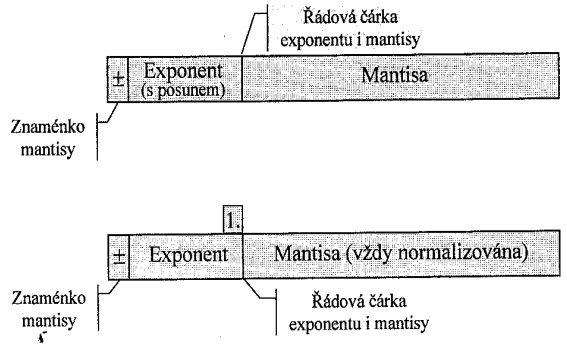
Obě čísla, M i e mohou nabývat jak kladných tak i záporných hodnot uvnitř vyhrazených intervalů, daných reálným provedením stroje. Číslo e je vždy celé, zatímco M může mít řádovou čárku umístěnu dle dohody, nejčastěji však na levém okraji. Pro správné zobrazení čísla musí být předem explicitně známa velikost mantisy a exponent. Implicitně se obvykle chápe základ exponentu a poloha řádové čárky. Protože platí

$$X = M \cdot z^e = (M, z^{-\alpha}) \cdot Z^{(e+\alpha)} = M' \cdot z^{e'}$$

kde α je celé číslo, panuje v tomto vyjádření nejednoznačnost. Pro její odstranění je udržována úmluva, že M leží v intervalu $z^{-1} \leq |M| < 1$. Toto vyjádření se nazývá normalizovaný tvar a jeho význačným rysem je, že první číslice za řádovou čárkou je nenulová. Normalizovat je možno tedy pouze zlomkové mantisy a to ve dvou krocích:

1. Posunutí mantisy vlevo tak, aby její první číslice za řádovou čárkou byla nenulová.
2. Snížení exponentu o hodnotu odpovídající počtu míst, o které byla mantisa posunuta v prvním kroku.

Normalizovat je možno nejenom čísla binárně zobrazená, ale i čísla BCD. vždy je však nutno dbát implicitně daného základu exponentu. Při normalizaci může dojít k situaci, kdy exponent již nemůžeme snížit natolik, abychom mohli posunout první platnou číslici až k řádové čárce. Tyto situace se pak řeší různými úmluvami, které se stanovují s ohledem na návrh aritmetiky a vlastní aritmetické jednotky, nebo jsou řešeny v doporučeních a normách (viz kap. 0).



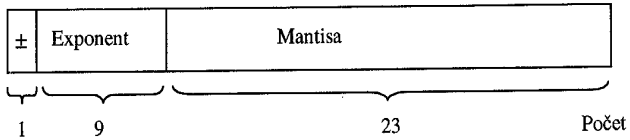
Obr. 26 Nejčastěji používané záznamy čísla v plovoucí řádové čárce

Z používaných zobrazení čísel v plovoucí řádové čárce patří přímé zobrazení uvedené na obr. 25 mezi nejsrozumitelnější vyjádření požadované hodnoty. Obě části, jak mantisa tak i exponent jsou opatřeny znaménky, uloženými zcela vlevo. Řádové čárky exponentu i mantisy jsou umístěny vpravo, což znamená, že jak exponent tak i mantisa jsou celá čísla. V tomto zobrazení, jak je zřejmé, bude docházet k nejmenší chybě převodu v případě celých čísel, která mohou být do rozsahu mantisy uložena.

Dva nejčastěji používané záznamy čísla v plovoucí řádové čárce jsou uvedeny na obr. 26. V prvém případě je použito zobrazení exponentu s posunem a zlomkového tvaru mantisy. Nastavením vhodné velikosti konstanty posunu můžeme posouvat i krajní meze intervalu zobrazení čísel tak, aby maximálně vyhovovaly očekávané aplikaci. Zlomková mantisa má umístěnu řádovou čárku zcela vlevo a tak první platná binární číslice má váhu 2^{-1}. Druhý obrázek ukazuje využití normalizovaných binárních hodnot pro zobrazení čísel v plovoucí řádové čárce. Předpokládá totiž, že první platná číslice za řádovou čárkou musí být jednička a proto provádí ještě jeden posun s tím, že tato jednička je vnímána implicitně. Takový způsob zobrazení čísla v plovoucí řádové čárce nazýváme zobrazení se skrytým bitem. V případě, že je použito zobrazení se skrytým bitem, musí být čísla vždy normalizována.

Příklad

Zobrazte číslo -0,375 jako číslo v plovoucí řádové čárce s exponentem o binárním základu v níže uvedeném tvaru. Řádová čárka exponentu i mantisy je umístěna vlevo od mantisy (tzn. mezi exponentem a mantisou). Exponent je uložen s posunem[4] o 64, mantisa je binárním ekvivalentem čísla. Pro uložení znaménka předpokládejte jedničkový bit při záporném znaménku. Následovně číslo normalizujte v běžném formátu a ve formátu se skrytým bitem.



Řešení: Číslo 0,375 je možno zapsat jako $2^{-2}+2^{-3}$, tedy v binárním tvaru budou jedničkové pouze tyto bity. Nenormalizovaný tvar bude

```
1  001000000  01100...
```

Po normalizaci obdržíme

```
1  000111111  11000...
```

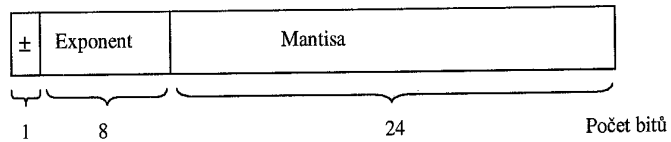
a po normalizaci se skrytým bitem

```
1  000111110  10000...
```

Normalizovat je samozřejmě možno i záznamy čísel, jejichž exponenty nemají binární základ[5] nebo jejich mantisy jsou zobrazeny jinak než binárně. V tom případě je nutno si uvědomit, jaká binární kombinac reprezentuje příslušnou číslici a jak se bude posouvat exponent.

Příklad

Zobrazte číslo -0,035 jako číslo v plovoucí řádové čárce s exponentem o dekadickém základu v následujícím tvaru: Řádová čárka exponentu i mantisy je umístěna vlevo od mantisy (tzn. mezi exponentem a mantisou). Exponent je uložen s posunem o 64, mantisa je v kódu BCD. Pro uložení znaménka předpokládejte jedničkový bit při záporném znaménku. Následovně číslo normalizujte.



Řešení:

Číslo 0,035 je možno zapsat jako tři číslice BCD. Nenormalizovaný tvar bude

```
1  01000000   0000 0011 0101 0000...
```

Po normalizaci obdržíme

```
1  00111111   0011 0101 0000...
```

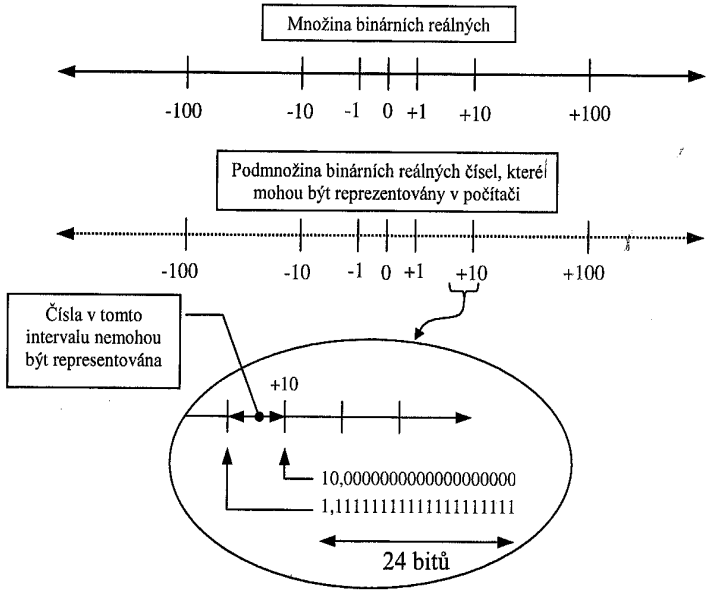
4.5.1 Přesnost záznamu reálných čísel

Reálná čísla představují kontinuum, které je spojité od -∞ do +∞. Protože velikost a počet registrů res paměťových míst v počítači je vždy omezena, může být prakticky zobrazena pouze podmnožina reálných čís Jak je ukázáno v obr. 27, podmnožina reprezentovatelných reálných čísel je pouze aproximací skutečné množir reálných čísel, kterou podporuje konkrétní architektura počítače. Jako příklad zobrazení reálného čís v dekadické a binární soustavě může sloužit Tab. 1. Minimální rozdíl mezi dvěma sousedními reprezentacer

[4] Často se používá též termín *excess 64*.

[5] To samozřejmě nelze u normalizace se skrytým bitem.

o jedničku menší než je váha nejméně významného bitu mantisy.



Obr. 27 Množina binárních reálných čísel - velikost mantisy určuje interval mezi dvěma možnými reprezentacemi

Tab. 1 Různé způsoby záznamu reálného čísla v dekadické a binární soustavě

Způsob záznamu čísla	Hodnota		
Dekadický, v pevné řádové čárce	178,125		
Dekadický, vědecký	1,78125 E_{10} 2		
Binární vědecký	1,0110010001 E_2 111		
Binární vědecký (exponent s posunem)	1,0110010001 E_2 10000110		
Formátovaný zápis binárního vědeckého záznamu (exponent s posunem)	Znaménko	Exponent s posunem	Normalizovaná mantisa
	0	10000110	01100100010000000000000

Otázka volby délky mantisy a poměru délky mantisy a exponentu hraje velmi důležitou roli při návrhu rchitektury počítače. Ve spojitosti s charakteristikami zobrazení čísel mohou totiž nevhodně volené algoritmy ejich zpracování v případě čísel v plovoucí řádové čárce vést k nestabilitě výpočtu, či neočekávanému chování troje. K tomu může dojít i při tak elementárním výpočtu jako při hledání kořenů kvadratické rovnice podle ztahu

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

když kořeny budou velmi blízké nebo naopak, když se budou velmi lišit. Rovněž otázka přetečení a podteče bude hrát důležitou roli ve chvíli, když koeficienty *a, b, c* budou velmi velké či naopak velmi malé. Pro unifika způsobů práce s čísly v plovoucí řádové čárce bylo připraveno doporučení IEEE-754 (později 854), kte zobrazení těchto čísel v architektuře počítače standardizuje.

4.5.2 Doporučení IEEE-754 (854)

Standard IEEE-754[6] z roku 1985 definuje reprezentaci čísel v plovoucí řádové čárce v několika variantách i když není závazný, dodržuje jej většina současných výrobců. Rozšířený standard IEEE-854, který byl vyd v roce 1987 je používán méně. Základem standardu je stanovení parametrů a omezujících podmínek p zobrazované entity. Každé číslo podle tohoto doporučení je udáno ve tvaru

$$(-1)^s \cdot b^E \cdot (d_0\, d_1\, d_2 \ldots d_{p-1})$$

kde parametr

- s je algebraické znaménko,
- b je základ,
- p je počet číslic v mantise,
- E je libovolné číslo mezi E_{min} a E_{max},
- d_i je číslice se základem b.

Omezující podmínky stanoví hodnotu základu b podle normy IEEE-754 $b = 2$, norma IEEE-854 rozšíř definici základu o $b = 10$. Hodnota E musí být stanovena tak, aby byla splněna podmínka

$$\frac{E_{max} - E_{min}}{p} > 5$$

a doporučuje se hodnota tohoto podílu větší než 10. Poslední omezující podmínka stanoví minimální rozm čísla předpisem

$$b^{p-1} > 10^5.$$

Kromě definic číselných hodnot norma pamatuje i na zvláštní případy a definuje

- dvě nekonečna ($+\infty$ a $-\infty$), které vznikají podílem s nulovým dělitelem. S tím souvisí i případná definic aritmetiky v nekonečných hodnotách např. $\arctan(\infty) = \pi/2$,
- nečíselnou hodnotu označovanou **NaN** (Not a Number), která se používá zejména jak výsledek nedefinovaných operací, např. $\sqrt{-1}$ nebo pokud argumenty hledané funkce jsou mim definovaný obor. Standard definuje dva typy NaN - **tiché NaN**, které nezpůsobí nastavení příznaku procesoru provádějícího aritmetickou operaci s tímto NaN a **signální NaN**, které naopak tent příznak nastaví.

Zaokrouhlování ve standardu IEEE-754 je definováno tak, že se vždy zaokrouhluje na číslo, jehož nejniž číslice je v binární soustavě sudá a procedura zaokrouhlení je definována pro čtyři případy

- zaokrouhlení k nejbližšímu číslu,
- zaokrouhlení k nule,
- zaokrouhlení k $+\infty$,
- zaokrouhlení k $-\infty$.

[6] IEEE - Institution of Electrical and Electronics Engineers je americká normotvorná organizace, vydávající standardy formou doporučení.

Norma definuje rovněž čtyři přesnosti vyjádření - jednoduchou přesnost 24 bitů, jednoduchou rozšířenou přesnost ≥ 32 bitů, dvojitou přesnost 53 bitů a dvojitou rozšířenou přesnost ≥ 64 bitů. Z těchto přesností je doporučeno implementovat alespoň dvě. Hodnoty jednotlivých obrazů reprezentací a jejich odpovídající položky jsou pro 32-bitové zobrazení uvedeny v Tab. 2.

Exponent	Mantisa	Reprezentovaná hodnota
$E = 255$	$M = 0$	$\pm\infty$
$E = 255$	$M \neq 0$	NaN
$0 < E < 255$	normalizovaná	$(-1)^s \cdot 2^{E-127} \cdot 1,M$[7]
$E = 0$	nenormalizovaná	$(-1)^s \cdot 2^{-126} \cdot 0,M$
$E = 0$	$M = 0$	± 0

Tab. 2 Hodnoty reprezentace ve standardu IEEE-754 pro přesnost 32 bitů

Významné položky v této tabulce jsou pro maximální a nulovou hodnotu exponentu. Pokud exponent je maximální a mantisa je nenulová, pak reprezentovaná hodnota je NaN ("Not a number"). V případě nulové mantisy určuje znaménkový bit zda reprezentovaná hodnota je kladné či záporné nekonečno. Nula je zobrazena jako nulový exponent a nulová mantisa, což vzhledem k doporučení k normalizaci se skrytým bitem je vybočení ze standardního předpisu, stejně tak jako nenormalizované zobrazení. Pro zobrazení používající 64 bitovou přesnost jsou uvedeny odpovídající hodnoty v Tab. 3.

Exponent	Mantisa	Reprezentovaná hodnota
$E = 2047$	$M = 0$	$\pm\infty$
$E = 2047$	$M \neq 0$	NaN
$0 < E < 2047$	normalizovaná	$(-1)^s \cdot 2^{E-1023} \cdot 1,M$
$E = 0$	nenormalizovaná	$(-1)^s \cdot 2^{-1022} \cdot 0,M$
$E = 0$	$M = 0$	± 0

Tab. 3 Hodnoty reprezentace ve standardu IEEE-754 pro přesnost 64 bitů

4.5.3 Reprezentace reálných čísel v procesorech Intel

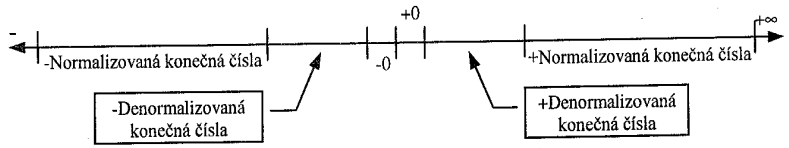
Procesory firmy Intel byly mezi prvními které používaly pro reprezentaci reálných čísel standard IEEE-754. Typicky jsou reprezentovány v binární formě se třemi význačnými částmi - znaménkem, exponentem a mantisou. Znaménko je binární hodnota, které označuje positivní (0) nebo negativní (1) entitu. Exponent používá záznam s posunem a mantisa je ve většině případů normalizována se skrytým bitem[8]. To znamená, že mantisa se vždy skládá z celého jednobitového čísla a zlomkové části ve tvaru

`1.dddd...dd`

Hodnota o kterou je posouván exponent je dána přesností ve které bude reprezentace pracovat. Jednotka procesoru, pracující s plovoucí řádovou čárkou, rozeznává následující reprezentace reálných čísel a nečíselných kódů:

- nuly se znaménkem,
- nenormalizovaná konečná čísla,
- normalizovaná konečná čísla,
- nekonečna se znaménkem,
- kódy NaN.

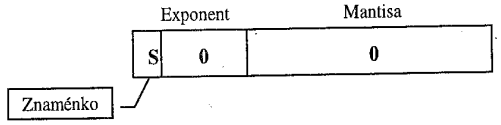
Rozložení reprezentovaných reálných čísel podél číselné osy ukazuje obr. 28. Hodnoty NaN nejsou zobrazeny, protože nejsou součástí číselné osy.



Obr. 28 Rozložení reprezentace reálných čísel podle číselné osy

4.5.3.1 Nuly se znaménkem

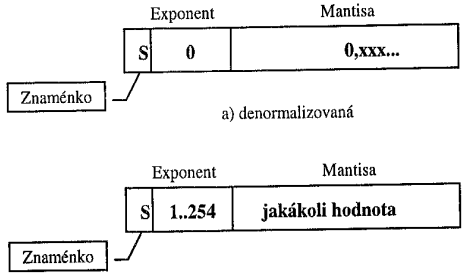
Nula může být reprezentována jako +0 nebo -0, v závislosti na znaménkovém bitu. Obě reprezentace mají stejnou hodnotu a znaménko závisí na předchozí prováděné operaci a použitém zaokrouhlovacím algoritmu. Znaménko nuly může odrážet směr ze kterého došlo k podtečení nebo znaménko ∞, které bylo účastno přechozí operace dělení. Tvar zobrazení obou nul je na obr. 29.



Obr. 29 Zobrazení reálných nul

4.5.3.2 Normalizované a denormalizované konečné hodnoty

Reprezentace konečných nenulových hodnot je rozdělena do dvou tříd - normalizované hodnoty nenormalizované hodnoty. Třída normalizovaných konečných hodnot obsahuje všechny konečné hodnot v intervalu $(0, \pm\infty)$, které mohou být uloženy v normalizovaném formátu čísla v plovoucí řádové čárce. Pr jednoduchý formát Intel to představuje čísla s posunutým exponentem v intervalu $\langle 1,254 \rangle$[9].

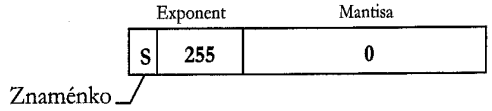


Obr. 30 Normalizované a denormalizované číselné hodnoty

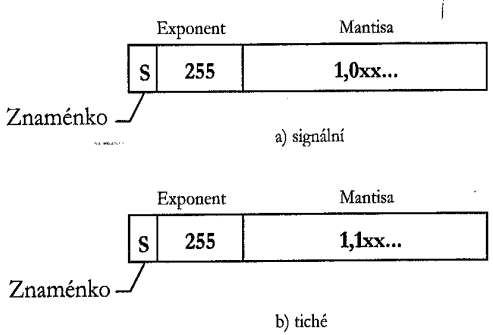
[7] Výraz 1,M znamená "jedna celá" - "řádová čárka" - "zlomková část určená mantisou"

[8] V terminologii Intel se nazývá J-bit.

[9] To znamená, že skutečná hodnota exponentu se pohybuje v intervalu $\langle -126_{10}, +127_{10} \rangle$.



Obr. 31 Reprezentace nekonečna se znaménkem



Obr. 32 Reprezentace NaN, tichého a signálního

V okamžiku, kdy se hodnota reálného čísla dostane velmi blízko k nule, nemůže být nadále použito normalizované číslo pro reprezentaci odpovídající reálné hodnoty. Důvod je prostý - rozsah exponentu již nestačí kompenzovat potřebné posuny mantisy. Jakmile dosáhne posunutý exponent hodnoty **0**, pak menší číslo může být reprezentováno pouze tak, že hodnota úvodního celočíselného bitu se změní z implicitní **1** na **0**. Takto zobrazená čísla se nazývají denormalizovaná a umožňují reprezentovat velmi malé hodnoty, bohužel za cenu ztráty přesnosti[10]. Zobrazení normalizovaných a denormalizovaných čísel v procesorech Intel je na obr. 30.

4.5.3.3 Nekonečna se znaménkem

Standard IEEE-754 definuje dvě nekonečna - kladné a záporné, které v počítači reprezentují maximální dosažitelné kladné nebo záporné hodnoty v zobrazení s plovoucí řádovou čárkou. Nekonečno v tomto smyslu je vždy reprezentováno nulovou celočíselnou i zlomkovou hodnotou mantisy a maximální hodnotou exponentu (např. 255_{10} v jednoduché přesnosti) a chápáno tak, že $+\infty$ je větší než největší konečné číslo a podobně $-\infty$ je menší než jakékoliv konečné číslo[11]. Typický formát reprezentace nekonečna se znaménkem je na obr. 31.

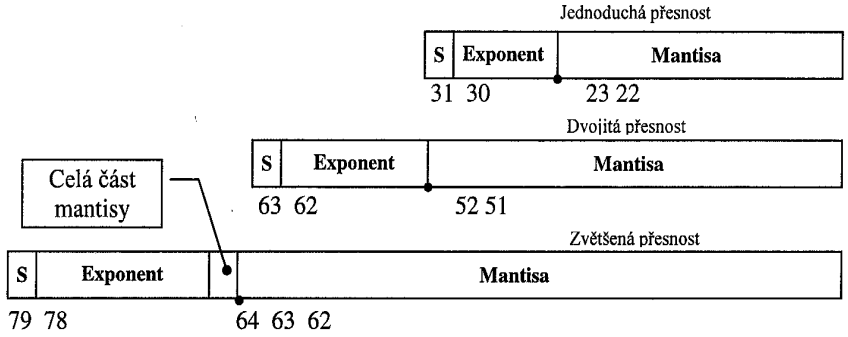
4.5.3.4 Nečíselné hodnoty

Nečíselné hodnoty označované jako NaN jsou kódy, které nejsou součástí číselné osy reálných čísel. Jsou reprezentována jako maximální hodnota posunutého exponentu a nenulová mantisa. Jak již bylo uvedeno, standard IEEE-754 předpokládá existenci tichého a signálního NaN, které jsou v implementaci Intel odlišeny hodnotou nejvyššího bitu zlomkové části mantisy. Tiché NaN má nenulovou hodnotu tohoto bitu, zatímco hodnota nejvyššího bitu zlomkové části mantisy pro signální NaN je nulová. Intel označuje ještě speciální hodnotu tichého NaN jako "nedefinováno".

[10] Denormalizovaná čísla se někdy nazývají "tiny" a jednotka pro práci v plovoucí řádové čárce obvykle způsobí nastavení příznaku "speciálního podtečení", když výsledkem nebo součástí operace je denormalizované číslo. Přesnost se rychle ztrácí se stoupajícím počtem úvodních nul mantisy.

[11] Procesory Intel umožňují operaci porovnání znamének operandů, které jsou $\pm\infty$.

Architektura procesoru Intel v plovoucí řádové čárce rozlišuje sedm typů dat rozdělených do tří skupin *reálná čísla, celá čísla* a *pakovaná celá čísla* ve formátu BCD. Základní formáty pro skupinu čísel v plovoucí řádo čárce v procesorech Intel jsou uvedeny na obr. 33. Jsou rovněž podporovány nenormalizované hodnoty, tak j požaduje standard IEEE-854. Poznamenejme, že uvedené formáty mohou být uloženy do paměti počíta s výjimkou 80-ti bitového zvětšeného reálného formátu, který je interní reprezentací procesoru a do paměti ukládám n pouze výjimečně. Pro zvýšení přesnosti jsou všechna čísla po zavedení do registrů proceso převedena do této interní reprezentace a po provedení operace opět uložena do paměti v původním formátu.



Obr. 33 Formáty zobrazení reálných čísel v procesorech Intel

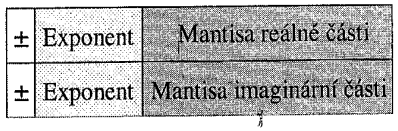
Čísla v jednoduché přesnosti jsou vhodná pro aplikace, kde je nutno hospodařit s pamětí; jsou méně přesn a existuje větší pravděpodobnost přetečení. Rovněž při odlaďování programů je vhodné použít jednoducho přesnost, protože případný problém způsobený zaokrouhlováním se objeví v tomto formátu rychleji. Zvětšen reálný formát je, jak již bylo uvedeno, interní reprezentací procesoru v plovoucí řádové čárce a slouží pr ukládání mezivýsledků a konstant. Jeho úkolem je ošetřit konečný výsledek od vlivu zaokrouhlování a podtečer nebo přetečení při mezioperacích. Procesor plovoucí řádové čárky Intel umožňuje, pro zachování vysok přesnosti, uložit tento formát i do paměti. Speciální hodnota "nedefinováno" je navrácena procesorem jak výsledek neplatné operace. Rozsah, přesnost a normalizovaný rozsah zobrazovaných reálných čísel je uveden Tab. 4.

Zobrazená entita		Sign	Posunutý exponent	Mantisa	
				Celá část	Zlomková část
Kladná čísla	$+\infty$	0	11 ... 11	1	00 ... 00
	+ normalizovaná	0	11 ... 10 až 00 ... 01	1	11 ... 11 až 00 ... 00
	+ denormalizovaná	0	00 ... 00	0	11 ... 11 až 00 ... 01
	+ 0	0	00 ... 00	0	00 ... 00
Záporná čísla	- 0	1	00 ... 00	0	00 ... 00
	- denormalizovaná	1	00 ... 00	0	00 ... 01 až 11 ... 11
	- normalizovaná	1	00 ... 01 až 11 ...10	1	00 ... 00 až 11 ... 11
	$-\infty$	1	11 ... 11	1	00 ... 00
NaN	signální NaN	X	11 ... 11	1	0X ... X
	tiché NaN	X	11 ... 11	1	1X ... X
	"nedefinováno"	1	11 ... 11	1	10 ... 00

Tab. 4 Zobrazení reálných čísel a NaN v procesorech Intel

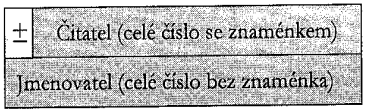
4.6 Speciální typy záznamu čísel

Během rozvoje oblastí s uplatněním počítačů se objevily i požadavky na některá speciální zobrazení čísel tak, aby lépe vyhovovaly připravované aplikaci. Ke zvláštním typům patří např. komplexní číslo, které obsahuje dvě části, obě reprezentovány jako čísla v plovoucí řádové čárce, tak jak je ukázáno na obr. 34. Na takový typ je možno nahlížet i jako na objekt realizovaný na nejnižší úrovni stroje. V této úrovni jsou rovněž implementovány algoritmy operací s komplexními čísly, které výrazně zvýší rychlost zpracování tohoto typu.



Obr. 34 Reprezentace komplexního číslo

Jiným případem je spíše již historický požadavek na aritmetiku pracující s racionálními čísly, uloženými v paměti počítače jako dvojice celých čísel, čitatel a jmenovatel zlomku. Objekt vytvořený ze dvou standardních obrazů je na obr. 35. Experimenty s tímto způsobem uložení čísel, prováděné na konci 70-tých let ukázaly, že v rychlosti zpracování nemůže tento obraz čísla dosáhnout možností daných zápisem čísla v plovoucí řádové čárce. Rovněž mnoho numerických hodnot, kterých může nabývat číslo během zpracování, není možno zapsat do srovnatelného prostoru jako odpovídající reprezentace v plovoucí řádové čárce. Navíc, konverze čísla do tvaru zlomku vykazovala mnohem větší chyby než odpovídající konverze do čísla v plovoucí řádové čárce. Nicméně, i když tzv. racionálním záznamem čísla nelze rovnoměrně nahradit záznam v plovoucí řádové čárce, zůstává možnou eventualitou pro speciální případy, kdy může být realizován jako speciální objekt v oblasti mikroprogramů.



Obr. 35 Zlomková reprezentace čísla

4.7 Data s automatickou identifikací

Požadavek na včasnou detekci chyb a jejich jednoduchou opravu si vyžádal i doplnění typů dat o položky sloužící k jejich automatické identifikaci. V této oblasti se uplatňuje rozšíření datové položky o speciální atribut, kterým je doplněna každá datová položka, a který zabraňuje použití této položky dat nevhodným způsobem.

Historickým příkladem použití autoidentifikace je implementace interpreteru WATFOR na stroji IBM 7090 na University of Waterloo. Na počátku WATFOR inicializoval celou paměť dat daty s paritní chybou[12]. Zatímco zápis položky dat do paměti změnil tuto položku na platnou hodnotu (bez paritní chyby), čtení z programem neobsloužené buňky dat vyvolalo hardwarovou signalizaci chyby, která byla detekována programem a dále zpracována. S příchodem strojů s vestavěnou kontrolou parity se tento způsob autoidentifikace dat ztratil.

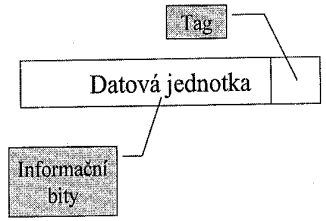
4.7.1 Tagované paměti

V klasickém stroji, který splňuje požadavky von Neumannovy architektury, je pojem **atributu datové položky** velmi subjektivní. Specifické hodnoty (syntaxe) mohou být interpretovány různým způsobem (sémantika). Tatáž položka může mít jiný význam jako celé číslo nebo jako kódovaný znak. Přidáme-li ke každé datové položce speciální skupinu bitů, která bude blíže specifikovat uložený typ, každá datová položka tak obdrží informaci důležitou pro její interpretaci - stane se automaticky identifikovatelnou. První stroj, kde tato myšlenka byla použita, byl Burroughs B5000. Používal tříbitový tag[13] umístěný podobně jako tag na obr. 36. Výhodou tagovaných pamětí je

- jednoduchá korekce chyb
- automatická konverze datových typů vykonávaná hardwarem místo konverzních funkcí generovaných překladačem, která má dále dopad na
- jednodušší překlad a rychlejší běh programu,
- používání obecných typů instrukcí - např. jedna instrukce sčítání **ADD** místo různých instrukcí **ADD** pro různé datov typy.

Naopak mezi jejich nevýhody patří

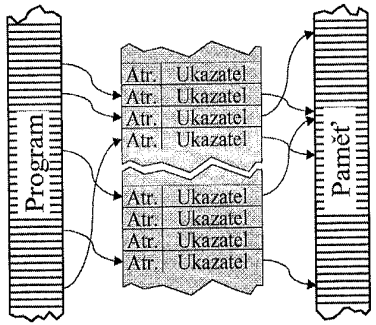
- zvýšení ceny paměti o přidané bity ke každé adrese paměti, registru nebo položce zásobníku, spolu s cenou širších datových cest
- zvýšení ceny obvodů pro zpracování informace z tagů,
- nutnost vytvoření korektního prostředí pro práci s daty, které může omezit některé programátorské triky.



Obr. 36 Buňka tagované paměti

4.7.2 Deskriptory dat

Deskriptory dat obsahují informace, které souvisí s typem uložených dat, jejich přesností, rozměrů pol omezení ukládaných hodnot, stejně jako odkazují na umístění dat v paměti. Operandy instrukcí, které používá deskriptory dat, se odkazují na blok deskriptorů, který ukazuje na příslušnou buňku dat a obsahuje atribu obsahu této buňky - viz obr. 37. Všechny operandy jsou v tomto případě nepřímo adresované nebo zahrnu alespoň jednu mezioperaci při přístupu. Mezi výhody deskriptorů dat patří, stejně jako u tagovaných pamě zlepšená korekce chyb a automatická identifikace typu. Nevýhodou je navýšení ceny paměti o prostor ve které budou deskriptory uloženy, a prodloužení času zpracování instrukce, způsobený nepřímým adresováním kontrolou atributů. V některých případech se však kombinuje používání tagů i deskriptorů v paměťových obvodech jednoho stroje.



Obr. 37 Blok deskriptorů s paměťovými odkazy

4.7.3 Objekty

Koncepce objektu jako nositele datového obsahu je stále důležitějším pojmem jak v oblasti software ta hardware. Objekt v tomto případě je abstrakce nositele dat - paměťové buňky, která implicitně očekává, že

- objekt může být vytvořen nebo zrušen jenom jako celá entita,
- k objektu lze přistupovat jenom jako k celku,
- vnitřní struktura objektu je neviditelná
- objekt je transformovatelný pomocí instrukcí
- objekt obsahuje autoidentifikační informaci

[12] Parita je dodatečná informace uložená v paměti a sloužící pro identifikaci chyb čtení (viz dále).

[13] Český ekvivalent pro "tag" by byl "štítek" nebo "značka, nálepka". V dalším výkladu se přidržíme původního anglického termínu.

Objektově orientované architektury jsou velmi úzce spojeny s adresovacími možnostmi stroje a možnostmi ochrany informace ve strukturách stroje. Tak např. adresace musí umožňovat kategorizaci přístupu k objektu nebo subobjektu, zatímco ochrana informací uvnitř datových struktur stroje může být realizována definovanými doménami přístupu či jinými mechanismy.

4.8 Strategie matematických operací

Vhodné zobrazení reálných čísel v plovoucí řádové čárce je základním předpokladem pro implementaci operací s těmito čísly v architektuře stroje. Obvykle tyto operace představují algoritmy, které jsou, podle požadavku na rychlost, realizovány buď v hardware nebo nejnižším software systému.

4.8.1 Operace sčítání a odčítání

Dva operandy vstupující do operace sčítání nebo odčítání v plovoucí řádové čárce se mohou lišit velmi výrazně ve své velikosti. K tomu aby operace byla realizovatelná, je nutno před provedením sčítání nebo odečítání docílit stejných exponentů u obou čísel. Předpokládejme, že v operaci

$$M_1 . 2^{E_1} + M_2 . 2^{E_2}$$

je $E_1 \geq E_2$. Abychom docílili stejných exponentů, musí být buď mantisa M_1 posunuta doleva a odpovídajícím způsobem snížen exponent E_1 nebo opačná operace musí být provedena s operandem "2". Jestliže přistoupíme na první volbě, pak může dojít k tomu, že operand M_1 přeteče (zvláště, když již byl v normalizované tvaru). Jediným východiskem z této situace je denormalizovat druhý operand[14]. Oba operandy budou tedy uvedeny na stejný exponent a může dojít k jejich sečtení nebo odečtení. Po této operaci však nemusí nutně být výsledek v normalizovaném tvaru a bude jej nutno znovu normalizovat. Proces opětné normalizace nazýváme *post-normalizací*. Během procesu denormalizace a sčítání/odčítání může dojít

- k přetečení mantisy, což se obvykle řeší přidáním zvláštního bitu nad nejvýznamnější bit; podobně může dojít i k podtečení mantisy,
- k přetečení exponentu při post-normalizaci, když předtím dojde k přetečení nebo podtečení mantisy,
- ke ztrátě přesnosti, která je způsobena malou délkou pole mantisy a při porovnávání dvou hodnot se tyto mohou jevit identické
- k dalším anomáliím, které mohou vzniknout při nevhodně zvolených krocích algoritmu - např. bude-li rozdíl dvou operandů nula a nebude-li provedena kontrola tohoto stavu před započetím normalizace výsledku, dojde ke spuštění nekonečné smyčky.

4.8.2 Operace násobení a dělení

Násobení a dělení čísel v plovoucí řádové čárce lze popsat rovnicemi

$$(M_1 . 2^{E_1}) . (M_2 . 2^{E_2}) = (M_1 . M_2) . 2^{E_1+E_2}$$

$$\frac{M_1 . 2^{E_1}}{M_2 . 2^{E_2}} = \frac{M_1}{M_2} . 2^{(E_1-E_2)}.$$

Je zřejmé, že může dojít k přetečení nebo podtečení a proto je nutno kontrolovat, zda operand, vstupující do operace jako dělitel, je nenulový. Při násobení je možno násobit absolutní hodnoty a teprve do výsledku doplnit znaménko, nebo násobit čísla se znaménkem přímo. Vlastní násobení může být prováděno sekvenčně, krok po kroku, a nebo paralelně v jednom kroku.

Jako velmi nepraktické se jeví nutnost rozšíření obrazů násobence i násobitele, což se projeví nutností rozšíření znaménkového bitu doleva u záporných čísel, tedy i záporných dílčích součinů, a zbytečným rozšířením počtu operací o násobení všemi bity rozšířeného záporného násobitele. Takový postup je velmi nevýhodný a proto se hledaly vhodné algoritmy, které by jej zjednodušily. Jedním z těchto algoritmů, často používaným, je tzv.

Pro tento proces se používá anglický termín "scaling".

Boothovo překódování . Základem Boothova algoritmu je překódování násobitele do soustavy relativních čís 0,1,-1.

Příklad

Máme-li násobit číslem 63 = **00111111**, použijeme Boothova překódování a budeme násobit číslem (64-1) zapsaným v soustavě relativních číslic jako **0100000-1**. Podobný princip můžeme aplikovat na všechny skupiny jedniček, přičemž za skupinu můžeme považovat i jedinou jedničku. Výsledkem je překódovávací tabulka, kdy nejjednodušší je překódovávací tabulka nazývaná "s radixem 2" obsahuje dvojice bitů překódované následovně

překódovávaná číslice	číslice vpravo	Boothův kód
0	0	0
0	1	1
1	0	-1
1	1	0

Boothovo překódování je jednou z metod usnadňujících výpočet násobení, podobně jako např. vroubení p zamezení šíření znaménka vlevo a další. Avšak tyto metody jsou mimo rozsah tohoto textu a čtenář je může na např. ve [S1].

pochopen, pak zajisté bude.
A. Bloch: "Murphy's Law"

5 Instrukce

Pod pojmem instrukce obvykle rozumíme řetězec symbolů, které při interpretaci procesorem způsobí jednoznačnou a definovanou změnu stavu stroje. Obecněji můžeme chápat instrukci jako typ dat a pak instrukce vykazuje všechny charakteristiky datových typů tak, jak byly uvedeny v kapitole "Reprezentace dat" na straně 37:

1) Jednoznačně pojmenovaná buňka dat s definovaným formátem (syntaxí) tvoří bitový obraz kódu určující entitu, která je zde uložena. Každá instrukce se přitom sestává ze dvou částí:
   a) operačního kódu, který specifikuje, jaká změna stavu bude docílena provedením této instrukce,
   b) explicitně nebo implicitně specifikované identifikátory v adresovém prostoru počítače, které mají být při této očekávané změně stavu použity jako parametry.
2) Soubor pravidel, která určují, jaké manipulace jsou povoleny s tímto typem dat. K těm obvykle dochází před tím, než jsou instrukce ukládány do paměti, a jejich cílem je modifikace způsobu, jak instrukce bude později strojem interpretována. V minulosti používali programátoři možnosti změny instrukcí během práce programu, a to tak, že instrukce byly modifikovány pomocí jiných instrukcí. V současnosti většina systémů takovou práci zakazuje a některé architektury ji ani neumožňují.
3) Soubor pravidel definující způsoby a oprávnění přístupu k těmto instrukcím bude tento typ dat zcela evidentně splňovat.

Interpretace instrukce procesorem je jediný způsob, jak změnit stav počítače, pokud odhlédneme od možných chyb, které mohou způsobit neadekvátní změny, nebo od externích příčin např. přerušení. Vykonaná změna stavu stroje se promítá do dvou oblastí - výsledku operace a nastavení příznaku. Příznak, obvykle stav některého z registrů architektury stroje, charakterizuje průběh ukončené operace a může být používán jako vstupní parametr pro následující operaci.

Klasifikovat instrukce, je možné z několika hledisek. Uvědomíme-li si, že instrukce bude mít vždy tvar

<operační kód> : <operand 1> <operand 2> ... <operand n>

pak můžeme vyjít ze dvou základních hledisek. První z nich bude instrukce dělit podle jejich instrukčního kódu a druhé podle počtu a druhu operandů instrukce. První dělení podle instrukčního kódu velmi úzce souvisí s architekturou návrhu procesoru, kde můžeme očekávat tři základní typy:

- zásobníková
- střadačová
- registrová.

Většina procesorů je obvykle používá smíšenou koncepci, tedy kombinaci několika možností uložen operandů. V případě zásobníkové architektury, může být **zásobník** vytvořen buď vyhrazením úseku v hlavn paměti a doplněním dalších speciálních ovládacích registrů, případně konstrukcí speciální paměti včetně těchto registrů. Pojem zásobníku jako speciální paměti typu LIFO[1] byl poprvé použit v počítači B5000, kde sloužil jako rychlá paměť pro přechodné uložení lokálních proměnných.

Střadačová koncepce je v historii počítačů snad nejrozšířenější a nejznámější. Základem této koncepce je speciální registr - **akumulátor**, nebo-li střadač. Většina instrukcí z instrukčního souboru se potom orientuje na operace s tímto střadačem, přičemž mezi základní operace patří uložení obsahu paměti do střadače, uložení obsahu střadače do paměti, aritmetické operace mezi obsahem střadače a obsahem vybraného místa v paměti nebo inkrementace obsahu střadače.

Registrové architektury používají obvykle sadu takových střadačů, kterou nazývají registrovou sadou. Jednotlivým registrům v této sadě mohou být přiřazeny speciální úkoly, jako je adresace dat, indexace dat apod.

---

[1] LIFO - Last In First Out, název pro paměť, kde k uloženým datům je možno přistupovat pouze sekvenčně a to tak, že data uložená naposledy jsou k dispozici jako první.

5.1 Instrukční repertoár

Chování počítače je zcela determinováno druhem a pořadím operací tak, jak je procesor provádí na základě příkazů předávaných mu prostřednictvím instrukcí. Existuje množina interpretačních pravidel, implementovaná obvykle v hardware procesoru nebo v jeho mikroprogramu, která zcela přesně popisuje způsob chování procesoru, jenž je determinován instrukcí předloženou procesoru ke zpracování. Rozhodujícím pro cílový stav stroje po interpretaci instrukce bude ta část instrukce, která se nazývá **operační kód**. Na základě analýzy operačního kódu můžeme jednotlivé instrukce rozdělit do několika skupin. Základní skupinu přitom budou tvořit instrukce, které spadají do skupiny instrukcí dostupných kterémukoliv programu. Tyto instrukce, resp. tuto skupinu instrukcí budeme nazývat **instrukcemi neprivilegovanými**. Druhá skupina instrukcí, které mohou být používány pouze operačním nebo jiným řídicím systémem, bude skupina **privilegovaných instrukcí**.

Obě části instrukce, operační kód i operand, mohou podléhat různým variacím, které nazýváme modalitami. **Modalita** bude ovlivňovat způsob, jak bude instrukce procesorem interpretována a jaký bude její konečný efekt. Modality mohou být dvou typů

- modalita, která způsobí změnu nebo modifikaci operačního kódu a
- modalita, která bude měnit nebo modifikovat adresy operandů.

Obecně tedy můžeme zapsat

```
<operační kód> [modalita] : <operand> [modalita]
```

Jako příklad **modality operačního kódu** můžeme použít instrukci posunu, která specifikuje počet kroků a směr posunu. Jako příklad **modality operandů** může posloužit specifikace, která bude ukazovat pouze na příslušnou část vyzvednutého operandu. Význam modality operačního kódu vyplývá z možnosti změnit individuální operační kód specifickým způsobem. Podobně význam modality operandu vyplývá z možnosti změnit konečnou hodnotu, která se účastní příslušné operace.

5.1.1 Variace operačních kódů

Většina operací může být implementována jako řada velmi podobných instrukcí nebo jako jedna instrukce, která bude mít různé modality. Pro příklad lze uvést instrukci **`Load/Store`** jako modalitu operace **`Move`** nebo instrukci **`Add Memory/Add Register`** jako modalitu operace **`Add`**. Jiným případem implementace modality operačního kódu je použití jednotlivých částí operačního kódu pro volbu mezi typy operací a mezi oblastmi, kde bude uložen výsledek operace. Pro objasnění významu modality při konstrukci operačního kódu předpokládejme operace

`Add Register to Register`	**`01010`**	**`<operand 1>,<operand 2>`**
`Add Memory to Memory`	**`00010`**	**`<operand 1>,<operand 2>`**
`Add Memory to Register`	**`11010`**	**`<operand 1>,<operand 2>`**
`Add Register to Memory`	**`10010`**	**`<operand 1>,<operand 2>`**

V tomto případě bit nejvíce vlevo tvoří modalitu instrukce s významem

0 → oba operandy stejného typu
1 → oba operandy různého typu

druhý bit nejvíce vlevo určuje místo uložení výsledku takto

0 → výsledek bude umístěn do paměti
1 → výsledek bude umístěn do registru

Výše uvedené konfigurace bitů můžeme též chápat jako čtyři různé operační kódy. Avšak při konstrukci operačního kódu hraje modalita velmi důležitou roli, neboť zavádí do této konstrukce jistý řád. Budeme-li pokračovat v předcházejícím příkladu tak, že použijeme další bit pro specifikace typu operace, kde **0** bude znamenat sčítání a **1** odčítání, obdržíme další čtyři typy instrukcí

`Subtract Register from Register`	**`01011`**	**`<operand 1>,<operand 2>`**
`Subtract Memory from Memory`	**`00011`**	**`<operand 1>,<operand 2>`**
`Subtract Memory from Register`	**`11011`**	**`<operand 1>,<operand 2>`**
`Subtract Register from Memory`	**`10011`**	**`<operand 1>,<operand 2>`**

Jak je zřejmé, pojem modality je pojmem dosti subjektivním. Avšak jeho použití při vytváření operačních kódů zcela jasně vyhraní skupiny instrukcí s podobnou či zcela odlišnou funkcí. Obecně lze najít tři základní skupiny nejčastěji používaných modalit:

- **Aritmeticko-logické modality**, umožňující volbu například mezi způsobem úpravy výsledku, zdali se bude jednat o zanedbání některých míst nebo jejich zaokrouhlení, stejně tak jako mohou ukazovat na různé způsoby pro nakládání se zbytkem po aritmetické operaci. Nebude ani na škodu si uvědomit, že pomocí modality můžeme určovat i datové typy a případně i číselné základy proměnných, účastných na aritmetické logické operaci.
- **Modalita instrukcí posuvu a rotace** bude poskytovat údaje o směru posunu nebo rotace a o jejím rozsahu. Rozšířením modality této skupiny instrukcí můžeme označit i případ, kdy znaménkový bit bude při operace posunu nebo rotace zahrnut, případně kdy zahrnut nebude (aritmetický typ operace). V případě operace posuvu modalita umožňuje i specifikovat element, který bude zaveden do příslušného pole operandu na opačném konci, než je směr jeho posuvu.
- **Modality použité pro instrukce řízení programu** jsou obvykle spojeny s instrukcemi podmíněného skoku a podmiňují nebo specifikují podmínky, za kterých by ke skoku mělo či nemělo dojít.

5.1.2 Variace operandů

Efektivní adresou operandu rozumíme místo v paměti, kde se nacházejí požadovaná data. Obvykle se odvozuje z obsahu operandu, který je součástí instrukce a může být podrobena celé řadě změn, kterých se mohou účastnit i jiné datové struktury. Tedy, **efektivní adresa operandu** může být též chápána jako modifikace obsahu operandu jinými datovými strukturami. Můžeme rozeznávat:

- **Indexovou modalitu**, umožňující např. pohyb mezi jednotlivými elementy pole.
- **Zásobníkovou modalitu**, umožňující volbu mezi uložením údaje do zásobníku a nebo vyzvednutím tohoto údaje ze zásobníku.
- **Modalitu nepřímého adresování**, což je situace, kde operand neobsahuje požadovaná data, ale spíše ukazatel na místo v paměti, kde se požadovaná data nacházejí. Mnohé počítače umožňují řetězení těchto ukazatelů, a tedy poměrně značnou hloubku nepřímého adresování. Omezení hloubky nepřímého adresování může být řešeno několika způsoby, kde mezi nejjednodušší patří použití zvláštního bitu v adrese, který bude určovat, zdali obsah příslušného slova je chápán jako ukazatel nebo jako efektivní adresa operandu.

5.1.3 Modalita polí operandu

V počítačích, které používají operandové struktury implementované jako řetězec bytů, většina instrukcí obsahuje pole, specifikující počet bytů v operandu - to znamená délku operandu. Toto pole můžeme chápat jako modalitu operandu. Některé starší bytově adresovatelné počítače používaly zvláštní konfiguraci bitů jako indikaci konce operandu (například IBM 1401). Ve strojích, které jsou slovně orientovány, velmi často existuje modalita jako součást operačního kódu, která specifikuje jistou část celého slova jako operand. Interpretace této části, zda se bude jednat o příslušný počet bitů, bytů, části slova či násobku slov, záleží pak již na procesoru interpretujícím tuto instrukci.

5.1.4 Modality ochrany paměti

Stroje, které jsou navrženy pro práci v multiprogramovém režimu, mají rezervovánu část paměti, která je dostupná pouze některým programům. Tato část paměti může být specifikována buď jako paměť pro zápis a čtení nebo pouze pro čtení a nebo jako zcela nepřístupná jakémukoliv programu s výjimkou operačního systému. Oblasti s omezeným přístupem, jako jsou oblasti určené pouze pro čtení nebo oblasti zcela nedostupné jsou používány operačním systémem. Zároveň musí být zajištěno, že uživatel X nesmí přistupovat do paměti kterou používá uživatel Y, pokud to uživatel Y nepovolil. Na druhé straně operační systém musí mít přístup do celé paměti, a to jak pro čtení, tak i pro zápis. To lze zajistit např. tím, že poskytneme speciální bit nebo pár bitů každé instrukci, a tak definujeme její přístupová práva. Tento bit **modality přístupových práv**, nebo **modalit**

chrany paměti, může být změněn pouze privilegovanou instrukcí, která není dostupná aplikacím. V tomto případě uživatel X může získat přístup do oblasti náležející uživateli Y pouze tak, že požádá operační systém, aby provedl přístup do této části pro uživatele X, pokud to uživatel Y dovolil. Jiný způsob ochrany paměťových oblastí může být zajištěn pomocí speciálních registrů, které budou označovat počátky a konce chráněných oblastí paměti.

5.2 Operandy instrukcí

Podle syntaxe pole operandu instrukce je možné rozdělit typy instrukcí podle toho, kolik a jakých operandů každá instrukce obsahuje, resp. jakou transformací tyto operandy projdou během změny stavu stroje. Z historického hlediska je zřejmé, že celá řada formátů operandů instrukcí vyplývala z architektonického návrhu a technologického omezení těchto počítačů. Například uložení instrukcí na povrchu otáčejícího se magnetického bubnu vyžadovalo, aby sekvence instrukcí vzala v úvahu i rotační zpoždění záznamového zařízení, a tak pro zajištění dostatečně efektivního programování nemohly jednotlivé instrukce ležet souvisle za sebou. To vedlo k návrhu instrukcí, jejichž jeden operand určoval polohu následující instrukce na tomto rotačním magnetickém paměťovém zařízení.

5.2.1 Čtyřadresové instrukce

Čtyřadresová instrukce má tvar

OPCODE	M_1	M_2	M_3	M_4

a bude prováděna následujícím způsobem

$[M_3] \blacktriangleleft [M_2]\ op\ [M_1]$[2]

IA[3] $\blacktriangleleft [M_4]$

Čtyřadresové počítače nedoznaly praktického využití, s výjimkou konstrukcí v počátečním rozvoji oboru, kdy se mnohoadresové instrukce vyskytovaly (např. SAPO s pětiadresovou instrukcí). Obdoba této instrukce známá jako "3 + 1" s primární pamětí realizovanou rotačním magnetickým bubnem nebo akustickou zpožďovací linku, používala speciální registr, který umožňoval posun následující adresy s uvažováním rotačního zpoždění paměťového zařízení.

5.2.2 Tříadresové instrukce

Tříadresová instrukce má typický tvar

OPCODE	M_1	M_2	M_3

a zpracovává se podle předpisu

$[M_3] \blacktriangleleft [M_2]\ op\ [M_1]$

$PC \blacktriangleleft [PC] + 1$

Tříadresový počítač byl navržen jako výsledek zavedení technologie pamětí na feritových jádrech. Jejich vysoká cena vedla konstruktéry k tomu, aby maximálně šetřili s pamětí, zatímco tlak zákazníků, zejména z oblasti nukleární fyziky, požadoval co největší adresový prostor. Možnost zápisu výsledku přímo do zvolené oblasti v paměti byla rovněž daleko přirozenější a umožňovala tehdejším programátorům programujícím ve strojovém kódu používání celé řady triků. Tříadresové počítače však zcela nezapadly a i takové stroje jako superpočítače typu Cray používají tříadresové instrukce.

5.2.3 Dvouadresové instrukce

Obecná forma dvouadresové instrukce je

OPCODE	M_1	M_2

a při její interpretaci se výsledek ukládá na místo jednoho z operandů podle předpisu

$[M_1] \blacktriangleleft [M_2]\ op\ [M_1]$

$PC \blacktriangleleft [PC] + 1$

Návrh dvouadresových instrukcí souvisí se zavedením primárních pamětí používajících polovodičové paměti s náhodným přístupem. Tento typ instrukce je velmi populární zejména u malých počítačů orientovaných na kancelářské aplikace. Z hlediska programátora se při použití dvouadresového počítače jeví každá paměťová oblast jako akumulátor. Moderní implementace dvouadresového schématu zahrnuje instrukce typu paměť-paměť, stejně jako registr-registr, paměť-registr nebo registr-paměť. Fyzická existence střadače v těchto architekturách není nutná a pokud se používají dvouadresové instrukce, jsou velmi často spojeny s registrovou sadou.

5.2.4 Jednoadresové instrukce

Typický tvar jednoadresové instrukce je

OPCODE	M

a při její interpretaci se používá akumulátor

$[ACC] \blacktriangleleft [ACC]\ op\ [M]$

$PC \blacktriangleleft [PC] + 1$

Jednoadresová instrukce je typická pro klasickou von Neumannovu architekturu a celá řada hypotetických počítačů pro výukové účely používá jednoadresovou instrukci jako dobrý příklad čistého návrhu. Není však mnoho počítačů, které by používaly pouze jednoadresové instrukce. I když existuje jisté pokušení zahrnout mezi jednoadresové instrukce instrukce pracující s zařízením vstupu a výstupu, musíme si uvědomit, že i v tomto případě existují dvě adresy. Jedna adresa pro oblast paměti, kde jsou uložena, nebo mají být uložena data, a druhá adresa, která identifikuje příslušné vstupní a výstupní zařízení.

5.2.5 Bezadresové počítače

Bezadresové počítače jsou typické pro zásobníkovou architekturu a jsou to instrukce, které mohou být implementovány pouze na této architektuře. Typický způsob manipulace se zásobníkem si vyžádal i specifický způsob zápisu těchto operací, který na rozdíl od našeho běžného zápisu - **infixní notace** - nazýváme notace **polskou**[4] a rozeznáváme dva druhy zápisu - **postfixní**, kdy operátory jsou zapisovány za operandy a **prefixní** s zápisem operátorů před operandy. Výhodou polského zápisu je, že nepoužívá závorky a asociuje operandy s operátory jednoduchým způsobem. Každá trojice obsahující operátor a operandy je zapsána tak, že dva operandy jsou následovány operátorem. Takže například infixní zápis

```
(B * (C + D)) / (E + F)
```

může být v polské postfixní notaci zapsán jako

```
BCD+* EF+/
```

a v polské prefixní notaci jako

```
/*+ CDB +EF
```

Protože počítač používající bezadresové instrukce je organizován okolo zásobníku, je nutno zajistit, aby existovaly instrukce uložení dat na vrchol zásobníku a instrukce vyzvednutí informace z vrcholu zásobníku stejně tak jako operace, které budou s hodnotami uloženými na vrcholu zásobníku pracovat. Poznamenejme ještě, že lze teoreticky dokázat, že počítač, který by obsahoval pouze bezadresové instrukce, není možno zkonstruovat. Je to zřejmé při elementární úvaze, že proměnnou, kterou chceme uložit na vrchol zásobníku musíme někde získat, a stejně tak výsledek z operace na zásobníku musíme časem někam uložit.

---

[2] *op* označuje libovolnou operaci provedenou s těmito operandy

[3] IA označuje místo, kam bude zavedena adresa příští instrukce

[4] Tento druh zápisu byl pojmenován po polském matematikovi a logikovi Janu Lukasiewiczovi

5.3 Skupiny instrukcí

Instrukce můžeme rozdělit do několika skupin, které se budou lišit obecným stavem stroje po jejich interpretaci. Dále uvedené rozdělení popisuje jednu z možných a používaných taxonomií, avšak v žádném případě si nečiní nárok na to, být zcela úplné.

5.3.1 Instrukce přesunu dat

Základní tvar instrukcí přesunu dat je

Instrukce přesunu	:	Operandy
Zdroj dat/Cíl dat		Adresové modifikátory
Modifikátory		Druh přesouvané entity
		Délka

V jednoadresovém stroji budou mít instrukce typu **`MOVE`** jednoduchý explicitní operand. Ve strojích s více než jedním registrem bude požadována specifikace obou operandů. Obvyklé a nejčastější instrukce tohoto typu zajišťují přesun operandů nebo jejich obsahu mezi dvěma místy v paměti a typicky mezi ně patří instrukce typu **`Load Register`** a nebo instrukce typu **`Store Register`**.

Modalita této skupiny instrukcí se může vztahovat na oblast zdroje nebo oblast místa určení dat a může být specifikována operace přesunu typu *register-register*, *register-paměť* atd. Adresová modalita skupiny instrukcí přesunu dat může zahrnovat všechny možné variace od uložení konstanty přes nepřímou adresaci až po nejrůznější permutace elementárních možností. V případech, kdy zdrojová oblast a cílová oblast budou mít různou délku, musí být při interpretaci instrukce specifikována pravidla, jakými bude tato situace řešena. Do této skupiny rovněž patří instrukce přesunu s modifikací, kdy přesouvaná data jsou během přesunu data nějakým způsobem modifikována. Příkladem může být například instrukce **`Move Complement`** nebo **`Load Negativ`**, které v obou případech přesunou do cílové oblasti negaci hodnot uložených ve zdrojové oblasti. Specifickým případem instrukce přesunu může být instrukce typu **`Exchange`**, která vymění dvě oblasti dat mezi sebou.

5.3.2 Skupina aritmeticko-logických instrukcí

Tato skupina instrukcí obsahuje sadu aritmetických a logických operací implementovaných na počítači se základním tvarem

Aritmeticko-logická instrukce	:	Operandy
Do/Z		Adresové modifikátory
Mod práce		Druh zpracovávané entity
Zbytek, zaokrouhlení apod.		

Aritmetické operace jsou obecně implementovány jako sčítání, odčítání, násobení, dělení a aritmetika jednotkových, dvojkových nebo decimálních doplňků. Obvyklé logické operace jsou unární operace zahrnující jednu proměnnou, stejně jako šestnáct bolovských binárních operací se dvěmi proměnnými. V této souvislosti je vhodné poznamenat, že veškeré logické operace mohou být prováděny s využitím dvou základních operací, jedné unární operace inverze (**`not`**) a jedné binární operace logického součinu nebo logického součtu (**`and`** nebo **`or`**). Do této skupiny instrukcí lze rovněž zařadit instrukce modifikující explicitně zadaný operand aritmetickou nebo logickou operací s přímým operandem v instrukci. Některé počítače požadují, aby tento přímý operand byl specifikován implicitně. V tom případě se setkáváme s operacemi např. **`Set Bit`**, **`Clear Buffer`** apod.

Použité zobrazení jednoho nebo obou operandů může být podle návrhu počítače binární, decimální nebo jiné a podobně kterýkoliv z operandů může být celým číslem, znakem nebo reálným číslem v příslušném zobrazení. Operace mohou být prováděny na délce počínajíce jedním bitem a obvykle konče délkou odpovídající násobku nejmenších adresovatelných jednotek. Jestliže budou operandy různé délky, pak musí existovat pravidla, jak se tato situace bude řešit. Výsledek aritmetické operace může vést ke zbytku při dělení nebo k výsledku násobení, který nebude možno uložit do struktur počítače, popřípadě při normalizaci čísla v plovoucí řádové čárce bude výsledek uložen na nevýznamných místech reprezentace tohoto čísla. V takovýchto případech musí existovat explicitní modalita instrukce nebo implicitní konvence, které budou specifikovat, jakým způsobem bude s tímto výsledkem naloženo. Instrukce musí specifikovat zdali přebývající bity budou uloženy do jiného registru, nebo budou zanedbány, popřípadě jestli výsledek bude zaokrouhlen apod. Modalita této skupiny instrukcí může být velmi široká, počínajíce modalitou zdrojové a cílové oblasti a konče modalitou specifikující způsob nakládání s výsledkem aritmetické operace.

5.3.3 Skupina instrukcí posuvu a rotace

Základní tvar instrukcí této skupiny je

Posuv a rotace	:	Operandy
Směr		Paměť/Registr
Rozsah posuvu/rotace		Délka operandu
Mod aritmetický/logický		Rozsah
Posouvaný element		
Vyplňující element		

Primárním rozdílem mezi posuvem a rotací je, že při posuvu se informace, která je vysouvána mimo rozsah paměťové jednotky, ztrácí zatímco při rotaci se informace opouštějící na jedné straně položku dat objevuje na jejím druhé straně. Operace posuvu tedy uloženou informaci ničí, zatímco při operaci rotace lze původní informaci opět obnovit. Operace posuvu i rotace mohou probíhat na obě strany, tj. doleva i doprava a mohou být aritmetické nebo logické. Aritmetická operace posuvu nebo rotace nezahrnuje znaménko, resp. znaménkový bit operandu, zatímco logická operace posuvu nebo rotace zachází se znaménkovým bitem stejně jako s jakýmkoli jiným. Rozsah posuvu nebo rotace se vyjadřuje v násobcích základního posouvaného elementu, obvykle bitech.

Operace logického posuvu nebo rotace předpokládá, že znaménkový bit je posouván doprava nebo doleva a jeho pozice je zaujata jiným bitem, který bude následovně interpretován jako znaménkový bit. Při aritmetickém posuvu datové položky, kde tato datová položka bude reprezentovat celé číslo, bude tento posuv interpretován jako násobení nebo dělení tohoto čísla mocninou dvou[5]. Algoritmy používající aritmetický posuv se často vyskytují v místech, kde zastupují operaci násobení nebo dělení. Při operaci posuvu ať již aritmetické nebo logické se bity na jedné straně ztrácejí a na druhé straně nově vznikající bity musí být nahrazeny odpovídajícím elementem. Tento element, tedy jedničkový nebo nulový bit, může být implicitně specifikován pro daný počítač nebo může být součástí instrukce posuvu[6].

Velmi specifickou instrukcí v této skupině je instrukce sloužící pro normalizaci čísel v plovoucí řádové čárce. Instrukce normalizace u počítačů, které používají zobrazení mantisy s řádovou čárkou zcela vlevo posouvá tuto mantisu o jednotlivé elementy směrem k nejvýznačnějšímu bitu, zatímco exponent je snižován. Operace posuvu bude ukončena ve chvíli, kdy se objeví na místě nejvýznačnějšího elementu nenulové číslo. Tento element bude jednobitový v případě binárních počítačů, tříbitový u oktalové reprezentace a čtyřbitový v případě použití hexadecimálního základu exponentu nebo zobrazení BCD. Při normalizaci musí být jak vyplňující elementy použity nuly, čímž dojde k redukci přesnosti aritmetiky v plovoucí řádové čárce. Některé návrhy procesorů používají dodatečné bity při práci s čísly v plovoucí řádové čárce, které jsou uchovány v zvláštním registru, a při posunu mohou být přesouvány z tohoto speciálního registru do mantisy zobrazovaného čísla, čímž se zajistí zvýšená přesnost. Tento způsob byl použit například v počítači IBM 360, a používají jej rovněž mikroprocesory Intel ve své matematické části[7].

---

[5] Samozřejmě, že jenom v případě, kdy je pro reprezentaci použit binární obraz.

[6] Toto je příklad existence modality ve skupině instrukcí rotace a posuvu.

[7] Bity doplňované ze zvláštního registru se podle autora, Herondo Kuki z University of Chicago, nazývají "Kuki bits".

5.3.4 Skupina skokových instrukcí

Předpokládáme-li instrukce skoku ve tvaru

Skoková instrukce :	Operandy
Podmínka	Explicitní/Implicitní
	Přímý/Nepřímý
	Literál/Proměnná

ak instrukci skoku a vynechání (**Skip**) můžeme považovat za téměř identické s výjimkou případu, kdy tato nstrukce explicitně specifikuje alternativní adresu příští instrukce. Pro instrukci vynechání, **Skip**, je alternativní dresa příští instrukce implicitní a nachází se v definované vzdálenosti relativně od adresy této instrukce. Může šak existovat i takový případ, kdy instrukce **Skip** bude udávat přímo rozsah instrukcí, které mají být řeskočeny ve svém operandu. Jak instrukce **Skip**, tak instrukce skoku, **Jump**, mohou používat nepřímé dresování.

Typické podmínky spojené s podmíněnými skoky nebo vynecháním jsou testy na nulový obsah, pozitivní či egativní obsah, testování specifického podmínkového bitu apod. Některé počítače, například známý IBM 360, bsahují speciální registr podmínkového kódu, ve kterém se nastavují specifické hodnoty, podmínkové kódy, ako výsledek předcházejících operací. Zvláštní místo patří instrukcím skoku, které slouží pro řízení cyklu, jako apř. instrukce typu **Repeat** nebo **Do-count**. Rovněž instrukce **Loop** patří do této skupiny a je používána rocesory Intel 8086.

Významnou podskupinu této skupiny tvoří instrukce pro volání podprogramů, kde se nejčastěji provádí kok na druhou adresu v podprogramu, zatímco současný stav programového čítače zvětšený o jedničku je ložen do první adresy podprogramu. Ačkoliv tento koncept vstupu do podprogramu lze jednoduše mplementovat a je snadno pochopitelný, omezuje rekurzivní volání podprogramů a opomíjí základní principy tvorby reentrantního kódu. Daleko elegantnější řešení je využití zvláštní oblasti pro každý program, která se bude chovat jako zásobník pro programové čítače, registry a další informace, které je nutno uschovávat během volání podprogramů. Instrukce návratu z podprogramu pak vybere obsah tohoto zásobníku a obnoví stav počítače do podoby před voláním příslušné procedury, avšak ovlivněný výsledkem této procedury.

Poslední specifickou instrukcí této skupiny je instrukce **Execute**, která vykoná instrukci uloženou na adrese, na kterou ukazuje operand této instrukce. V konečném efektu se jedná vlastně o podprogram, který je implementován jednoduchou instrukcí. Typicky je možno najít i instrukce **Execute**, které budou adresovat další instrukce **Execute**, a tak umožňovat řetěz nepřímého adresování téměř libovolné délky. Do této skupiny můžeme zařadit i instrukci **Noop**, tedy příkaz procesoru, aby zůstal ve stavu, ve kterém se nacházel před prováděním této instrukce. Obě uvedené instrukce mohou mít svoji podmíněnou verzi, tedy **Noop-If**, nebo **Execute-If**. Možnost modality těchto instrukcí je velmi široká vzhledem k rozsahu jejich různých funkcí.

5.3.5 Instrukce pro operace se zásobníkem

Pro operaci se zásobníkem lze instrukce předpokládat ve tvaru

Operace se zásobníkem :	Operandy
Do/Z	Adresový modifikátor
Vrchol/Dno	Druh dat

s nejčastější implementací dvou základních instrukcí pro uložení na zásobník (**Push**) a vyzvednutí hodnoty z vrcholu zásobníku (**Pop**). Mimo tyto operace se vyskytují ještě často speciální operace se zásobníkem, jako například odstranění položky dat z vrcholu zásobníku, nebo záměna prvních dvou položek na zásobníku. Do této skupiny rovněž patří operace s ukazatelem zásobníku, jehož modifikací se může měnit nejenom pozice okamžitého vrcholu zásobníku, ale i pozice jeho dna a velikost. V některých strojích je implementováno i více zásobníků a pro různé zásobníky jsou používány různé instrukce nebo jsou zásobníky odlišeny operandy.

5.3.6 Instrukce řízení stroje

Tato skupina instrukcí nastavuje některé podmínky, které jsou nutné pro změnu stavu stroje nebo pr správné vykonání následující instrukce a její syntaxe se stroj od stroje liší. Jako příklad můžeme použít instruk **Mode Control**, která se vyskytovala u počítačů s velmi omezeným instrukčním repertoárem a sloužila k změně způsobu zpracování následující instrukce. Tak mohlo dojít k nastavení podmínky, že následující instrukc součtu (**Add**) pracovala v jednom případě s čísly celými a ve druhém případě s čísly v plovoucí řádové čárc Existuje celá řada dalších instrukcí, které mohou sloužit například k analýze stavu stroje s cílem určit důležit momenty práce multiprogramovaného systému, nebo instrukce diagnostiky speciálně používané pro technick personál. Konkrétně diagnostika se u mikroprocesorů rozvinula do té míry, že existují speciální instrukc přístupné a realizované prostřednictvím tzv. **testovacího portu**, které jsou interpretovány zvláštní čás procesoru a umožňují diagnostikovat složité procesorové obvody používané při hlavních výpočtech.

5.3.7 Skupina instrukcí vstupu a výstupu

Skupina operací vstupu a výstupu je po aritmetických a logických operacích nejčastěji jednou z největšíc skupin instrukcí v počítači. Jejich základní tvar můžeme očekávat

I/O operační kód :	Operandy/Zařízení
Vstup/Výstup	Adresový modifikátor
Ulož/Testuj/Uvolni zařízení	Délka dat
Zrušení operace	Druh dat
	Cílová trasa dat (řídící jednotka, zařízení)

Základními typy vstupně-výstupních operací je instrukce čtení (**Read**) a instrukce psaní (**Write**). Instruk **Read** umožňuje, aby specifikované vstupní zařízení poskytovalo data, která budou uložena do primární pamě na danou efektivní adresu, jež bude součástí této instrukce, nejčastěji její operand. Některé modifikovar instrukce tohoto typu umožňují, aby místo jedné položky se uložil do primární paměti celý blok dat. Instruk zápisu, **Write**, umožňuje, aby data z primární paměti byla poskytnuta specifickému výstupnímu zařízení pr další zpracování. Podobně jako v předchozím případě je možné tyto instrukce modifikovat tak, aby výstupnímu zařízení předávaly nikoliv jenom jednotlivá data, ale celé bloky dat.

Specifickou skupinou těchto instrukcí jsou tzv. **imperativy**, které nevedou přímo k přenosu dat na nebo zařízení, ale nastavují podmínky tohoto přenosu, popřípadě jiné charakteristiky zařízení. Mezi tento typ instruk patří například **Rewind** pro nastavení zařízení na začátek média, **Erase** pro vymazání média nebo **Abort** pr okamžité ukončení operace. Doplněním těchto imperativů mohou být instrukce podmíněného skoku pod stavu testovaného vnějšího zařízení. Na základě výsledku zpracování instrukce imperativu spojené podmíněným skokem se může program rozvětvit.

Modalita instrukcí vstupu a výstupu je opět velmi široká a může být reprezentována nejenom modalitc operandů, ať již půjde o instrukce jednoadresové nebo víceadresové, ale i modalitou vlastního operačního kód který může např. odlišovat mezi různými typy zařízení.

5.4 Instrukční repertoár - velikost, složitost a úplnost

Ačkoliv počítač s několika málo instrukcemi povede pravděpodobně programátora k tomu, aby konstruov úseky velmi složitě kódované, a tak produkoval program, který bude delší a pomalejší, návrh počítače se složitý a velkým instrukčním souborem nebude cílem, vzhledem k tomu, že

- programátoři nejnižších úrovní používající asembleru obvykle nezvládnou celou množinu instruk a budou mít tendenci používat svoji favorizovanou podmnožinu těchto instrukcí,
- v kompilátorech vyšších programovacích jazyků se obvykle používá pouze podmnožina instruk které jsou k dispozici, k tomu, aby se konstruovaly objektové moduly ze zdrojového kódu,
- bude narůstat délka kódu o další bity, které jsou nutné ke kódování dalších operačních kódů p tyto instrukce. I když argument, že změna repertoáru ze 128 do 256 instrukcí zahrnuje pouze jed

přidaný bit, a tak výrazně nepříspívá k obsazování paměti, bude zřejmě správný, je nutno si uvědomit, že přidáním jediného bitu do kódu instrukce se zvýší i složitost logiky pro dekódování této instrukce a její následovnou interpretaci.

Hodnocení instrukčního souboru může být prováděno z různých hledisek, můžeme jako příklad uvést hodnocení použité v (Marovac, [A8]), které definuje čtyři základní charakteristiky[8].

- **Konzistence**. Jestliže podmnožina instrukcí má vysoký sémantický obsah, pak celá množina musí mít odpovídající úroveň abstrakce. Jestliže podmnožina instrukcí umožňuje pracovat v různých adresových modech, pak totéž musí umožňovat celá množina instrukcí. Jestliže podmnožina instrukcí generuje jisté podmínkové kódy, pak celá množina instrukcí musí generovat tytéž podmínkové kódy s použitím téže syntaxe a téže sémantiky.
- **Podmínkové kódy**. Stroj musí obsahovat úplnou a ortogonální množinu podmínkových kódů k tomu, aby byly umožněny podmíněné skoky. Ve výše uvedené práci Marovac požaduje, aby explicitní podmínkové kódy byly přímo dostupné programátorovi v asembleru. Jiní autoři doporučují spíše řešení implicitních podmínkových kódů od úrovně asembleru nedostupných.
- **Šíře repertoáru a sémantický obsah**. Šíře repertoáru a sémantický obsah instrukce je jedním z charakteristických rozdílů mezi procesory s úplným a redukovaným souborem instrukcí. Přestože Marovac ve své práci doporučuje malou a kanonickou množinu instrukcí, velká většina používaných procesorů má rozsáhlý repertoár se specializovanými instrukcemi na vysoké úrovni abstrakce.
- **Účinnost**. Instrukční repertoár musí umožňovat zápisy algoritmů buď člověkem pomocí asembleru nebo pomocí kompilátoru do relativně malých bloků strojového jazyka, které se provádějí velmi rychle.

S ohledem na charakteristiku instrukčního souboru lze dále odvodit šest základních kroků při návrhu množiny instrukcí, které lze shrnout následovně:

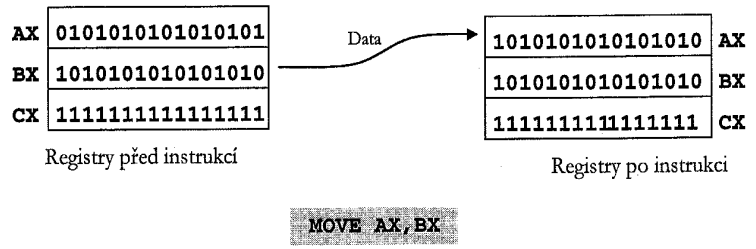
1. Analýza zamýšlené oblasti aplikace, tedy jakýsi výzkum trhu.
2. Stanovení měřitelných vlastností aplikace, charakteristiky programů a formulování nezbytných a požadovaných funkcí na makroúrovni.
3. Převedení každé funkce formulované na makroúrovni do primitiv na úrovni nejnižších kroků - instrukcí.
4. Určení nejčastěji se vyskytujících skupin instrukčních kroků - primitiv vytvořených na předcházející úrovni a jejich frekvence výskytu na funkční makroúrovni.
5. Frekvencím výskytu odvozeným v předcházejícím kroku přiřadit váhy podle očekávaného času výpočtu této funkce na makroúrovni. Primitiva - instrukční kroky - u kterých se očekává, že budou nejčastěji používány, dostanou přidělenu maximální preferenci při návrhu tak, aby jejich konečná implementace zajišťovala maximální účinnost zpracování instrukce v konečné realizaci.
6. Analyzovat posloupnosti instrukčních kroků navržené v předcházejících paragrafech a zhodnotit jejich přijatelnost na úrovni repertoáru instrukcí asembleru.

5.5 Způsoby adresování

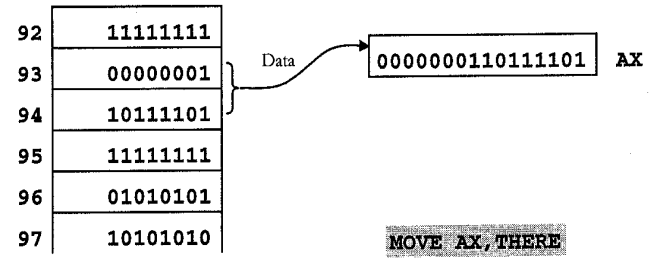
Každá instrukce pracující s daty musí obsahovat specifikaci, jaká manipulace má být s daty provedena, tedy operační kód, a označení dat, která mají být v této operaci použita. Toto označení nazývané operand může mít celou řadu variant. Jednotlivé varianty mohou obsahovat buď tzv. přímé operandy, kdy přímo hodnota použitá v operaci bude do tohoto operandu uložena nebo ukazovat na adresu, kde se příslušná položka dat nachází. Různé adresovací varianty a metody spojené s výpočtem efektivní adresy operandu mohou být rozděleny do několika skupin. V přípravné práci pro standard IEEE (Fisher, [A9]) , definující charakteristiky asemblerů pro použití

[8] Mimo uvedených charakteristik se často hovoří i o **úplnosti instrukčního souboru** , což zjednodušeně znamená, že procesor může realizovat kiaždou základní operaci pomocí jedné instrukce. **Ortogonalita instrukčního soub oru** zase vede k nezávislosti instrukcí, tedy např. existuje pouze jediný jednoduchý způsob jak provést konkrétní operaci.

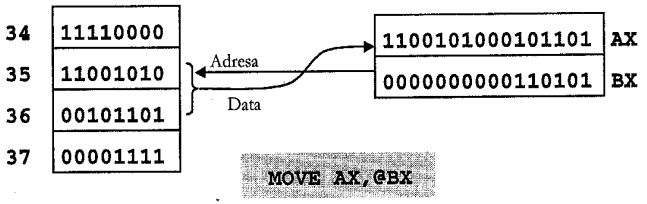
v mikroprocesorech, se vyskytuje devatenáct různých metod výpočtu efektivní adresy jak cílového, tak zdrojového operandu. Pro zjednodušení však lze shrnout těchto devatenáct adresovacích modů do čty základních skupin.



Obr. 38 Přímé adresování registrů



Obr. 39 Přímé adresování paměti, THERE=93



Obr. 40 Nepřímé adresování obsahem registru

5.5.1 Přímé adresování

Při přímém adresování je adresa dat přímo uložena do operandu instrukce. Tato adresa může b identifikátor příznaku, registr nebo adresa v hlavní paměti. Podle toho můžeme použít další dělení na **přímo adresaci příznaku, přímou adresaci registru a přímou adresaci místa v paměti**. Příklady přímé adresac jsou na obr. 38 a obr. 39.

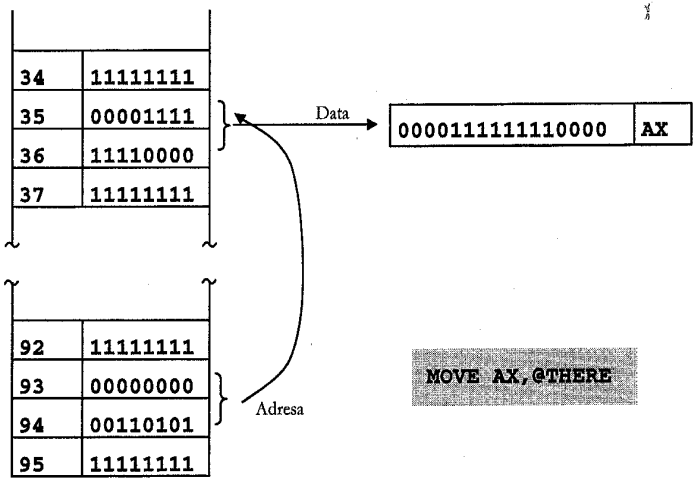
5.5.2 Nepřímé adresování

Nepřímé adresování umožňuje, aby nemodifikovatelná adresa uložená v operandu instrukce ukazovala r oblast v hlavní paměti nebo registr, kde je uložena adresa požadované datové položky. Podle druhu paměti, v které je tento ukazatel uložen, můžeme rozeznávat **nepřímou adresaci obsahem registru** (obr. 40) neb **nepřímou adresaci obsahem paměti** (obr. 41). V některých případech dochází k automatické inkrementa nebo dekrementaci hodnoty uložené v registrech buď před nebo po jejím použití. Variacemi tohoto postup získáme následující čtyři možnosti (viz obr. 42):

- **autopostinkrementace** registru, to znamená, že obsah registru, který ukazuje na data v paměti, bude vžc po jeho použití automaticky zvýšen o šířku datové položky po každém přístupu k datům,

- **autopostdekrementace** registru je obdobný postup jako v předcházejícím s tím rozdílem, že obsah registru bude automaticky snížen o šířku datové položky po každém přístupu k datům,
- **autopreinkrementace** registru bude znamenat, že obsah registru, který ukazuje na data v paměti, bude vždy před jeho použitím pro přístup k datům automaticky zvýšen o šířku datové položky,
- **autopredekrementace** registru je poslední možnou kombinací a znamená, že obsah registru, který ukazuje na data v paměti, bude vždy před jeho použitím pro přístup k datům automaticky snížen o šířku datové položky.

Zvláštním případem této skupiny adresace je adresace zásobníku, kdy ukazatel, který je uložen v registru, se o každém přístupu na zásobník mění. V případě uložení hodnoty na zásobník se zvyšuje (snižuje), v případě vyzvednutí hodnoty ze zásobníku se snižuje (zvyšuje)[9].



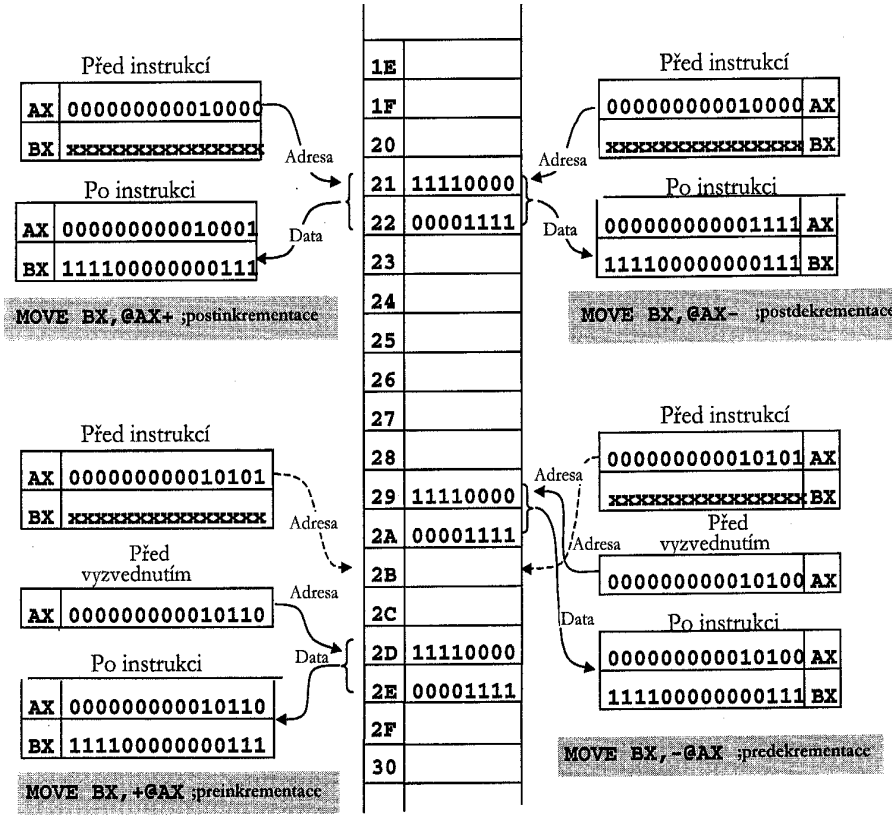
Obr. 41 Nepřímé adresování obsahem paměti, THERE=93

3.5.3 Relativní adresování

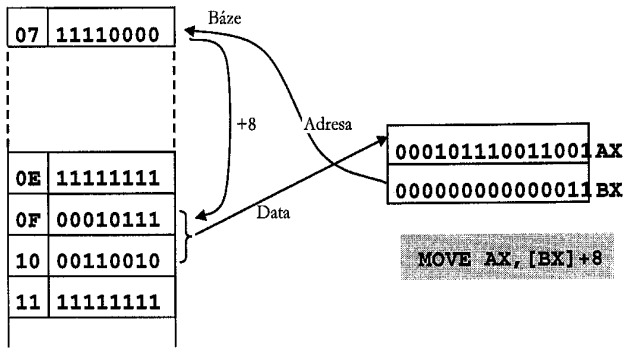
Při použití relativního adresování je hodnota operandu v instrukci přičtena k nějaké základní adrese obsažené v registru, který je explicitně deklarován instrukcí, nebo může být chápán implicitně. Rozeznáváme dva základní způsoby relativního adresování, a to **relativní adresování vůči bázové adrese** (obr. 43) a **relativní adresování vůči programovému čítači** (obr. 44). V prvém případě je hodnota efektivní adresy získána jako součet hodnoty uložené v bázovém registru a hodnoty operandu. Ve druhém případě je registrem účastnícím se této aritmetické operace programový čítač.

Specifickým případem tohoto relativního adresování je případ, kdy **hodnota je uložena přímo do pole operandu** - literál. Tuto variantu si můžeme rovněž představit tak, že se jedná o relativní adresování vůči programovému čítači, kde hodnota přičítaná k programovému čítači je 0, tedy adresa operandu bezprostředně následuje za operačním kódem (obr. 45).

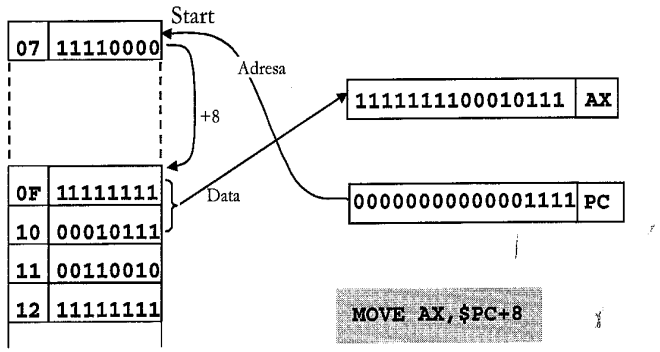
[9] Snížení nebo zvýšení záleží na organizaci zásobníku.



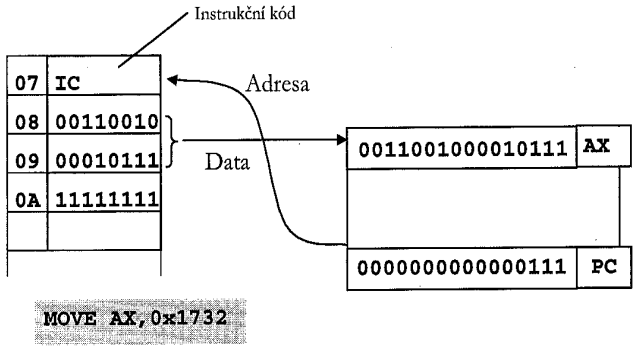
Obr. 42 Varianty nepřímého adresování registrem



Obr. 43 Relativní adresování vzhledem k bázové adrese



Obr. 44 Relativní adresování vzhledem k programovému čítači



Obr. 45 Relativní adresování s hodnotou v operandu

5.5.4 Indexované adresování

Při použití indexovaného adresování modifikujeme adresu položky v hlavní paměti obsahem **indexového registru**. Obvykle se indexové adresování kombinuje ještě s dalšími adresovacími mody a podobně jako v případě inkrementace nebo dekrementace ukazatele lze i v tomto případě použít možnost preindexace nebo postindexace. Tak můžeme obdržet například

- přímé indexové adresování paměti
- nepřímé adresování registrem s preindexací
- nepřímé adresování registrem s postindexací
- nepřímé adresování obsahem paměti s preindexací
- nepřímé adresování obsahem paměti s postindexací
- indexované relativní adresování vůči bázovému registru.

Poznamenejme, že jednotlivé kombinace indexací a nepřímého adresování mohou specifikovat, že indexace může být provedena předtím a nebo potom, co je vyzvednuta nepřímá adresa.

Příklad

Na níže uvedeném obrázku jsou uvedeny příklady některých možností adresace:

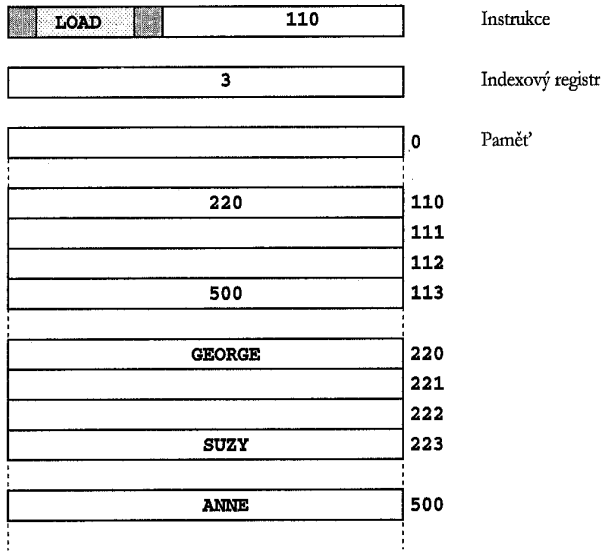
Přímé adresování bez indexace: Hodnota v operandu instrukce (**110**) je použita jako efektivní adresa (**110**) datové položky v paměti. To znamená, že při zpracování instrukce bude použita hodnota **220**.

Přímé adresování s indexací: Hodnota v operandu instrukce (**110**) je modifikována obsahem indexového registru (**3**) a výsledná hodnota (**113**) je použita jako efektivní adresa dat v paměti. Při zpracování instrukce bude použita hodnota **500** ležící na adrese **113**.

Nepřímé adresování bez indexace: Hodnota v operandu instrukce (**110**) je použita jako adresa v paměti, jejíž obsah (**220**) je efektivní adresou položky dat v paměti. V tomto případě bude datová položka **GEORGE** použita při zpracování instrukce.

Nepřímá adresace preindexování: Hodnota v operandu instrukce (**110**) je modifikována obsahem indexového registru (**3**) a je použita jako adresa v paměti (**113**), jejíž obsah (**500**) bude efektivní adresou položky dat v paměti. Pro zpracování instrukce bude použita hodnota datové položky **ANNE**.

Nepřímé adresování postindexace: Hodnota v operandu instrukce (**110**) je použita jako adresa v paměti, jejíž obsah **220** je modifikován obsahem indexového registru (**3**) a výsledek je použit jako efektivní adresa položky dat v paměti. V tomto případě bude hodnota **SUZY** použita při zpracování instrukce.



v nalezení jeho řešitelů.

A. Bloch: "Murphy's Law"

6 Procesory

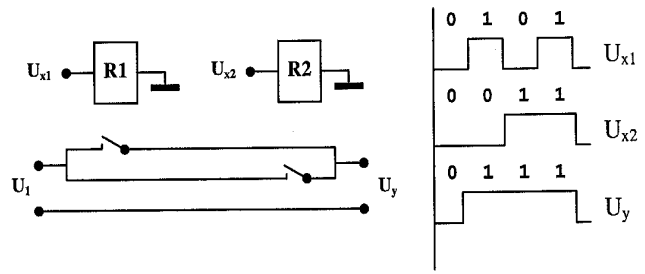
Architektura, organizace a implementace jednotlivých částí počítače, jsou disciplíny mezi sebou vzájemně velmi úzce provázané. Navržený druh a typ instrukcí a dat, zvolené šířky datových cest a další parametry musí být posléze realizovány v organizaci počítače a ta musí být implementována do příslušných elektronických prvků. Základem této implementace je vytvoření struktury, která bude poskytovat dostatečný základ pro realizaci složitějších postupů elementárními změnami obsahu prvků v této struktuře. Jinými slovy, budeme muset navrhnout v rámci organizace a implementace takovou strukturu, která nám umožní získat řízení nad touto strukturou, a to ve zvolených bodech a ve zvolených časových okamžicích. K tomu aby mohlo dojít k řízení procesoru je nutno vytvořit prostředí pro

- změnu stavu stroje v libovolném kroku
- uvolnění nebo znemožnění postupu dat na datových cestách

Toto řízení se obvykle provádí prostřednictvím **instrukcí**, tedy příkazů pro vnitřní struktury počítače ovlivňující jeho stav. K tomu, abychom instrukce mohli použít, musíme opět navrhnout takovou techniku, která umožní dekódovat informaci uloženou v operačním kódu instrukce, a tak realizovat jednotlivé kroky vyžadované k vykonání operačního kódu, včetně časové posloupnosti těchto datových kroků a otvírání či zavírání datových cest. Při konstrukci potřebného zařízení v elektronické oblasti se na nejnižší úrovni dostaneme na pevně vázanou logiku uzavřenou v elektronických obvodech, které jsou mezi sebou vzájemně propojeny a vytváří logické prvky, umožňující řešit výše uvedené požadavky na řízení. Základní **logické prvky**, které jsou použity k implementaci řídících funkcí, se rozdělují do dvou elementárních skupin - tzv. **kombinační logika** a **sekvenční logika**. Prvky v každé skupině charakterizujeme jejich

- **logickými vlastnostmi**, které jsou dány logickými funkcemi realizovanými prvkem a
- **fyzikálními vlastnostmi**, které umožňují hodnotit kvalitu provedení logické operace měřením fyzikálních veličin realizace. Tyto vlastnosti nazýváme parametry logického prvku a rozlišujeme mezi statickými parametry (např. počet vstupů prvku, maximální zátěž výstupu prvku v násobcích jednotkové zátěže, mezní hodnoty napěťových úrovní, šumová imunita atd.) a dynamickými parametry (zpoždění signálu mezi vstupem a výstupem a další kmitočtové vlastnosti)

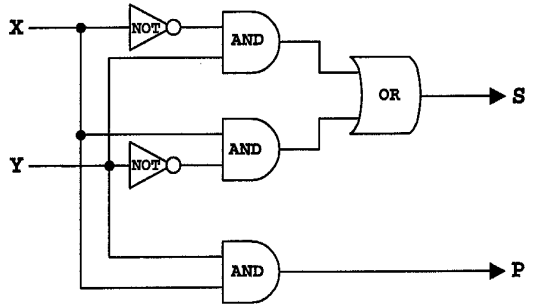
Kombinační logika je chápána jako obvodové řešení logické funkce a známe je pod rozšířeným názvem **logická hradla**. Tato logická hradla implementují např. funkce logické negace, logického součtu, logického součinu nebo i složitější logické funkce. Nejjednodušším logickým obvodem používaným ve výpočetní technice bylo relé, jehož kontakty spojeny do série realizovaly funkci logického součinu a spojeny paralelně vytvářely logický součet. Na obr. 46 je uveden příklad zapojení logického součtu vytvořeného z kontaktů relé a hodnoty napětí na výstupu U_y v závislosti na vstupních hodnotách U_{x1} a U_{x2}. Postupně s rozvojem elektroniky byly kombinační logické prvky vytvářeny z elektronek, diskrétních polovodičových součástek až po dnešní integrované obvody.



Obr. 46 Logický součet realizovaný kontakty relé

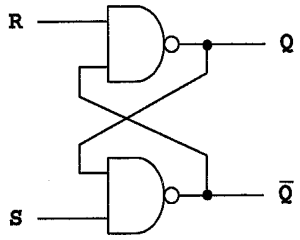
Pro výstavbu počítačů se sortiment logických prvků stabilizoval na sadě univerzálních logických členů, jejich sestavováním se realizují složitější logické funkce. Příkladem složení obvodu poloscítačky, tedy funkce součtu

vou bitů s přenosem, je uvedena na obr. 47. Pro sestavení poloscitacky jsou pouzity pouze elementarni logicke inkce **AND**, **OR**. a **NOT**[1].



X	Y	P	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Obr. 47 Realizace poloscítacího obvodu z logických hradel



t_n		t_{n+1}
S	R	Q_{n+1}
0	0	
1	0	0
0	1	1
1	1	Q_n

Tabulka přechodů

Obr. 48 Realizace klopného obvodu RS z kombinačních hradel NAND

Na rozdíl od kombinační logiky, jejíž výstup je závislý pouze na kombinaci vstupů, u sekvenčních obvodů je ýstup závislý nejenom na stavu vstupů, ale i na vnitřním stavu tohoto obvodu kterého nabyl v předcházejícím asovém okamžiku. Jednotlivé obvody v sekvenční logice, většinou nazývané **klopnými obvody**, se dělí podle oho, jakým způsobem může být jejich vnitřní stav ovládán. Zachování vnitřního stavu je docíleno vnitřními pětnými vazbami, které propojují vstupní a výstupní svorky tvořících kombinačních logických obvodů. Nejznámějším klopným obvodem je typ **RS**, jehož výstupní hodnota se mění v závislosti na příchozích odnotách na svorkách **R**, **S** a vzhledem ke zpětné vazbě i na stavu obvodu. Realizace klopného obvodu "RS" logických hradel je na obr. 48. Jak je vidět, ne všechny kombinace vstupních proměnných jsou povoleny a edou na definovaný stav výstupů[2]. Proto se používají složitější logické struktury, které se označují jako klopné bvody typu "D", popřípadě "JK" a další, jejichž výstupní stav závisí jak na hodnotě vstupních svorek, tak i na ynchronizačním, tzv. hodinovém pulsu. Např. klopný obvod typu "D" změní svůj vnitřní stav, tedy uloží ožadovanou informaci, při náběžné hraně zapisovacího impulsu, zatímco klopný obvod typu "JK" učiní totéž ž při závěrné hraně zapisovacího pulsu. Velmi často se vyskytuje klopný obvod, který má průchozí charakter, to namená, že při změně svého hodinového vstupu z jednoho logického stavu do druhého buď začne propouštět

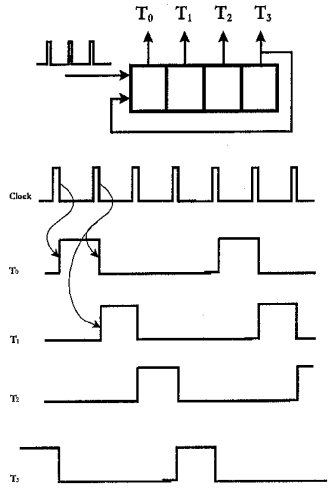
Pro větší názornost jsou ve schématech používány symboly hradel z původní americké normy.

Asynchronní sekvenční obvod RS, využívá v praxi rozdílného parazitního zpoždění v kombinačních členech. To umožňuje dlišit příští vnitřní stav od současného vnitřního stavu.

logické stavy na svém vstupu na svůj výstup, nebo poslední logický stav svého vstupu, který byl v okamžiku přechodu hodin uchován na svém výstupu. Tento obvod se nazývá **latch**[1].

Z těchto jednoduchých logických obvodů se skládají složitější obvody, které vykonávají elementární funkce při řízení procesoru. Mezi složitější sestavy kombinačních obvodů patří např.

- **kodéry a dekodéry**, které mění kombinaci logických vstupů na odpovídající kombinaci logických výstupů podle předem stanoveného předpisu (např. dekodér z tříbitového binárního záznamu na kód 1 z 8)
- **multiplexery** a **demultiplexery**, které umožňují přenést hodnotu logického vstupu na vybraný logický výstup nebo opačně, které na svém výstupu zobrazí hodnotu na vybraném logickém vstupu.
- **úplné sčítačky**, vícebitové **polosečítačky** nebo **generátory přenosů**, které se stávají součástí aritmeticko-logických jednotek a patří už mezi velmi složité sestavy kombinačních obvodů.



Obr. 49 Generování impulsů pro časování procesoru

Nejznámějším složitějším obvodem, který je sestaven ze sekvenčních obvodů, je **registr**. Ten slouží k uchování informace a mění svůj výstup v závislosti na hodnotě vstupu a příchodu hodinového impulsu, který hodnotu na logických vstupech do registru zapíše. Speciálním typem registru je **posuvný registr**, který umožňuje posouvat jednotlivé bity uložené v klopných obvodech registru směrem k bitu s nižší nebo vyšší váhou, a tak měnit svůj obsah bez toho, aby došlo ke změně jeho vstupů (tedy pouze pod řízením hodinového impulsu). Jinými, velmi často používanými sestavami klopných obvodů, jsou **čítače**, které čítají došlé hodinové impulsy a vydávají na svých výstupech hodnotu odpovídající načtenému počtu těchto impulsů. Výstupní hodnota může být ještě různým způsobem kódována, takže můžeme se setkat s čítači binárními, s čítači v kódu BCD nebo jinými speciálními čítači. I tyto relativně jednoduché sestavy klopných obvodů mohou nabývat různých vnitřních stavů a obvykle se přechody mezi těmito stavy reprezentují orientovanými grafy.

6.1 Řadič

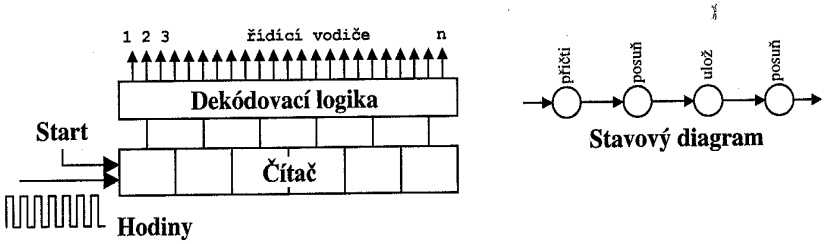
Každý procesor, který je sestaven z výše vyjmenovaných prvků, bude pro svoji práci potřebovat slec časových impulsů, který bude určovat, v jakých okamžicích budou aktivní jednotlivé části procesoru. Vstupem pro generování sledu impulsů budou tzv. **hodiny**, základní kmitočet vyráběný oscilátorem, které se zpracují tak aby výsledkem byly řídicí impulsy pro jednotlivé obvody procesoru. Ty budou přicházet v předem definovaných časových okamžicích, a tak umožňovat souslednost operací mezi jednotlivými bloky procesoru. Příkladem časovacího obvodu procesoru může být použití posuvného registru, který na svém výstupu vydá čtyři fázově posunuté impulsy, a tak bude možno zajistit časovou následnost čtyř po sobě jdoucích akcí (viz obr. 49) Propojením časových impulsů s dekódovací logikou vznikne jednotka, schopná řídit posloupnost dějů vyžadovaných k provedení úlohy – **řadič**. Pro jedinou úlohu lze navrhnout několik typů řadičů, jejichž funkce se bude lišit podle realizovaného algoritmu. Podle typu úlohy a předpokládaných dějů probíhajících při jejím zpracování můžeme rozlišit dva typy řadičů – direktivní a zpětnovazební.

Direktivní řadič působí vždy stejně a nezávisle na průběhu řízeného děje. Takovým způsobem řízení jsou vybaveny jednoúčelové jednotky, jejichž ovládání spočívá na sledu neměnné posloupnosti impulsů generovaných na řídících vodičích, např. jednotka provádějící elementární násobení dvěma bude vždy řízena stejným sledem impulsů, které zajistí posun obsahu použitého registru o jeden bit doleva., nebo řadič světelné signalizace na

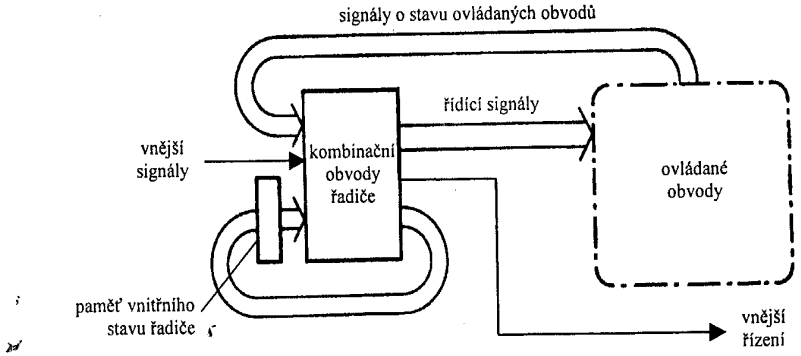
[1] Český název pro "latch" je záchytná paměť, ale i nadále budeme dávat v nejednoznačných ne mnohoslovných názvech přednost převzatému odbornému termínu.

žovatce. Neměnný impulsní sled může generovat autonomní sekvenční obvod, tedy soustava jednoduchých
:venčních obvodů propojená tak, že má pouze jeden "spouštěcí" vstup. Stavový diagram - graf přechodů mezi
vy direktivního řadiče bude vždy lineární řetězec (obr. 50).

Zpětnovazební řadič je obecnější formou uspořádání než řadič direktivní. Od předchozího typu se liší
ména v počtu vstupních veličin, neboť na jeho vstup mohou přicházet signály nejrůznějšího druhu o stavu
ládaných prostředků., např. stav některého bitu v registru, přenos při sčítání, přeplnění, výskyt chyby, žádost
sluhy atd. Posloupnost řídících signálů je vzhledem k možnosti větvení, na základě vstupních signálů, závislá
ději ovládaných obvodech. Zpětnovazební řadič musí být realizován jako sekvenční obvod na jehož vstupy
u přivedeny signály z ovládaných obvodů. Obecné schéma řadiče je na obr. 51. Do uvedené struktury spadá
ioho variant, které se vzájemně liší rozčleněním paměti vnitřního stavu a rozčleněním kombinačních částí,
př. způsobem její realizace.



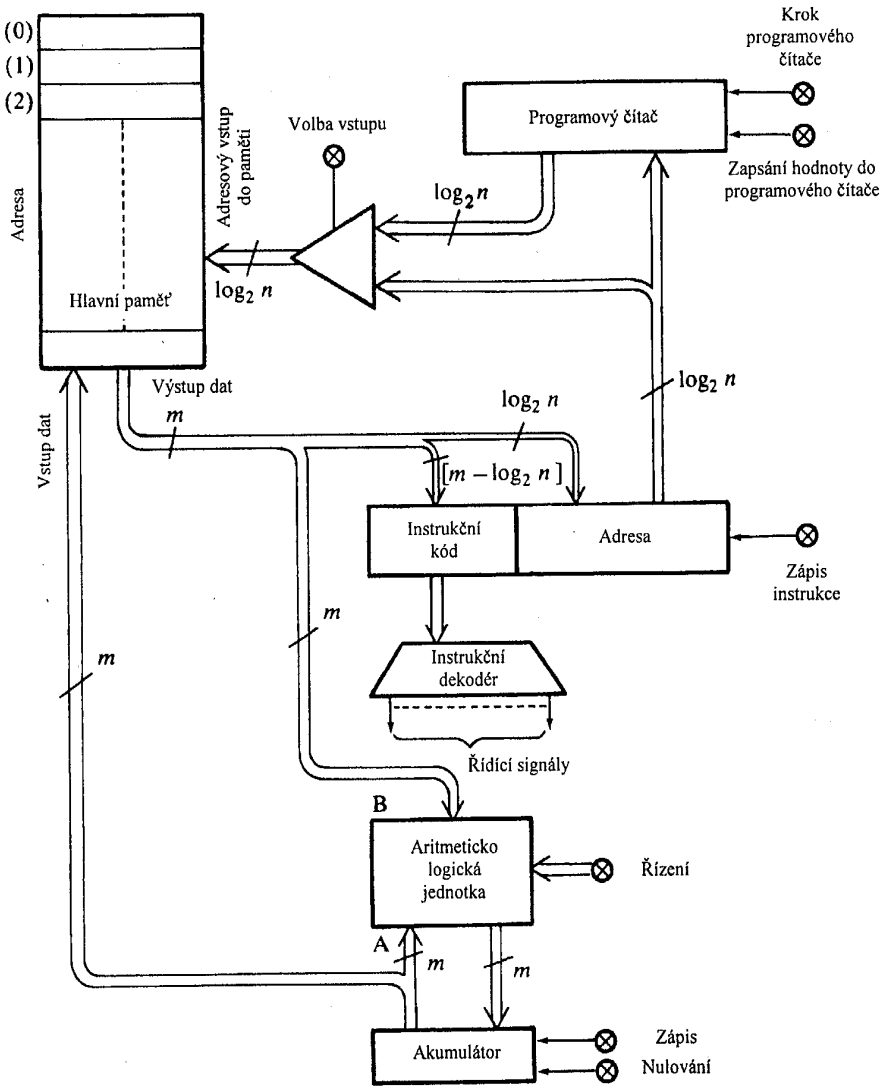
Obr. 50 Direktivní řadič a jeho stavový diagram



Obr. 51 Obecné schéma zpětnovazebního řadiče

Řadič, řízený hodinovým kmitočtem, je ve smyslu klasického modelu John von Neumanna jednou
rozhodujících částí počítače a jeho struktura, mnohdy velmi modifikovaná, zůstává neoddělitelnou součástí
:ech stavových strojů. Úlohu řadiče při řízení procesoru můžeme osvětlit na schématu jednoduchého
:ocesoru, který bude zpracovávat instrukci obsahující pouze operační kód a adresu (viz obr. 52). Abychom
:ohli instrukci uloženou v paměti provést, musíme obsah programového čítače, jenž ukazuje na adresu
isledující instrukce, použít pro adresování v oblasti paměti, ve které je tato instrukce uložena. Po adresování
ıměti se obsah příslušné buňky paměti objeví na výstupu paměti a bude uložen pro další zpracování do registru
strukce, který uchová instrukci po dobu jejího zpracování. Následovně může být operační paměť uvolněna pro
ılší použití. Část obsahu instrukčního registru odpovídající operačnímu kódu instrukce bude předložena
:kodéru, který na základě hodnoty operačního kódu vytvoří ve spolupráci s řadičem posloupnost řídicích
gických signálů, jejichž druh a počet budou vždy závislé na druhu zpracovávané instrukce. Na základě tohoto
peračního kódu budou pod řízením řadiče vykonávány všechny další aktivity procesoru. Druhá část instrukce
bsahuje adresu a v případě, že se jedná o aritmeticko-logickou instrukci, pak tento údaj bude ukazatelem na
dresu v paměti, kde se nachází operand. Přístup do hlavní paměti se zopakuje a výsledná data jsou předložena
: zpracování aritmeticko-logické jednotce.

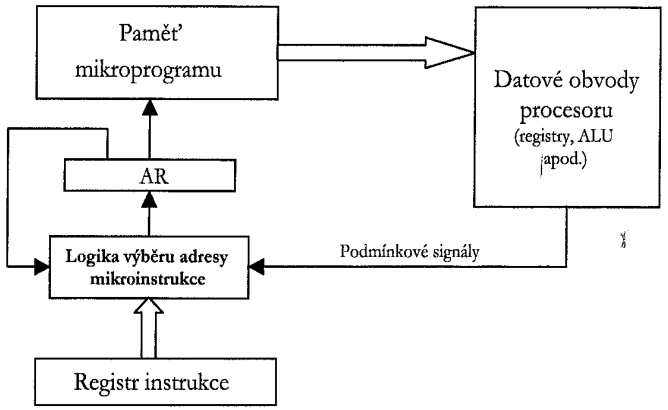
Výstupem práce aritmeticko-logické jednotky je hodnota, která může být uložena do akumulátoru
Současně hodnota programového čítače může být zvětšena o jedničku. Při vykonávání elementární operace
podle předcházejícího popisu bylo v činnosti několik součástí základních obvodů procesoru. Tyto součásti mus
být postupně synchronizovány tak, aby nedošlo ke vzájemnému blokování nebo ovlivňování dat, která používaj
stejné přenosové cesty. Řízení a synchronizace práce jednotlivých částí byla přitom vykonávána časovacím
obvody procesoru - řadičem, který vydává řídicí signály pro jednotlivé bloky procesoru v potřebné posloupnosti



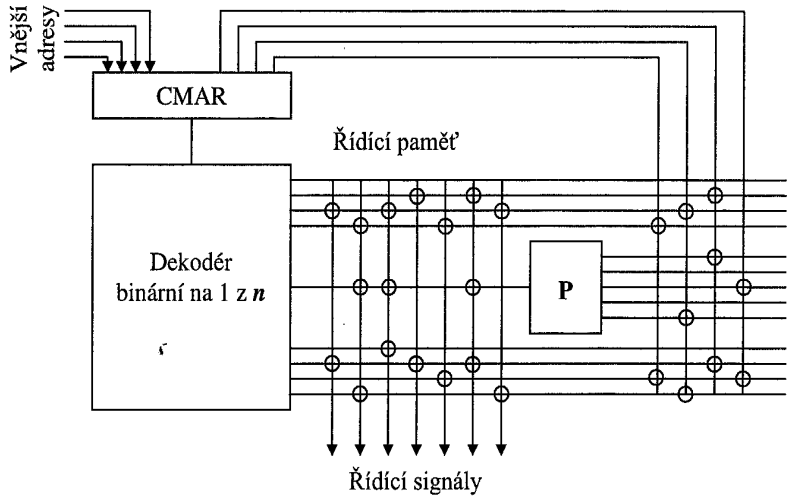
Obr. 52 Jednoadresový procesor

Složitost a náročnost pevné logiky, která tvořila jádro řadiče procesoru, velmi omezovala jeho flexibilitu
zvyšovala cenu návrhu. Bylo tedy snahou konstruktérů snížit složitost prováděných operací, a tím i složitost říd
logiky procesoru. To vedlo k technikám implementace, ve kterých levnější, jednodušší a rychlejší **hostitelsl**
počítač je řízen zvláštním programem hluboko pod úrovní asembleru tak, že se programátorovi jeví jako **cílo**
počítač poskytující všechny očekávané vlastnosti. Každá instrukce vykonávaná na hostitelském počítači využí
plně zdrojů hardware, zatímco každá instrukce na cílovém počítači spustí odpovídající instrukční sekvenci

osuteiskem pocitaci. Jestuze nosuteisky pocitac je implementovan tak, ze hardware obsahuje pouze jednoduche instrukce pro obsluzeni datovych cest registrů paměti a vstupně-výstupních zařízení, můžeme používat termíny ikroprogramování a emulace.



Obr. 53 Obecné schéma mikroprogramového řadiče



Obr. 54 Wilkinsovo schéma mikrořadiče

Mikroprogramování můžeme popsat jako systematický postup, kdy se za pomoci překladače ikroprogramů napsaných v mikroasembleru vytvoří binární obsah řídicí paměti, která je základem ikroprogramovaného řídicího obvodu. Převažující část problému syntézy řídicích obvodů procesoru se tak esune na programovou úroveň, která disponuje daleko komfortnějšími prostředky než vlastní obvodový ávrh. Zároveň je umožněna daleko větší flexibilita řízení stroje a zjednodušuje se postup při opravě chyb.

Základem mikroprogramového řadiče je paměť mikroprogramů, která je řízena adresovým registrem ikroinstrukce (viz obr. 53). Začátek mikroprogramu, který zajišťuje provedení určité instrukce, se určí pomocí giky výběru adresy z kódu operace uloženého v instrukčním registru cílového počítače. Jednotlivé ikroinstrukce se potom čtou z paměti mikroprogramů podle obsahu adresového registru mikroinstrukce, a to ať sekvenčně, kdy se adresa příští mikroinstrukce určí inkrementací současné adresy mikroinstrukce, nebo ímo, kdy součástí pole mikroinstrukce je adresa příští mikroinstrukce. Principiální schéma mikroprogramového řadiče definoval již M. Wilkes v r. 1951 a je uvedeno na obr. 54. Registr **CMAR** v tomto obrázku je adresov registr paměti mikroinstrukcí, přepínač **P** je řízen vnější podmínkou a určuje adresu příští mikroinstruk současně s dalšími signály, které poskytují vnější zdroj této adresy. Tento vnější zdroj adresy je potom generová z logiky pro výběr adresy, což je vlastně dekodér instrukce cílového počítače. Obsah paměti mikroprogramů obvykle nazývá **firmware**.

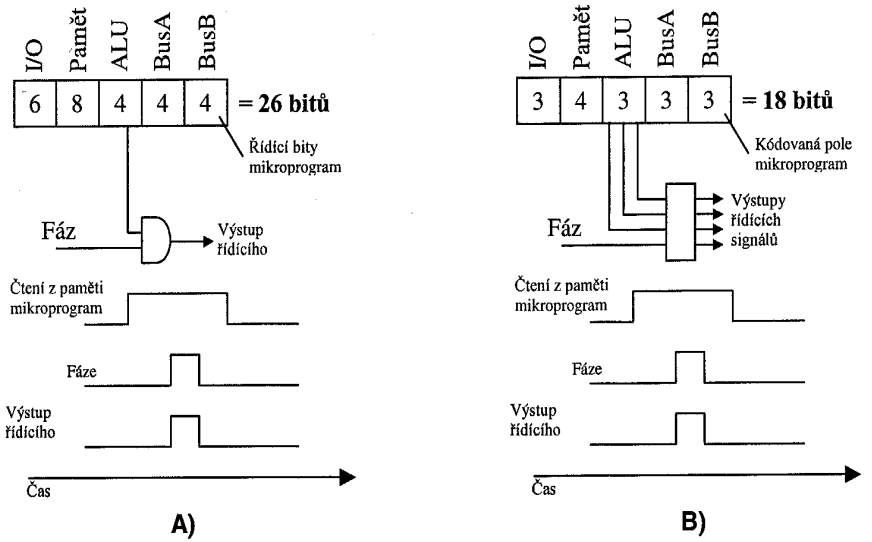
Vysoká flexibilita tohoto přístupu vedla i ke konstrukcím, kde mikroprogramy byly uloženy d zapisovatelné paměti a byla doplněna cesta, kterak do této paměti zapsat. Např. systém 370 firmy IBM nejdřív zaváděl z vnějšího zařízení do této paměti sadu diagnostických mikroprogramů, které provedly podrobnc diagnostiku systému, a teprve po úspěšném startu se zapsaly do řídicí paměti mikroprogramy pro standard instrukční soubor.

6.2 Formáty mikroinstrukcí

Základní dvě funkce mikroinstrukčního řadiče spočívají v řízení mikrooperací a v krokování slec mikroinstrukcí, tedy posloupnosti jejich provádění. Tyto dvě funkce musí být obsaženy ve všec mikroinstrukcích. K tomu, aby bylo možno mikroinstrukční program vůbec provést, je nutno, ab mikroinstrukce měla definován svůj formát. Tento formát obvykle obsahuje pole, jež reprezentují

- vstupní operandy
- operace aritmeticko-logické jednotky
- cílovou adresu pro výsledek operace
- adresu příští mikroinstrukce

Součásti struktury mikroinstrukce může být i bit příznaku skoku, podmínkový bit a další. Vlastní provádě mikroinstrukcí je potom časově řízeno synchronizačními signály, jejichž posloupnost se označuje jako fáz Obecně můžeme definovat tři různé formáty mikroinstrukcí podle toho, jakým způsobem jsou v mikroinstruk uloženy řídicí bity.



Obr. 55 Obsahy pamětí mikroinstrukce při horizontálním (A) a vertikálním (B) mikroprogramování

Dlouhý formát mikroinstrukce, který obsahuje přímo hodnoty řídicích signálů, které se dále nedekódují, nazývá **horizontální formát**, nebo **horizontální mikroprogramování**. V tomto případě vzrůstá nárok kapacitu paměti mikroprogramu, avšak na druhou stranu vybavování řídicích signálů probíhá největší rychlos

ıledem k tomu, že je umožněna jakákoliv kombinace řídicích signálů, je tento formát i nejpružnější. V ɔadě, kdy budou existovat skupiny řídicích bitů takové, že v jednom okamžiku nemohou současně existovat řídicí signály, můžeme tyto skupiny zakódovat do společného pole a hodnotu uloženou v tomto poli měnit na potřebný výstupní kód pomocí dekodéru ***1 z n***. Při kódování do polí dochází ke značné úspoře emu řídicí paměti, avšak dekódování obsahu pole přidává časové zpoždění při realizaci mikroinstrukce. ıledem k tomu, že již v počátku návrhu je nutno rozhodnout o sdružení jednotlivých bitů do polí a odstranit některé kombinace, existuje zde menší flexibilita celého návrhu a může dojít k tomu, že v budoucnu nebude žno provést některé modifikace.

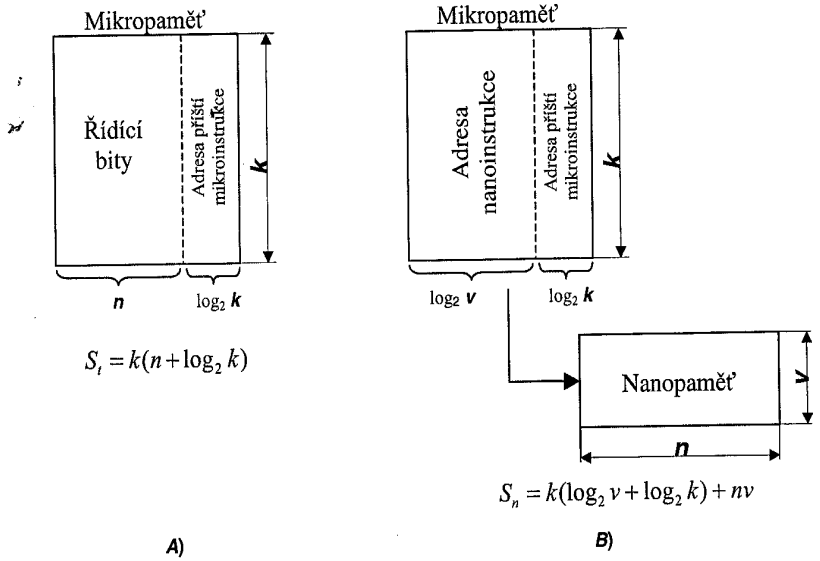
Aplikací metody kódování na celé pole řídicích bitů obdržíme tzv. vertikální formát mikroinstrůkce, neboli **tikální programování**, které je maximálně úsporné. Podobně jako u kódování do polí dochází k datečnému zpoždění při dekódování obsahu řídicí paměti. A vzhledem ke složité analýze současnosti řídicích nálů bude navržený dekodér velmi omezený a nebude dovolovat další rozšiřování.

Poznámka

Při porovnáním délky polí v obr. 55 je zřejmé, že pro kódování např. 4 bitů v horizontální mikroinstrukci je třeba 3 bitů ve vertikální mikroinstrukci. Důvodem je nutnost existence stavu, kdy žádný z dekódovaných a použitých signálů nebude aktivní.

3 Nanoprogramování

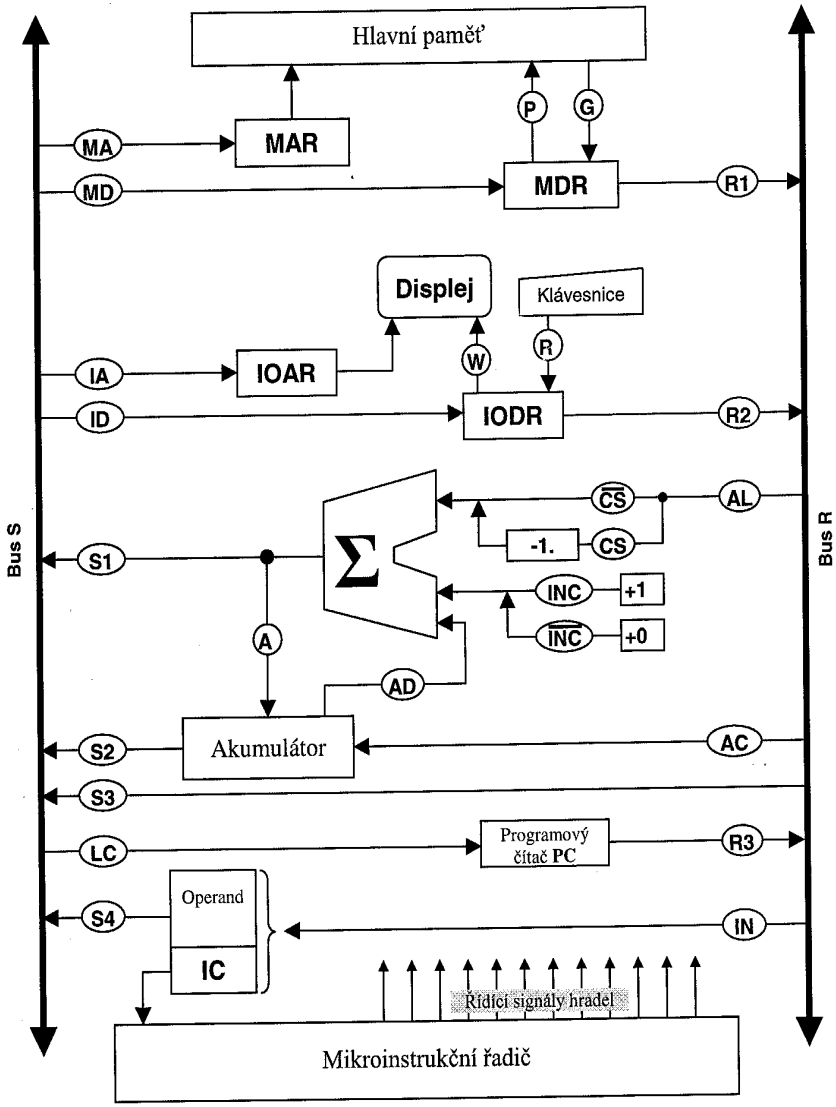
U mikroprogramových počítačů se každá mikroinstrukce používá k řízení okolního hardwaru, a to ať už mo nebo až po dekódování. Vzhledem k tomu, že se vyskytuje pouze konečné množství různých kombinací, rých řídicí bity mohou nabývat a toto množství je relativně malé, bude výhodné použít zvláštní paměť, tzv. **nopaměť**, ve které budou uloženy bity pro řídicí signály, a tuto paměť adresovat pomocí zvláštního pole v kroinstrukci. Tento dvouúrovňový mikropaměťový systém se nazývá **nanoprogramování** a je použit např. u kroprocesoru Motorola řady 68000. Použitím nanoprogramování dojde k výraznému snížení nároků na likost paměti mikroprogramu a jejich porovnání je uvedeno na obr. 56, kde S_t značí potřebný objem paměti při kroprogramování a S_n je docílený objem paměti při použití nanoprogramování.



Obr. 56 Velikost paměti mikroprogramu (A) a nanoprogramu (B)

6.4 Virtuální procesor

Operace mikroprogramovaného nebo nanoprogramovaného procesoru si můžeme ozřejmit na obr. 57, který znázorňuje virtuální procesor, používající dvě sběrnice **Bus S** a **Bus R**, pomocí kterých jsou propojeny jednotlivé bloky procesoru. Hradla, připojující tyto bloky na sběrnice, jsou ovládána řídicími signály z obvodu mikroprogramového řízení. Vstupem obvodu mikroprogramového řízení je operační kód instrukce cílového procesoru, na jehož základě jednotka mikroprogramového řadiče generuje posloupnost řídicích impulsů otevírajících hradla, která připojují funkční bloky procesoru ke sběrnicím.



Obr. 57 Organizace virtuálního procesoru

Pro přístup do hlavní paměti virtuálního procesoru slouží dva registry **MAR** (Memory Address Register) **MDR** (Memory Data Register), kde první z nich je použit pro uložení adresy, na kterou chceme v hlavní pamě přistoupit, a druhý slouží pro uložení čtených nebo zapisovaných dat. Hlavní paměť vykoná příslušnou opera

v okamžiku, kdy je zapsána adresa do **MAR**. To znamená, že data se objeví při operaci čtení v **MDR** nebo naopak při operaci psaní se zapíší z **MDR** do hlavní paměti. Podobně jako při přístupu do hlavní paměti jsou registry **IOAR** a **IODR** používány k manipulaci s výstupním a vstupním kanálem procesoru. Aritmeticko-logické operace jsou prováděny pomocí sčítačky a akumulátoru s doplněním jednoduchých obvodů umožňujících změnu znaménka při čtení jedničky a negaci. Pro řízení práce cílového procesoru slouží programový čítač.

Uveďme příklad provedení instrukce **LOAD** - zavedení obsahu paměti do akumulátoru na tomto virtuálním procesoru. Předpokládejme, že programový čítač ukazuje na místo v paměti, kde instrukce **LOAD** je uložena. První část mikroprogramu musí zajistit vyzdvižení této instrukce z hlavní paměti a bude obsahovat následující posloupnost otevírání hradel (znakem PC je označen programový čítač)

1. `[PC] → R3, S3, MA → [MAR], G → [MDR]`

2. `R1, IN → [instrukční registr]`

3. `[PC] → R3, S3, MD → [MDR]`[1]

4. `R1, $\overline{AL}$, CS, INC →` *<sumátor>* `→ S1, LC → [PC]`

V této části mikroprogramu dojde k vyzdvižení obsahu hlavní paměti, kde se nachází prováděná instrukce, zavedení tohoto obsahu do instrukčního registru a zvýšení programového čítače o 1 tak, že bude ukazovat na příští instrukci uloženou v hlavní paměti. Spínání jednotlivých hradel je nutno navrhnout tak, aby nedošlo ke konfliktu mezi jednotlivými sběrnicemi.

Zavedením instrukce **LOAD** do instrukčního registru došlo k dekódování operačního kódu této instrukce a spustil se mikroprogram, který zajistí její provedení. Ten musí realizovat následující kroky, které mohou být provedeny v jednom cyklu:

`[instrukční registr] → S4, MA → [MAR], G →`

`→ [MDR], R1, AC → [Akumulátor]`

V mikroprogramovaném stroji se instrukce hlavního procesoru (cílového procesoru) rozpadla do dvou částí - části vyzdvižení instrukce - **FETCH** a části provedení instrukce **EXECUTE**, které obě jsou realizovány pomocí sledu mikroinstrukcí. Část **EXECUTE** může být dále rozdělena na **dekódování instrukce** a na její vlastní provedení, pokud i dekódování bude realizováno posloupností mikroinstrukcí[2]. Tuto trojici neustále se opakujících operací nazýváme **cyklem instrukce**. V první fázi každého cyklu procesor vyzvedne instrukci z paměti. Adresa instrukce, která má být vyzdvižena, je uložena ve speciálním vnitřním registru, jak již bylo uvedeno výše, který se nazývá **programový čítač**. Obsah tohoto programového čítače se po vyzdvižení každé instrukce zvyšuje o délku zpracovávané instrukce. To znamená, že na počátku nové fáze vyzdvižení instrukce bude programový čítač ukazovat na následující instrukci v pořadí, pokud nebyl programový čítač modifikován v pozdějších fázích instrukčního cyklu k čemuž dochází při instrukcích skoku. Ve fázi provádějící dekódování instrukce procesor uloží instrukci přečtenou z paměti do jiného vnitřního registru, který se nazývá **instrukční registr**. Tento instrukční registr vždy obsahuje jednu strojovou instrukci, která je obvykle kódována jako binární číslo. Zvláštní obvody procesoru dekódují hodnotu v instrukčním registru, a tak získají pokyny pro provádění dalších fází instrukčního cyklu.

V exekuční fázi, tedy fázi, kdy procesor skutečně vykonává příkazy, které mu jsou v instrukčním kódu předány, jsou obvykle procesorem požadovány další operace na okolních subsystémech, např. instrukce může potřebovat další operandy z paměti nebo naopak bude chtít uložit výsledek operace do některé oblasti paměti apod. Na konci této fáze stroj spustí nový cyklus, který opět začíná fází "vyzdvižení následující instrukce". Jak bylo uvedeno v kap. 5.3, instrukce mohou být klasifikovány do různých skupin, nejčastější jsou však tři hlavní typy instrukcí - aritmeticko-logické, instrukce přesunu dat a řídicí instrukce.

---

[1] Zavedením hodnoty PC do registru MD v tomto kroku se vyvarujeme toho, aby při vyzdvižení instrukce byl přepsán akumulátor.

[2] Tento případ je velmi častý např. u mikroprocesorů RISC, realizujících úplnou sadu instrukcí. Některé složitější instrukce jsou nahrazovány posloupností mikroinstrukcí během dekódovacího cyklu.

Časování každé fáze cyklu velmi závisí na vnitřní konstrukci procesoru a složitosti instrukce, která je prováděna. V technickém popisu procesoru se obvykle dozvíme, kolik základních časových jednotek, tz. hodinových cyklů, je potřeba na příslušnou instrukci. Avšak nemůžeme očekávat, že jednoduchým vynásobení délky instrukcí a jejich počtu obdržíme skutečnou dobu trvání programu. Skutečné časování závisí dále ještě na organizaci paměti komunikačních kanálů, způsobu připojení procesoru k paměti a dalších vlastnostech organizace procesoru.

6.5 Optimalizace práce procesoru

Rozdělení procesu zpracování instrukce do několika postupných kroků nabízí možnost zpracování každého kroku ve zvláštní jednotce, která bude provázána s jednotkou následující. Tyto jednotky si výsledek svého zpracování budou předávat podobně jako montážní linka ve výrobním závodě. Tento způsob zpracování se nazývá **proudovým zpracováním** – "pipeline processing". Další práce na implementaci proudového zpracování vedly k návrhu **vektorových** a **superskalárních** procesorů, které využívají přirozeného paralelismu vyplývajícího ze zpracovávaných dat. Studiem závislosti mezi jednotlivými instrukcemi předkládanými ke zpracování procesoru byly nalezeny podmínky a metody pro přerovnání instrukcí a optimalizaci instrukcí skoků a další způsoby zrychlení práce procesorové jednotky.

6.5.1 Procesory s proudovým zpracováním

Výše uvedená analogie s výrobní linkou v průmyslové výrobě je běžně používaným příkladem při výkladu principu práce procesoru s proudovým zpracováním instrukcí. Jedním z důvodů zavedení výrobních linek byl zvýšení produktivity. To samozřejmě předpokládalo zvýšení počtu instrukcí, které se vykonávaly za jednotl času a tak se zvýšil i počet produktů vyrobených za jeden den. Tím, že se složité operace rozdělily na jednotlivé části bylo možno je provádět paralelně. Jednotliví dělníci vykonávali následné, vzájemně vázané kroky podél té výrobní linky tak, že když výrobek absolvoval jednu fázi na výrobní lince, byl předán podél této linky k provedení následujícího kroku.

Proudové zpracování, nazývané také **pipelining**, je použito ve dvou hlavních oblastech návrhu počítače: zpracování instrukcí a aritmetické operace. Pro každý systém, který využívá proudového zpracování, existují základní požadavky

- systém může být navržen pro proudové zpracování tehdy, jestliže opakovaně provádí základní funkci.
- základní funkce musí být dělitelná do nezávislých kroků, které vyžadují minimální vzájemné překrytí.
- složitost jednotlivých kroků musí být přibližně stejná.

Počet stupňů, které jsou použity při proudovém zpracování, se označuje jako **hloubka proudové zpracování**. Pro snadnější představu jak proudové zpracování pracuje předpokládejme, že budeme provádět instrukci sčítající dvě čísla v pohyblivé řádové čárce. Jestliže obě čísla budou zapsána v běžném tvaru $m \cdot 2^{exp}$, pak můžeme rozdělit funkci sčítání na následující kroky, stupně proudového zpracování:

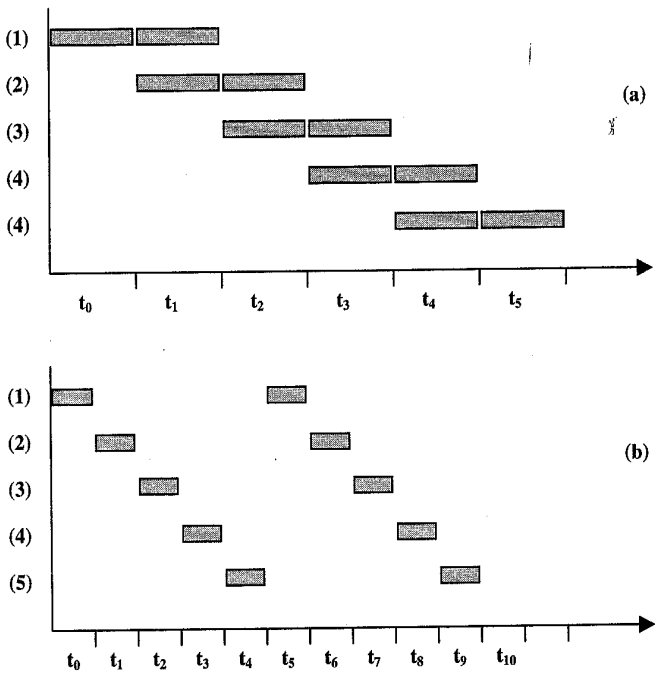
1. Jestliže $exp_2 < exp_1$, zaměň operandy. Nalezni rozdíl exponentů $exp_d = exp_2 - exp_1$.
2. Posuň m_2 doprava o exp_d bitů.
3. Vypočítej hodnotu mantisy sečtením m_1 a m_2. Exponent součtu má hodnotu exp_1.
4. Normalizuj součet.

Větší složitost sčítačky používajícího proudové zpracování se vyplatí tehdy, budeme-li v našem příkladě sčítat dlouhé sekvence čísel. Potom operace v každém stupni může být prováděna na různých párech vstupních hodnot. Např. jeden stupeň může porovnávat exponenty jednoho páru operandů, zatímco druhý stupeň sčítá mantisy odlišného páru operandů.

Velmi důležitým požadavkem pro práci jednotlivých stupňů je, že nenastane konflikt mezi požadavky a vstupy. To znamená, že jednotlivé operandy musí být nezávislé. Pro příklad uveďme program obsahující dvě instrukce:

```
R2 = R0 + R1
R4 = R3 + R1
```

oznamenejme, že obě instrukce označují registr **R1** jako jeden ze svých vstupů. Budeme-li podrobněji alyzovat postup práce při sčítání, zjistíme, že existuje potenciální konflikt ve stupních *2* a *3* naší teoretické ítací linky, protože stupeň *2* může vyžadovat posuv mantisy operandu **R1** ve stejnou dobu, kdy stupeň *3* bude žadovat, aby mantisa **R0** byla sečtena s mantisou **R1**. Jedním z řešení vzniklého konfliktu může být vytvoření pií operandů a pohyb těchto kopií podél linky proudového zpracování. Tak centrální procesorová jednotka edá sčítačce kopie registru **R0** a **R1** ve chvíli, kdy se začne zpracovávat první operace pro první pár. Druhý upeň proudového zpracování obdrží tyto kopie z prvního stupně současně s hodnotou exp_a.



Obr. 58 Gantovy diagramy pro hypotetickou sčítačku s proudovým zpracováním

Ve výše uvedeném příkladu můžeme nalézt i jiný potenciální problém proudového zpracování. Předpokládejme, že druhá instrukce se změní tak, že jeden z jejích operandů bude **R2**. To znamená, že nalyzovaný pár instrukcí bude modifikován následovně

```
R2 = R0 + R1
R4 = R2 + R1
```

Druhá instrukce tedy závisí na výsledku z předcházející instrukce. Centrální procesorová jednotka tedy iemůže vyslat druhý pár operandů na vstup proudové sčítačky do té doby, než první součet opustí její poslední tupeň. Tento druh interakcí při používání proudového zpracování vede k výraznému prodloužení zpracování rogramu a vytváří intervaly, kdy jednotlivé stupně proudového zpracování jsou prázdné. Těmto prázdným asovým úsekům říkáme často **bubliny**[1].

Na obr. 58 je ukázán záznam postupu operací při proudovém zpracování, známý jako **Gantův diagram**. Na horizontální ose je znázorněn čas, vertikálně jsou umístěny řádky, vždy jeden řádek pro jeden stupeň linky roudového zpracování. Úsečka v cyklu t_i znamená, že příslušný stupeň je aktivní právě v tomto cyklu. Pokud ide úsečka chybí, odpovídající stupeň aktivní není. Obrázek ilustruje funkci proudového zpracování při sčítání vou čísel v pohyblivé řádové čárce podle předcházejícího příkladu. Na prvním diagramu označeném *(a)* je ıveden případ, kdy dvě následující instrukce se mohou vzájemně překrývat. Diagram *(b)* znázorňuje případ, kdy

[1] Původní anglický termín "bubles".

druhá instrukce musí čekat do okamžiku, než se první instrukce zpracuje. Jak je vidět z obrázku, každá instrukcí cyklus t_i prvního stupně zpracovává druhou instrukci, zatímco druhý stupeň je aktivní při zpracování první instrukce. V každém následujícím cyklu těchto dvou instrukcí, které se pohybují podél linky proudového zpracování, se cykly posunují do dalšího stupně. Z uvedených příkladů vyplývá, že při nezávislých instrukcích budeme potřebovat ke zpracování dvojice součtů šest cyklů linky, zatímco při závislých součtech nebude výpočet dokončen dříve než po desátém cyklu. Zpoždění je zapříčiněno bublinou délky čtyř cyklů, která je projevem závislosti uvedených operací.

Obecně můžeme říci, že linka proudového zpracování instrukcí hloubky d může zpracovat n instrukcí v $n + d$ krocích, pokud se neobjeví bubliny. Bez použití proudového zpracování by každá aplikace základní funkce vyžadovala d cyklů, které by musely být prováděny sekvenčně a tedy celkový čas by byl $n \,.\, d$ cyklů. Zrychlení využitím proudového zpracování pak bude

$$\frac{n \,.\, d}{n + d}$$

Je-li počet instrukcí výrazně větší než hloubka linky proudového zpracování, $n >> d$, můžeme zanedbat hodnotu d ve jmenovateli, a tak zrychlení bude lineárně závislé na hloubky linky proudového zpracování.

Příklad

Abychom získali číselnou představu předpokládejme, že budeme sčítat tisíc párů čísel a na každý součet budeme potřebovat pět instrukčních cyklů. Pak stroj bez proudového zpracování bude zřejmě vyžadovat 5000 cyklů zatímco s naší teoretickou pětistupňovou linkou proudového zpracování můžeme očekávat, že správný součet získáme po 1000 + 5 cyklech. Zrychlení dosažené proudovým zpracováním je 4,97.

Existuje však mnoho možných příčin vzniku bublin v lince proudového zpracování. Nejčastěji jsou to **závislosti mezi následujícími instrukcemi**. Stejným problémem je i výskyt instrukce skoku nebo cyklu, který nazýváme **závislostí řízení**. Jestliže linka proudového zpracování instrukcí vyzvedne instrukci, která bude požadovat skok do jiné části programu, vznikne bublina vyplňující prostor mezi uskutečněním skoku a vyzdvižením nové instrukce do prvního stupně linky proudového zpracování dat. Je zřejmé, že budeme-li poskytovat lince proudového zpracování souvislý proud nezávislých operandů, který linku zcela zaplní, zajistíme výrazné zvýšení výkonu celého procesoru. Takovým případem je zpracování vektorů, kde stejná operace na jednotlivé složky vektoru může využívat buď plné nezávislosti provádění operací nebo očekávané závislosti mohou být již předem v procesoru ošetřeny.

6.5.2 Vektorové procesory

Vektorový procesor je procesor, který může současně zpracovávat celý vektor jednou instrukcí. To znamená, že musíme zavést předpoklad existence instrukcí, jejichž operandy jsou vektory. Například uvažujme následující instrukci sčítání

```
C = A + B,
```

která jak ve skalárním, tak vektorovém stroji představuje provedení operace *přičti obsah A k obsahu B a ulož výsledek do C*. Ve skalárním stroji jsou operandy čísla, ale ve vektorovém procesoru jsou operandy vektory a instrukce určuje stroji, aby vypočítal součet vždy páru elementů vektoru. Zvláštní registr v procesoru, který se obvykle nazývá **registr délky vektoru**, určuje kolik individuálních sčítání bude nutno provést, aby se sečetl celý vektor.

V praxi se můžeme setkat i s takovým řešením, kdy speciální kompilátor, tzv. **vektorizující kompilátor**, se bude pokoušet reorganizovat cykly takovým způsobem, aby mohly být provedeny v jedné **vektorové instrukci**. Následující cyklus může být proveden v jedné instrukci na vektorovém procesoru:

```
DO 10 I=1,N
A(I) = B(I) + C(I)
CONTINUE
```

ektorizující kompilátor přeloží výše uvedený kód do instrukce, která nastaví délku vektoru rovnou **N** a bude isledována instrukcí vektorového součtu.

Využití vektorových instrukcí se uplatní ve dvou případech. První spočívá v tom, že pro stejný počet erací musí stroj dekódovat daleko méně instrukcí, takže režie řídicí jednotky je výrazně snížena. Současně je i lpovídajícím způsobem snížena šířka pásma potřebná pro přístup do paměti. Ve druhém případě můžeme ozorovat výhody spojené s tím, že instrukce poskytuje procesoru odpovídající zdroj dat. Jakmile je vektorová strukce spuštěna, procesor již ví, že bude muset vyzvednout N párů operandů, které jsou odpovídajícím ůsobem uloženy v paměti. Tím, že je procesoru tato informace předem známa, může ji předat systému pamětí ty mohou posílat páry operandů v pravidelných intervalech. Při použití vhodné organizace pamětí budou páry icházet na příslušný vstup procesoru s rytmem jeden pár za jeden cyklus. Tím se docílí plynulého předávání dat ímo na vstup linky proudového zpracování instrukcí, která zpracuje příchozí data bez časových ztrát. Tato ednost vektorového zpracování je však silně závislá na implementovaném paměťovém systému a pokud ten bude organizován tak, aby poskytoval operandy v odpovídajícím rytmu, výhody zpracování celého vektoru v dné instrukci se sníží.

Při pokusu o systematizaci vektorových procesorů se vychází ze způsobu, jakým instrukce přistupují ke ým operandům. Při organizaci **paměť - paměť** jsou operandy vyzdviženy z paměti a předány do funkční dnotky. Výsledky jsou postupně ukládány zpět do paměti s postupem operace. V návrhu **registr - registr** musí t operandy nejdříve uloženy do sady vektorových registrů[1]. Vektorová operace pak pokračuje tím, že přečte erandy z vektorových registrů a opět do nich zapíše výsledky.

Výhoda strojů s organizací *paměť - paměť* je schopnost zpracovávat velmi dlouhé vektory, zatímco organizace *istr - registr* musí dlouhý vektor nejdříve rozdělit do menších segmentů s pevnou délkou, což je znevýhodněno měrně velkou režií známou jako **startovací čas**[2]. Dlouhý startovací čas ve strojích organizovaných *paměť - měť* je funkcí latence paměti a je delší než čas nutný k přístupu k hodnotě uložené ve vnitřním registru. Avšak mile je jednotka proudového zpracování naplněna, výsledky jsou produkovány synchronně s ukončením ždého cyklu, takže výkonnostní model pro vektorový procesor by bylo možno napsat ve formě

$$T = s + aN$$

le

s	je startovací čas,
N	je délka vektoru a
a	je konstanta určená typem instrukce, obvykle ½, 1 nebo 2.

Rané příklady těchto typů organizací můžeme najít u firmy *Texas Instruments Inc.* a *Control Data Corp.*, kde byly likovány v počítačích **Cyber 200** nebo později **ETA-10**. Tyto stroje se objevily v polovině 70. let a jejich ganizace měla přibližně desetileté trvání[3]. Jednou z hlavních příčin, proč tyto stroje zmizely z trhu, byl dlouhý rtovací čas, který byl řádu stonásobku procesorových cyklů. To znamená, že operace s krátkými vektory byly lmi neefektivní, ale i pro vektory, které měly 100 i více prvků, byl stroj schopen poskytnout přibližně polovinu ého předpokládaného výkonu.

Ve strojích organizovaných "registr - registr" se pracuje s relativně krátkou délkou vektoru, obvykle 64 vků, jak je obvyklé např. u strojů typu **Cray**. Potřebný startovací čas těchto strojů je však výrazně menší, než ve ojích organizovaných "paměť - paměť". Výsledkem je, že tyto stroje jsou daleko efektivnější pro operace s átkými vektory. V případě dlouhých vektorů musí být sada vektorových registrů naplněna další skupinou erandů předtím, než může být použita, což ve svém důsledku vede k prodloužení "efektivního" startovacího

o vektorových registrů je možno uložit jenom segment celého vektoru, např. 64 prvků, provést operaci, uložit následující gment, znovu provést operaci a tak pokračovat až do dosažení konce vektoru..

as mezi počátkem instrukce a časem, kdy první výsledek opustí jednotku proudového zpracování

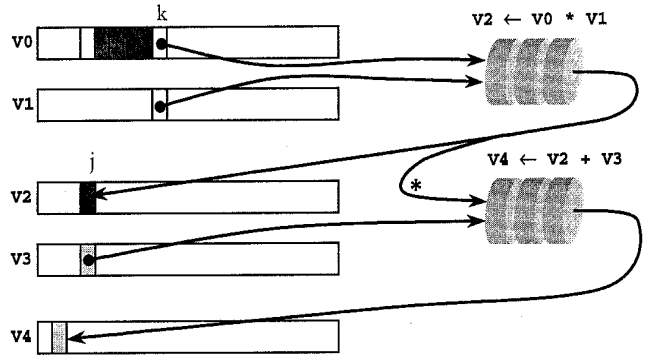
odrobnosti o těchto strojích lze najít v lit. A2.

času[1]. Právě převaha operací s krátkými vektory způsobila, že tyto stroje jsou dominantní na trhu a muzem setkat s výkonnými stroji firmy Cray Researche Inc. pod označením **Y-MP** a **C-90**. Stejný přístup k návrh vektorových procesorů je však používán i ostatními výrobci, např. japonskými výrobci počítačů - firmam Fujitsu, Hitachi a NEC. Hodinový cyklus na těchto moderních vektorových procesorech se pohybuje od 2,5 n (NEC SX-3) do 4,2 ns (Cray C-90) a výkon jednoho procesoru se pohybuje mezi 1 až 2 GFLOPS (měřen pomocí Linpack Bench Marks).

Vektorové zpracování instrukcí staví na efektu výrazného zrychlení výpočtu použitím proudové zpracování instrukcí. Avšak závislost mezi operacemi může tento efekt eliminovat. Zvláště pak v případ vektorového zpracování, kdy závislost na prvním prvku může způsobit zpoždění odpovídající délce zpracován celého vektoru. Tato situace se řeší pomocí **řetězení vektorů**, poprvé použitého v počítačích **Cray** Předpokládejme, že ve vektorovém procesoru pracujícím s vektory o délce 64 položek existují dvě následujíc vektorové instrukce

```
V2 = V0 * V1
V4 = V2 + V3
```

Výsledek první instrukce je operandem v druhé instrukci. Avšak vzhledem k tomu, že se jedná o vektorov instrukce, je zřejmé, že první instrukce bude směrovat až 64 párů čísel do jednotky proudového zpracování da která provádí operaci násobením. Přibližně v polovině této vektorové instrukce se stroj ocitne ve stavu, kd několik prvních prvků **V2** bude obsahovat právě provedené součiny, zatímco součiny, které by měly být uložen do dalších prvků **V2**, jsou stále v lince proudového zpracování dat. Zbytek operandů je stále ve vektorových registrech **V0** a **V1** a očekává, že bude vyzvednut a předán na vstup jednotky proudového zpracování dat. Tat situace je ukázána na obr. 59, kde operandy vektorových registrů **V0** a **V1**, které jsou právě zpracovávány násobící jednotkou proudového zpracování dat, jsou indikovány šrafováním. V tomto bodě systém právě provádí vyzvednutí `V0[k]` a `V1[k]` a předání těchto hodnot do prvního stupně jednotky proudového zpracování dat. Součin `V2[j]` právě opouští jednotku proudového zpracování dat.



Obr. 59 Řetězení vektorů

Metoda řetězení vektorů spočívá na principu, který je vyznačen na obrázku cestou označenou *. Zatímco V2[j] se ukládá do vektorového registru, je současně směrován na vstup sčítací jednotky proudového zpracován dat, kde je k němu přidružen odpovídající vektor V3[j]. Jak obrázek ukazuje, druhá instrukce se může zači provádět ještě předtím, než první instrukce skončí, a protože se provádějí obě tyto instrukce najednou, stroj bud produkovat dva výsledky za jeden cyklus (V4[i] a V2[j]) místo jednoho.

Bez využití řetězení vektorů by špičkový výkon počítače **Cray-1** byl v okolí 80 MFLOPS (jedna jednotk proudového zpracování dat produkuje výsledek každých 12,5 ns nebo též 8.10^7 výsledků za sekundu). Tím, ž zřetězíme tři jednotky proudového zpracování, obdržíme ve stejném čase tři výsledné hodnoty, což poskytuj špičkový výkon teoreticky až 240 MFLOPS.

[1] Naplnění vektorových registrů obsahem paměti však může být řešeno v organizaci počítače tak, že výsledný efekt na startovací čas je minimální.

Principiálně může být řetězení vektorů implementováno i ve vektorových procesorech pracujících
ásobem "paměť - paměť", avšak vyžadovalo by mnohem větší šířku pásma. Bez použití řetězení vektorů musí
stovat tři "kanály", dva jsou použity pro vstup proudu operandů a jeden pro uložení proudu výsledků, s
užitím řetězení vektorů bude nutno realizovat pět kanálů, z toho tři pro vstup a dva pro výstup. Tato
ıopnost řetězit operace umožňuje více než dvojnásobně zvýšit výkon v organizaci "registr-registr", a tím
/hodňuje tuto koncepci před návrhy typu "paměť-paměť".

5.3 Superskalární procesory

Koncepce použité ve vektorových procesorech realizovaných jako sálové počítače v letech 1970 (např. DC-6600 a Cray-1) představovaly špičku tehdejšího designu, avšak vývoj požadoval stále vyšší výkony ocesorů. Poměrně jednoduchou variantu výkonnějšího návrhu představovaly rozmáhající se procesory mezeným souborem instrukcí - procesory RISC. Původní stroje realizované v architektuře RISC kroprocesorovou technologií byly jednoduché procesory na jednom čipu. Jak postupoval vývoj technologie, yšoval se i prostor, který byl na jednom čipu k dispozici. To dalo prostor myšlence nejít při zvyšování výkonu stou zvyšování složitosti architektury, ale využít dodatečný prostor k lepší implementaci technik, které valitňovaly a zrychlovaly provádění instrukcí v některé ze současných architektur. Základními principy, které se likovaly, byly např. vyrovnávací paměti cache na čipu a jednotky pro proudové zpracování instrukcí.

Uplatnění nalezly i variace architektur a na ně navazující organizační řešení procesorů. Jedním z kroků v nto vývojovém procesu je tzv. **superskalární procesor**. Tento název značí, že tyto procesory jsou skalární ocesory[1], které však mohou vykonávat více než jednu instrukci v každém cyklu. Klíčem k vykonávání instrukcí uperskalární architektuře je jednotka pro vyzvednutí instrukcí, která může akceptovat více než jednu instrukci aměti v daném časovém okamžiku. Další důležitou součástí je dekódovací logika instrukce, která je schopna zhodnout, zda-li vyzdvižené instrukce jsou vzájemně nezávislé a tudíž je možno je provádět současně. A v poslední řadě je třeba mít dostatečný počet vlastních exekučních jednotek, které budou schopny zpracovávat kolik instrukcí v jednom časovém okamžiku. Poznamenejme, že exekuční jednotky mohou být jednotky oudového zpracování instrukcí, např. sčítačky nebo násobičky v pohyblivé řádové čárce. Důležité bude, aby acovní cykly jednotlivých částí superskalárního procesoru byly synchronizovány, tedy např. aby cyklus jednoho ıpně jednotky proudového zpracování byl přizpůsoben cyklu vyzvednutí a dekódování instrukce, jinak se hoda této organizace procesoru ztratí. Charakteristickým znakem superskalární architektury je použití ralelismů na instrukční úrovni.

Pro objasnění základního principu superskalární architektury uveďme příklad výpočtu, který se velmi často opakuje v počítačové grafice, tedy rotaci a posunutí množiny párů (x, y) o úhel Θ a vzdálenost (x_d, y_d):

$$x_i' = x_i \cos\Theta - y_i \sin\Theta + x_d$$

$$y_i' = y_i \cos\Theta + x_i \sin\Theta + y_d$$

Vektorový procesor může načíst hodnoty párů (x, y) do dvou vektorových registrů a potom použít ktorovou instrukci. V superskalární architektuře jsou tyto operace kompilovány překladačem do cyklu a nstanty x_d, y_d, $\sin\Theta$, $\cos\Theta$ jsou uloženy do registrů ještě předtím, než tato smyčka je spuštěna. Upravený znam překladu výše uvedené transformace může mít následující formu[2]:

```
L:    load    R8, x[i]
      fma     R10, R8, cosΘ, xd
      load    R9, y[i]
      fma     R11, R9, cosΘ, yd
      fma     R12, R9, -sinΘ, R10
      store   R12, x'[i]
      fma     R13, R8, sinΘ, R11
      store   R13, y'[i]
      branch  L
```

Mimo instrukce, jejichž funkce je zjevná z názvu je použita instrukce **fma w,x,y,z** , která vykonává násobení v pohyblivé řádové čárce společně se součtem, tedy aritmetickou operaci

$$w = x * y + z .$$

Poznamenejme, že dobře navržený kompilátor bude pracovat při překladu velmi opatrně tak, aby proložil instrukce čtení nebo uložení dat s instrukcemi zpracování těchto dat. Cyklus má celkem osm instrukcí, když nepočítáme poslední instrukci skoku, a osm operací v pohyblivé řádové čárce (dvě na jednu instrukci **fma**) v každém cyklu iterace. Během celého cyklu procesor inicializuje jednu operaci v pohyblivé řádové čárce na jednu instrukci. Protože instrukci skoku provádí obvykle sama jednotka pro vyzvednutí instrukcí, nejsou zde hluché cykly, kdy by jednotka pro práci v pohyblivé řádové čárce byla v nečinnosti. Tímto způsobem může stroj vydat vždy jeden výsledek za jeden cyklus pro libovolně dlouhý vektor tak dlouho, dokud veškerá potřebná data budou uložena ve vyrovnávací paměti[1].

Příklad

Tímto způsobem se dosáhlo u počítačového systému RS/6000 výkonu měřeného v Linpack Bench Mark (Gausova eliminace) až 104 MFLOPS a teoretický špičkový výkon činil 125 MFLOPS při hodinovém kmitočtu 62,5 MHz. Porovnáme-li to s počítačem Cray-1S s hodinovým kmitočtem 80 MHz, pak obdržíme hodnoty téměř srovnatelné (výkon 110 MFLOPS a teoretický špičkový výkon 160 MFLOPS).

Výrazná výhoda superskalárního přístupu spočívá v tom, že superskalární návrh nevyžaduje kompilátor, který bude vektorizovat program a za tím účelem detekovat cykly a měnit je na vektorové instrukce. Superskalární stroj však vyžaduje velmi dobře navržený kompilátor, který bude správně alokovat zdroje a rozvrhovat operace tak, aby využil výhody všech operací, které stroj poskytuje. Superskalární návrh je rovněž flexibilnější a snadněji aplikovatelnější ve větší šíři aplikací než vektorové zpracování.

6.5.4 Přerovnání instrukcí

Předpokládejme procesor používající proudové zpracování instrukcí, který zpracovává program v sedmi stupních tak, že instrukce prochází např. fázemi

F	vyzdvižení instrukce z paměti
D	dekódování instrukce
I	rezervace zdrojů – registrů
E	exekutivní část 1
E	exekutivní část 2
E	exekutivní část 3
W	uložení výsledků operace do paměti.

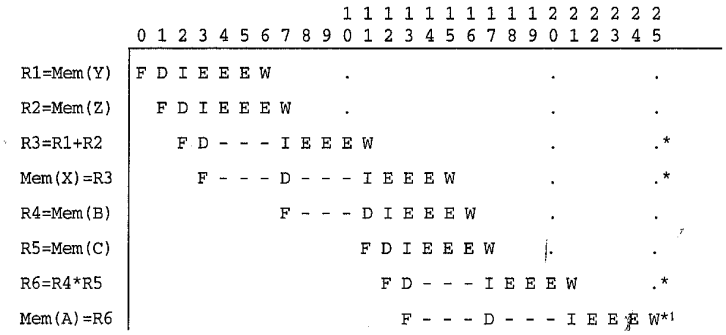
Každý stupeň potřebuje pro svoji činnost právě jeden hodinový cyklus a tedy každá instrukce je provedena v sedmi cyklech. Nechť dále je předložen tomuto procesoru program obsahující dvě aritmetické instrukce a nezbytné instrukce pro zavedení proměnných do registrů a uložení výsledků. Postup zpracování programu je uveden v následující tabulce:

---

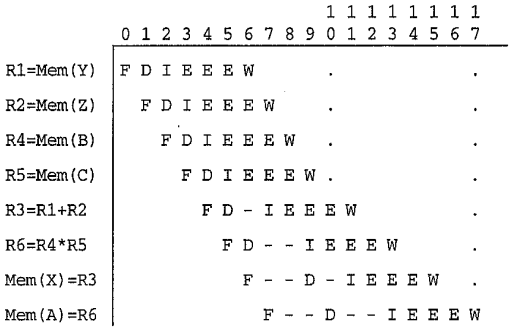
[1] Samozřejmě zde hraje roli i počet cyklů potřebných k provedení jedné instrukce, avšak předpokládáme, že tuto skutečnost bere kompilátor v úvahu.

[2] Zápis používá instrukční sadu procesoru IBM RS/6000.

[1] Pod pojmem skalární procesor se chápe jeden procesor von Neumannova typu.



Pokud budeme těchto osm instrukcí provádět bez pomoci proudového zpracování, pak budou vyžadovat 16 cyklů. Při zpracování v proudové lince je provedení některých cyklů instrukce blokováno, neboť nejsou k dispozici potřebné zdroje. Tyto cykly jsou označeny "–". Takto blokovaná instrukce samozřejmě znemožňuje postup zpracování instrukcí, které za ní následují, neboť obsazuje příslušný stupeň. Nezávisle na tom i tak došlo ke zkrácení běhu celého programu na 26 cyklů. Budeme-li však blíže analyzovat předložený kód zjistíme, že instrukce je možno bez ohledu na změnu funkce programu přerovnat takto



Protože blokování v předcházející verzi je způsobeno tím, že data z pomalejší paměti nebyly včas k dispozici, jsou přesunuty veškeré instrukce naplňující registry na počátek programu. Tím dojde k včasnému uvolnění zdrojů a program bude obsahovat minimum blokovaných cyklů. Jeho běh se zkrátí na 18 cyklů. Pokud vezmeme za rozhodující dobu od ukončení první instrukce do okamžiku konce běhu programu, pak přerovnáním instrukcí došlo k téměř dvojnásobnému zrychlení běhu programu. Uvedený moment dostupnosti zdrojů pro linku proudového zpracování, reprezentovaný uvedeným příkladem, je další možností urychlení práce procesoru. Která část procesoru bude zodpovědná za přerovnání instrukcí a jaký algoritmus bude optimalizátor používat, je součástí návrhu procesoru. V mnoha případech však může být optimalizace a přerovnání instrukcí provedeno již při překladu programu a tak zodpovědnost za urychlení práce přebírá software.

3.5.5 Optimalizace instrukcí skoku

Největším problémem linky proudového zpracování dat je, jak už bylo uvedeno, výskyt instrukce skoku. Instrukce skoku mění tok instrukcí, a tedy instrukce momentálně zpracovávané linkou proudového zpracování se mohou ukázat jako zbytečné. Je tedy nutno linku naplnit novými instrukcemi, což zpozdí běh programu o dobu plnění linky. Mnoho procesorů proto používá optimalizaci skoků pomocí dvou základních metod - predikce skoku a zpoždění skoku.

Predikce skoku používá speciálního bitu v instrukci skoku, jehož hodnota určuje, zda existuje vysoká pravděpodobnost, že podmínka bude splněna[2]. Tento bit je nastaven buď předem během kompilace programu

V řádcích označených * je nutno počkat na výsledek předcházející operace.

Zdali bit vyjadřuje splnění nebo nesplnění podmínky je otázkou vlastní realizace návrhu procesoru.

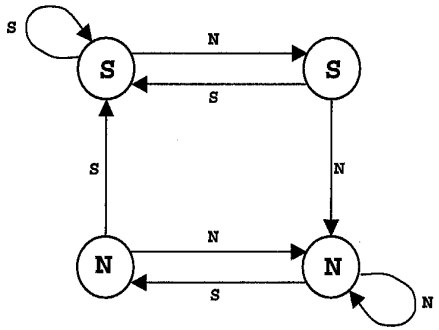
nebo při provádění programu. Typickým příkladem použití predikčního bitu je vyhodnocení stavu řídi proměnné cyklu na jeho konci. Pro cyklus je typické, že instrukce skoku, která cyklus ukončuje vrací řízení n počátek cyklu a pouze jednou, při ukončení cyklu, předá řízení na následující instrukci. Budeme-li analyzov program

```
for(i=0; i<20; i++)
{
     <funkce>
}
```

zpracování funkce
i < 20 ?

pak je zřejmé, že podmíněný skok na konci cyklu bude s daleko větší pravděpodobností směrován na počáte cyklu než na následující instrukci. Hodnota predikčního bitu bude tedy nastavena např. překladačem n předpoklad skoku. Na základě této predikce se do linky zavedou opět instrukce z počátku cyklu a tak bude link vždy využita s výjimkou jediného případu – konce cyklu.

Případ nastavení predikčního bitu při zpracování odpovídá náhodnému nastavení predikčního bitu během kompilace a jeho úpravě při zpracování programu. První rozhodnutí o skoku je provedeno při zpracová instrukce tak, že jsou do linky zaváděny instrukce v pořadí odpovídajícím náhodnému[1] nastavení bitu p překladu. Potom se na základě realizace skoku nastaví predikční bit a pokud je příslušná instrukce znovu použit vychází se z této nové hodnoty predikčního bitu. Bude-li kompilátor nastavovat predikční bity na hodnot odpovídající např. neprovedení skoku, pak budou všechny cykly zatíženy dvojím vyprazdňováním linky nejdříve při prvním průchodu, kdy se musí změnit predikce na "provedení skoku", a pak na konci cyklu kd dojde ke druhé chybě predikce.



Obr. 60 Stavový diagram přechodů pro dvoubitovou predikci skoků.

Uvedená predikce skoku jedním bitem však přes nesporné výhody má i určité nedostatky. Upravíme-li výš uvedený příklad prvotního nastavení predikčního bitu kompilátorem na případ, kdy nastavení predikčního bit bude důsledkem předcházejícího průchodu vnějším cyklem, pak dojde vždy ke dvojímu chybnému naplněn fronty při průchodu cyklem. Tento nedostatek můžeme odstranit zavedením dalších predikčních bitů, jejich hodnoty budou označovat stavy konečného automatu, který bude zachycovat nejbližší historii práce programu V tomto konečném automatu budou definovány stavy, které se vyhodnotí jako predikce provedení skoku stavy, které budou předpokládat neprovedený skok. Hrany stavového diagramu pak budou přiřazeny příznakům skutečných událostí. Příklad stavového diagramu pro **dva bity** je na obr. 60. Označení N znamená že skok neby nebo nebude realizován, naopak označení S slouží pro identifikaci realizace skoku.Zvýšení počtu predikčních bitů umožňuje zpřesnění predikce skoku, a tak i snížení režie linky proudového zpracování na opětné naplněn linky při výpadku pro nesprávnou predikci skoku.

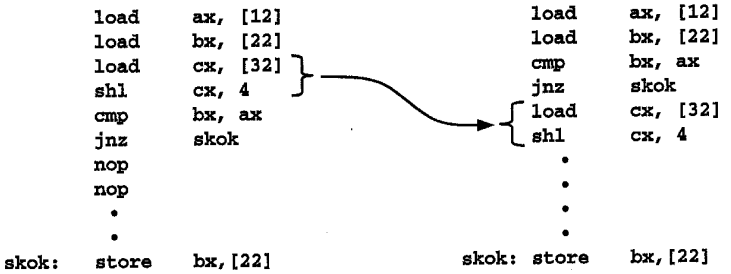
Důvod, proč instrukce skoku působí problémy při zpracování instrukcí v lince spočívá nejenom v narušen proudu instrukcí, ale také v tom, že nové instrukce musí být vyzdvíženy z paměti, která pracuje i několikrá pomaleji než linka proudového zpracování. Tedy, aby linka proudového zpracování mohla plynule pracovat, je nutno ji plynule plnit instrukcemi z paměti. Tyto příkazy k plnění je nutno vydávat s předstihem, aby paměť byla

[1] Pojem "náhodné nastavení" je nutno chápat tak, že v tomto případě kompilátor nevěnuje pozornost analýze programu při nastavování predikčního bitu a poncchá jej v libovolné hodnotě – tu definuje charakter kompilátoru.

s ohledem na svoji rychlost, schopna vydat potřebné instrukce nebo data. To je nemožné v případě instrukce skoku, protože se adresu příští instrukce dovíme až při jejím zpracování. Změníme-li však sekvenční tok instrukcí tak, že instrukce skoku se bude realizovat až o tolik cyklů později, kolik je třeba k vyzdvižení nové instrukce z paměti a prostor za touto instrukcí skoku využijeme pro zpracování instrukcí, které je nutno provést bez ohledu na to, zda podmínka skoku je splněna či nikoliv mluvíme o **zpoždění instrukce skoku**. V případě, kdy prostor za instrukcí skoku bude vyplněn prázdnými operacemi (**NOP** – No Operation), nedocílíme touto metodou žádného urychlení, protože zpoždění s nímž bude příslušná instrukce přečtena z paměti bude stejné jako zpoždění při zrušení špatné fronty instrukcí. Aby došlo ke zrychlení práce procesoru bude nutné, aby kompilátor umístil za instrukci skoku skutečné instrukce programu, které nejsou dotčeny prováděným skokem. Optimalizující kompilátor , který dokáže analyzovat a zaměnit pořadí instrukcí, může odstranit až 90% diskontinuit v práci proudové linky.

Příklad

V následujícím programu bylo použito přeskupení instrukcí, které neovlivňují instrukci skoku z pozice před instrukcí vyhodnocující podmínku pro provedení skoku do pozice za vlastní instrukci skoku. Latence paměti rovná provedení dvou instrukcí, tedy zpoždění skoku je rovné této latenci.



Doba potřebná k provedení programu se zkrátila o čas potřebný k provedení dvou instrukcí. Význam jednotlivých použitých mnemotechnických zkratek pro instrukce, mimo těch, které jsou zřemé, je: shl – posun vlevo, cmp - porovnání operandů, jnz – skok při podmínce nerovné nule.

Posledním způsobem používaným pro urychlení práce procesoru s proudovou linkou pro zpracování instrukcí v případě, kdy se vyskytne instrukce skoku, je použití zvláštní rychlé paměti, do které se odkládá vždy blok několika instrukcí, jehož první instrukce je ta, na kterou v poslední době směřoval skok. Tak se v této **paměti skoků** vytvoří úvodní sekvence instrukcí následující po každé skokové instrukci, které slouží pro překonání doby potřebné k vyzdvižení dalších instrukcí této sekvence z paměti[1]. Obsah paměti skoků se neustále s každou další instrukcí skoku obnovuje a záleží na její velikosti, jak účinné bude její použití při různých rozloženích skokových instrukcí v programu. Výměna bloku instrukcí v případě naplnění paměti skoků se obvykle provádí tak, že se vymění nejstarší blok v této paměti.

6.5.6 Instrukce vázané daty

Při zpracování instrukcí sekvenčním způsobem v proudové lince může velmi často dojít k tomu, že nově zpracovávaná instrukce bude jako jeden ze svých operandů výsledek předcházející instrukce. Budou-li tyto instrukce následovat těsně za sebou, pak jak je vidět např. z Gantova diagramu na obr. 58, dojde ke konfliktu požadavku na vstupy, neboť instrukce a hodnota jejích operandů bude z paměti vyzdvižena dříve než její operandy nabudou správných hodnot v důsledku provedení předcházející instrukce. Řešení tohoto konfliktu mezi požadavky na vstupy, kdy jedna instrukce je svými daty vázána s druhou, bezprostředně následující

[1] Pro tento typ paměti se používá anglický název "branch target buffer". Byl aplikován u některých procesorů RISC.

instrukcí, je možné založit na analýze shody adresy a vyvolání takového stavu linky, ve kterém bude vznik konflikt řešitelný.

První a nejjednodušší způsob řešení takových situací spočívá ve vhodném návrhu kompilátoru, který mů při analýze programu najít dvojice instrukcí vázaných daty a vhodným přeskupením, pokud je možné, tuto vazl upravit s ohledem na parametry linky proudového zpracování. Jedná se tedy o podobné řešení jako při zpoždě skoku, a tedy bude zřejmou snahou při **přeskupení instrukcí** nevkládat do "čekacích" míst jalové instruk **NOP**, ale instrukce, které nejsou závislé na předcházející hodnotě, avšak musí být procesorem vykonán Účinnost analýzy programu kompilátorem a množství takto upravených úseků programu bude určovat celko zrychlení výpočtu s ohledem na řešení konfliktu instrukcí vázaných daty.

Metoda vytvoření "minimální bubliny", neboli zpoždění exekuce instrukce do doby, než je k dispoz potřebný operand, je typicky hardwarovým řešením problémů instrukcí vázaných daty. Obvodový mechanism v procesoru analyzuje tok instrukcí a musí rozpoznat vazbu mezi instrukcemi. V případě, kdy tuto vazl indikuje, zpozdí zpracování následující instrukce o minimální dobu potřebnou k získání hodnoty dotčené operandu pomocí zvláštního vloženého registru, do kterého je uložena předcházející instrukcí – tedy instruk nemusí čekat, až hodnota bude uložena do odpovídajícího "programového" registru. Tento způsob řešení programově nezávislý a byl použit např. u procesorů SPARC. (viz strana 102).

6.6 Další zvyšování výkonnosti procesorů

Vývoj na začátku osmdesátých let směřoval k neuvěřitelnému růstu složitosti procesorů. Současně s nárol na výstavbu takových procesorů vzrůstala i jejich spotřeba a velikost, stejně jako nároky na technologie. V té době se vyhranila skupina konstruktérů, propagujících zjednodušení struktury procesorů, zejména s ohledem na minimální využívání některých implementovaných instrukcí. Formovaly se dvě skupiny, zastánci procesorů označených souborným názvem **CISC**[1], tedy procesorů s úplným a složitým souborem instrukcí, a obhájci kategorie označované jako **RISC**[2], nebo-li procesorů s redukovaným souborem instrukcí. S ohledem na výzkum četnosti výskytu jednotlivých instrukcí argumentovali zastánci procesorů typu RISC, že jak programátoři, tak kompilátory využívají instrukce velmi nerovnoměrně. Testy prováděné na různých programech pro počítače IBM360 ukázaly, že 75% programu používá pouze 8 různých instrukcí (viz Tab. 5) a četnost ostatních instrukcí je tak malá, že často dosahuje pouze zlomku promile. Rozbory, které byly prováděny např. při návrhu integrované verze počítače VAX-11 ukázaly, že 60 % mikroprogramů je použito pouze pro realizaci jedné pětiny instrukcí. Navíc souhrnná četno výskytu všech těchto instrukcí byla pouze 0,2 %.

Statistická četnost instrukcí	%
Čtení z paměti	28,6
Uložení do paměti	15,6
Podmíněný skok	10,0
Zápis adresy do registru	7,0
Odečtení obsahu registru	5,8
Skok do podprogramu	5,3
Logický posuv vlevo	3,6
Zavedení konstanty	3,2

Tab. 5 Výsledky testů statické četnosti

Koncem sedmdesátých let se objevují první návrhy procesorů s omezeným souborem instruk Zkušenosti s procesory RISC byly získávány zejména na počítači IBM 801, který existoval pouze experimentální verzi, a byly využity při návrhu dalších procesorů firmy IBM, zejména mikroprocesoru Power pr pracovní stanice řady RS 6000. První sériově dodávané počítače třídy RISC se objevily na trhu v polovi osmdesátých let, ale skutečný rozvoj této technologie nastal teprve tehdy, když se tato koncepce prosadila c sériově vyráběných mikroprocesorů.

6.6.1 Koncepce architektury RISC

Pod pojmem **architektura RISC** se obecně má za to, že se jedná o procesor, který implementuje pou velmi omezený počet instrukcí a který je schopen pracovat rychleji nežli tradiční návrhy. Tedy v tomto smys

[1] Complete Instruction Set Computer

[2] Reduced Instruction Set Computer

ypu RISC, obvykle jenom s ohledem na tento malý počet instrukcí. Nemálo k tomuto zmatení přispěla i narketingová oddělení firem vyrábějících integrované obvody, když začala staré procesory s návrhem CISC azývat procesory založenými na koncepci RISC, nebo obsahující jádro RISC. Avšak ve skutečnosti mnohdy yto procesory nekorespondovaly se základními charakteristikami, které jsou spojeny s architekturou a organizací rocesoru RISC.

Spolu s návrhem procesorů RISC přišel i nový soubor principů výstavby procesoru a pojem RISC lze. spíše hápat jako filosofii než jako architektonický proud. Důležité body této filosofie návrhu uvádí Patterson [L1] a ezi ně patří:

- jednoduchá sada instrukcí
- instrukce musí být zpracovávány co nejrychleji (bez interpretační úrovně, jako např. mikrokód)
- důležitější než velikost programu je vhodná struktura předloženého programu připravená pro zpracování v jednotce proudového zpracování instrukcí
- kritickou součástí návrhu RISC je technologie kompilátoru a optimalizující kompilátory musí přenést náročnost kladenou na hardware do kompilační fáze.

Mezi uvedenými významnými rysy není rozdíl v důležitosti, který z nich je důležitější a který méně, každý z nich : stejně charakteristický pro architekturu RISC. Pokud odhlédneme od této spíše obecné klasifikace a budeme opisovat architekturu procesoru RISC ve smyslu vlastností procesoru, pak můžeme zdůraznit následující rysy:

- omezený soubor instrukcí
- architektura typu load/store
- pevná délka kódování instrukcí
- rozsáhlý soubor registrů
- zpoždění větvení programu
- vykonávání jedné instrukce v jednom cyklu procesoru

ři posuzování charakteristiky procesorů RISC prováděné z hlediska filosofie návrhu nebo jako prostý soupis lastností procesoru lze pozorovat výrazný rozdíl, který zůstal dodnes populární počítačové literatuře utajen. pory o definici architektury RISC jsou však velmi časté a tak i názory vedoucí k zařazení procesoru do této třídy ıphou být velmi složité. I definice, považovaná za oficiální, a publikovaná v roce 1985 [L2], která říká, že **rocesor třídy RISC obsahuje instrukční sadu, jejíž jednotlivé instrukce mohou být vykonávány jednom cyklu s využitím proudového zpracování**, je mnohdy zpochybňována. Pro hlubší pochopení ýznamu jednotlivých charakteristických rysů architektury RISC se vrať me k charakteristice tohoto procesoru a okusme se hlouběji analyzovat jednotlivé její rysy.

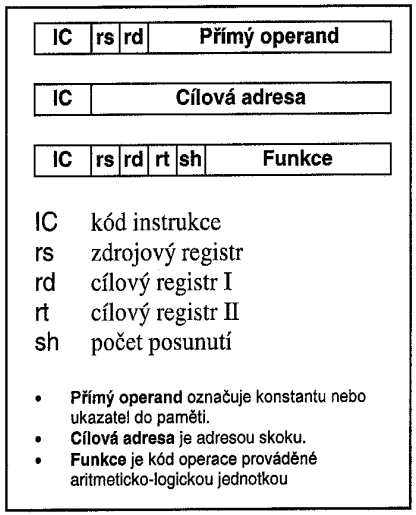
Jedním ze základních rysů procesoru RISC je **regulérní zpracování jednotlivých fází proudového pracování instrukcí a velká délka linky**. Je důležité, aby jednotlivé logické úrovně linky proudového racování byly dobře definovány a vzájemně vůči sobě vyváženy. Instrukce, která je zpracovávána linkou roudového zpracování, musí v jednotlivých stupních strávit stejný čas, a tedy její zpracování musí být ovnoměrně distribuováno podél linky. Předpokládá se, že na zpracování v jednom stupni bude zapotřebí jeden lý pracovní cyklus procesoru. Typické procesory typu RISC mají více stupňů, zúčastňujících se na práci linky roudového zpracování instrukcí, než tři a obvykle používají minimálně pět až šest úrovní. Čím delší je linka roudového zpracování, tím poskytuje větší potenciál pro paralelismus, i když její délka zároveň působí roblémy koordinace zpracování.

V typických počítačích s úplným souborem instrukcí se používá proměnná délka instrukce a obvykle musí ýt přečteno několik slov z paměti, aby mohla být dekódována celá instrukce. Tím se zavádí proměnný časový ement do fáze vyzdvižení instrukce, což by vedlo ke zdržení procesů rozpracovaných na lince proudového racování, neboť dekódující stupeň by musel čekat na nová data (instrukce). Některé procesory se snaží zbavit hoto problému používáním vyrovnávací paměti pro instrukce, do které jsou instrukce načítány v sekvenčním řadí. Podobně mikroprocesory typu CISC používají malý tzv. "prefetch buffer", nebo-li **předsunutou rovnávací paměť pro instrukce**, která je schopna pojmout několik slov. Nejjednodušší způsob, jak se vyvarovat proměnlivého času potřebného pro vyzdvižení celé instrukce, je zakódovat každou instrukci ve stejn délce, a to nejlépe v délce, která bude odpovídat jednomu slovu paměti. V tom případě stupeň linky proudového zpracování dat, který se bude účastnit fáze vyzdvižení instrukce z paměti, bude potřebovat pevný čas l vyzdvižení této instrukce, a tak jedna instrukce může být předána do dalšího cyklu dekódování v rytmu normáln práce linky proudového zpracování dat (to se netýká problémů s větvením programu, které budou popsán dále). Tato vlastnost – **pevná délka instrukce** – je dalším charakteristickým rysem procesorů třídy RISC.

Tím, že zavedeme pevnou délku instrukce, umožníme též **zjednodušení dekódovacích obvodů** instrukčního kódu. Typické procesory typu RISC používají obvykle 6 bitů z celkových 32 pro instrukční kód, což dává prostor pro zakódování 64 instrukcí. Operandy a výsledek se typicky ukládají do registrů. Jestliže tedy bude každý operand instrukce zakódován s použitím 5 bitů, pak můžeme adresovat celkem 32 registrů. Dekódován operačního kódu a přístup k registrům mohou být prováděny současně, což je velmi důležitá vlastnost, jestliže mají být operandy k dispozici při provádění v dalším stupni linky.

Podobně jak fáze vyzdvižení instrukce tak i její exekuční fáze by měla být provedena během jednoho cyklu všude, kde to bude možné. Aritmetické instrukce, které potřebují operandy z paměti, obvykle nesplňují tyto podmínky, neboť doba přístupu do paměti vyžaduje několik cyklů procesoru. K tomu, abychom se vyvarovali této nehomogenity, jsou u procesorů typu RISC používány operace s vnitřními registry, což na druhou stranu vyžaduje poměrně rozsáhlé soubory těchto registrů. Instrukce jako celočíselné násobení nebo dělení mohou být přímo implementovány aritmeticko-logickou jednotkou, ale obvykle také vyžadují víc cyklů ke svému ukončení a mohou způsobit zdržení linky proudového zpracování dat. Některé typy procesoru RISC, jako např. procesor SPARC, nepoužívají přímou implementaci dělení a násobení a odpovídající algoritmy jsou implementovány v software. Procesory CISC pracující s linkami proudového zpracování, jako např. VAX nebo 68 020, připouštějí současné operace mezi registrem a pamětí, což vzhledem k velké latenci přístupu do paměti je příčinou vzniku bublin v lince proudového zpracování dat.

Jestliže předpokládáme, že všechny operandy pro aritmetické a logické operace jsou umístěny v registrech, pak je zřejmé, že obsah těchto registrů musí být nejdříve obnoven zavedením dat z paměti. To se v procesorech typu RISC dosahuje prostřednictvím instrukce "**load**", která může naplňovat registry byty, poloviční slovy nebo celými slovy z paměti. Opačná instrukce "**store**" pak přenáší obsah těchto registrů do paměti. Tento typ práce nazýváme **architekturou load/store**. Bez tohoto speciálního opatření by procesor musel čekat pokaždé, když by potřeboval operand z paměti, na odezvu paměti a linka proudového zpracování dat by byla brzděna. Aby k tomu nedocházelo při operacích **load** a **store**, procesory typu RISC používají tzv. **zpožděného zavedení**. To znamená, že instrukce **load** se sice provede v jednom cyklu, avšak výsledek jejího provedení, tedy nový obsah registru, bude k dispozici o jeden či několik cyklů později. Tedy instrukce, které následují po instrukci **load**, nesmí použít registr, jehož obsah má být obnoven, jako svůj argument. Ve většině případů je tato podmínka je zajišťována při kompilaci, kdy se pořadí zpracování instrukcí rozvrhne tak, že časový interval mezi vydáním příkazu k zavedení hodnoty do registru a jeho vlastním použitím byl dodržen. V případech, kdy takové přeskupení instrukcí není možné a kompilátor nemůže ani vložit prázdné instrukce instrukce **load** zadrží linku proudového zpracování dat do té doby, než budou příslušná data zavedena do registru.



Obr. 61 Příklad instrukčních formátů procesoru třídy RISC

Největším problémem, který ohrožuje plynulé zpracování instrukcí v lince proudového zpracování, je větvení programu, jak již bylo uvedeno v kap. 6.5.5. Instrukce jsou totiž čteny z paměti sekvenčně a jakékoliv větvení bude měnit tento sekvenční tok instrukcí. Ve chvíli, kdy k takovému větvení dojde, nová instrukce musí být nalezena a zavedena na vstup linky proudového zpracování dat a její původní, nyní již irelevantní obsah, musí

t zrušen. Statistika ukazuje, že reálné programy obsahují zhruba 15 procent instrukcí, které způsobují větvení ɔgramu. Z tohoto počtu se uskuteční zhruba polovina skoků směrem dopředu v programu a 90 procent ıěrem zpět. U typických procesorů typu CISC v každém takovém případě dojde ke zrušení obsahu linky ɔudového zpracování dat a zavedení nové instrukce na vstup této linky, což vede k výraznému zpoždění.

Procesory typu RISC však používají odlišnou strategii. Podstatné je, že **rozhodnutí o větvení programu se kuteční** co nejdříve, pokud možno **ještě ve stupni dekódování instrukce**. Toho může být dosaženo tehdy, tliže testy na podmínky větvení jsou velmi jednoduché, např. porovnání obsahu registru s nulou nebo tování podmínkového příznaku. Tak může dojít k tomu, že na konci dekódovací fáze procesor začne s hledáváním instrukce z nové adresy a to přesto, že během dekódovacího cyklu byla vyzvdižena instrukce, která í bezprostředně po instrukci, jež způsobila větvení. Abychom se vyvarovali žablokování linky proudového racování dat, provedeme tuto instrukci nezávisle na výsledku rozhodnutí o větvení. Tento způsob řešení tvení programů, který nazýváme **zpoždění instrukce skoku** (delayed branching) byl popsán v kap. 6.5.5. Při ekladu programu se kompilátor obvykle pokouší využít tento tzv. zpožďovací interval (delay slot) a umístit do j použitelnou instrukci. Některé procesory s velmi dlouhými linkami zpracování dat mohou používat i více hto zpožďovacích intervalů. Čím více zpožďovacích intervalů bude muset být akceptováno kompilátorem, ı složitější bude umístění užitečné instrukce do těchto intervalů, a proto v mnoha případech si kompilátory dnodušují práci a vkládají do těchto intervalů prázdné instrukce (**NOP**).

Technika zpoždění instrukce skoku není ojedinělá a procesory typu RISC používají ještě další techniku, tzv. **tvení s nulovým cyklem**. Pro tento případ se používá zvláštního hardware, který se pokouší o predikci cíle tvení ve chvíli, kdy je instrukce větvení v programu vyzdvihována z paměti. Jakmile predikce určí ıpravděpodobnější cíl instrukce skoku, následující instrukce je vyzvednuta z této predikované adresy. V tomto ípadě není nutné za instrukci skoku vkládat zpožďovací intervaly. Když se predikce nepodaří, procesor se chová stejně jako klasický procesor CISC, zruší obsah linky proudového zpracování dat a vyzvedne novou ıtrukci ze správné adresy. Některé procesory používají speciální jednotku, která se zabývá pouze obsluhou racování instrukcí větvení programu. V těchto procesorech pro zpracování větvení (branching processor) jsou edzpracovány všechny instrukce skoku tak, že aritmetický procesor dostává ke zpracování pouze sekvenční oud instrukcí.

Interakce mezi zpožděným zaváděním obsahu registrů a zpožděným větvením programu mohou být velmi ɔžité. Všechny výhody architektury RISC tak mohou být docíleny pouze tehdy, jestliže kompilátor dokáže eskupit instrukce tak, aby za sebou následovaly v optimálním pořadí. Architektury RISC **maximalizují učinnost mezi hardwarem a softwarem procesoru**, a tak optimalizující kompilátory nejsou jenom hodnou vlastností systémů pracujících s procesory RISC, ale jejich nezbytnou součástí.

Ve všech předchozích popisech vlastností procesoru RISC jsme předpokládali, že instrukce jsou zvedávány, dekódovány a prováděny v jednom instrukčním cyklu, což vyžaduje dostatečně velkou šířku pásma ístupu do paměti. Procesory typu RISC jsou závislé na poměrně složité hierarchii, pokud mají pracovat plnou chlostí. V mnoha případech se v těchto procesorech vytvářejí oddělené vyrovnávací paměti pro instrukce a pro ta, aby se zabránilo kolizím nebo kontencím na sběrnici, propojující procesor a paměť, když jsou vyzvedávány ıtrukce a nebo je prováděn přístup do paměti za účelem uložení nebo vyzdvižení dat. To je důvodem, proč tšina procesorů typu RISC zahrnuje vlastní bloky řízení přístupu do paměti.

Z předcházející diskuse je zřejmé, že všechny vlastnosti procesorů RISC jsou součástí obecné strategie, která ıěřuje k tomu, aby zajistila nepřerušený chod linky proudového zpracování dat a v tomto smyslu i paralelní ovádění sekvenčně kódovaného programu. Všechny výše uvedené metody, jako pevná délka instrukce, ırdwarové dekódování instrukce, zpožděné zavádění a zpožděné větvení atd., jsou jenom cestami k dosažení dného prostupu instrukcí linkou proudového zpracování.

6.2 Taxonomie procesorů RISC

Pro porovnávání jednotlivých skupin procesorů třídy RISC vyjdeme z taxonomie, která bude schopna ɔpsat nejdůležitější aspekty architektury a zahrnout relevantní charakteristiky, uvažované při návrhu procesorů ISC. Mezi tyto základní rysy architektury a organizace mimo jiné patří:

- **Šířka slova**, která u současných procesorů typu RISC je většinou 32 bitů nebo 64 bitů.
- Oddělená nebo společná **vyrovnávací paměť**, která je pro mnoho procesorů typu RISC běžná. Účinnost obou těchto metod je obvykle velmi podobná, avšak mnoho procesorů RISC používá oddělené vyrovnávací paměti pro instrukce a pro data. Vyrovnávací paměti mohou být umístěny na čipu nebo mimo čip. Umístění na čipu nebo co nejblíže procesoru umožňuje dosažení rychlejšího přístupu procesoru k této vyrovnávací paměti.
- Práce s **harwardskou nebo princetonskou architekturou**, tedy oddělení paměti dat a paměti instrukcí nebo použití společné paměti. Například procesor MIPS R3000 používá tzv. princetonskou architekturu, tedy společnou sběrnici pro přístup k datům i instrukcím. Procesory typu Motorola 88000 nebo Analog Devices Shark používají tzv. hardwardskou architekturu, která odděluje paměť dat od paměti instrukcí. Je dlužno poznamenat, že hardwardská architektura neznamená nezbytné rozdělení i hlavní paměti.

Příliv instrukcí k procesoru může procházet ještě další úrovní paměťové hierarchie. Rychlá vyrovnávací paměť pro předběžné vyzdvižení instrukcí, která je vložena mezi regulérní vyrovnávací paměť instrukcí a procesor, čte z řádné vyrovnávací paměti instrukce s předstihem a v sekvenčním pořádku je předkládá na vstup linky proudového zpracování. Toto uspořádání se nazývá **prefetch buffer** - předsunutá vyrovnávací paměť instrukcí. Podobně jako předsunutá vyrovnávací paměť pro instrukce, je realizována i zvláštní vyrovnávací paměť pro data, tzv. zápisová vyrovnávací paměť - **"write buffer"**. Tato speciální paměť je použita procesorem k ukládání dat do paměti, a díky ní procesor nemusí čekat, až data budou do vyrovnávací paměti uložena. Jedinou jeho starostí je vydání požadavku na zápis do vyrovnávacího zapisovacího buferu a speciální část hardware tento požadavek autonomně vyřídí.

Organizace procesorů RISC často používají **koprocesoru**, což znamená, že proud instrukcí je současně analyzován dvěma nebo více procesory (např. procesor pro práci s čísly celými a s čísly v pohyblivé řádové čárce). Každý procesor si pak z proudu instrukcí vybírá ty, které může zpracovat a ostatní interpretuje jako prázdné instrukce (NOP). Tímto způsobem mohou být operace s celými čísly a čísly v pohyblivé řádové čárce vykonávány současně ve dvou různých procesorech. Tyto procesory pak mohou komunikovat prostřednictvím sdílené oblasti paměti nebo speciálních řídicích signálů.

Architektura a organizace využívající **více funkčních jednotek** obsahuje centrální dekódovací jednotku, která vydá povel k vykonávání instrukce v příslušné jednotce podle toho, jak zpracovávaná instrukce byla dekódována. Například dekódovací jednotka může odstartovat sčítání celých čísel v jednotce sčítání celých čísel a o jeden cyklus později může spustit násobení čísel v pohyblivé řádové čárce v násobící jednotce apod. Tento typ zpracování používají například procesory Motorola 88 000 nebo IBM RS/6000, zatímco procesory SPARC a MIPS používají architekturu koprocesorovou.

V architektuře používající koprocesorů se každý procesor zabývá svými vlastními registry a výměna obsahu registrů je zprostředkována pomocí sdílené paměti. V architektuře používající funkční jednotky existují dvě možnosti. Buď je realizován **společný soubor registrů**, který může být použit všemi funkčními jednotkami, nebo každá funkční jednotka může pracovat se svým **privátním souborem registrů**. Existuje i kombinace těchto dvou extrémních řešení.

Zvýšení výkonu exekučních jednotek může být docíleno prostřednictvím širších datových cest v interní architektuře procesoru. To vede k rozdílu výkonnosti mezi procesory používajícími 64 bitů a 32 bitů.

Tradičně bylo řízení programu zprostředkováno podmínkovými bity, které byly nastavovány jako vedlejší efekt zpracování aritmetických nebo logických operací. Architektury typu RISC umožňují nastavit tyto podmínkové bity explicitně v jednom z obecných registrů. Tento registr potom může být testován instrukcemi větvení programu. Strategie testování explicitních bitů v registru je výhodná zejména proto, že odstraňuje problémy spojené s prací dlouhé linky proudového zpracování dat, kde nemusí být vždy zcela jasné, která instrukce změnila podmínkový kód naposledy.

Protože procesory typu RISC pracují s velkými soubory registrů, řízení práce s tímto souborem patří k základním charakteristikám procesoru RISC. Existují tři různé způsoby, jak je možné zajistit správné rozvržení použití těchto registrů. V prvním případě ponecháme **rozvržení registrů překladači** a softwaru, který během analýzy programu musí zajistit, aby při jeho zpracování nedocházelo ke kolizím při použití jednotlivých registrů. Druhé řešení staví na **použití speciálního hardwaru**, který sleduje použití a dostupnost jednotlivých registrů. Kdykoliv se vyskytne požadavek přístupu do registru, který ještě není uvolněn, tento požadavek se pozdrží do doby, kdy bude registr k dispozici. Poslední řešení je založeno na **dynamickém přejmenovávání registrů hardwarem**. V tomto případě, pokud např. dvě instrukce požadují registr R2 pro uložení dočasného výsledku, pak jedna z těchto instrukcí bude mít umožněn přístup k tomuto registru a druhá obdrží přístup pouze ke kopii

K2. Tak, jak bude probíhat výpočet, bude skutečný registr K2 naplněn výsledky operace podle sekvence instrukcí, která byla původně procesoru předložena.

V procesorech typu RISC s mnoha funkčními jednotkami může být důležitým parametrem **hloubka linky proudového zpracování** instrukcí v každé jednotce. Jednotky pracující s čísly v pohyblivé řádové čárce jsou obvykle implementovány využitím delší linky proudového zpracování, než linky ve funkčních jednotkách pracujících s celými čísly. Důležitým momentem je koordinace těchto různě dlouhých linek tak, aby se předešlo kolizím na jejich výstupech, kdy jeden nebo více jednotek se bude pokoušet k přístupu do souboru registrů.

Pokud výstup prováděcí (exekuční) jednotky je přímo připojen na vstup ostatních výkonných jednotek můžeme hovořit o architektuře podobné řetězení vektorových procesorů. Bude-li návrh architektury umožňovat např. přímé propojení sčítačky a násobičky, pak operace násobení dvou vektorů bude velmi rychlá.

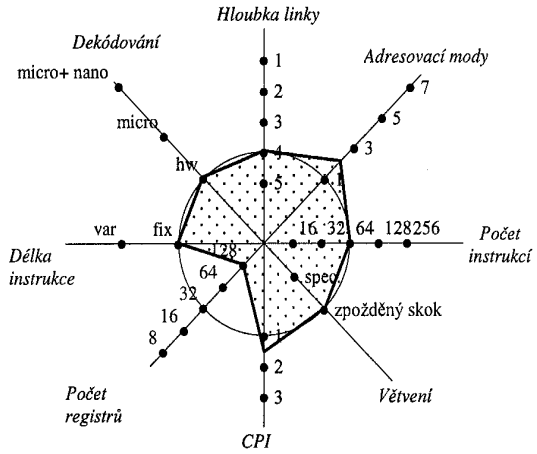
Posledním výrazným architektonickým rysem je obecnost návrhu architektury procesoru RISC. Obecně navržený čip potřebuje ke své práci implementovat i takové vlastnosti, jako je zpracování přerušení, realizaci ochrany jednotlivých úrovní zpracování a management paměti. Téměř všechny procesory třídy RISC poskytují tyto vlastnosti. A to i v případě, kdy byly navrženy pro práci v síti, jako např. transputery, a tyto vlastnosti jsou výrazně modifikovány.

3.6.3 Přehled vlastností komerčně dodávaných procesorů

V následujících odstavcích je uvedeno několik příkladů procesorů patřících do třídy RISC spolu s porovnáním jejich parametrů pomocí tzv. Kiviatových grafů [L3], které patří k užitečným nástrojům při porovnávání charakteristik procesorů. Charakteristiky zobrazené v Kiviatových grafech uvádějí: počet stupňů linky proudového zpracování, počet adresovacích modů, počet instrukcí, metody řešení větvení programu (speciální jednotka pro zpracování instrukcí skoku, zpožděná instrukce skoku), průměrný počet cyklů na jednu instrukci - CPI, počet registrů, délka instrukce (pevná nebo proměnná), a úroveň dekódování (hardware, nikroprogramovaná, mikroprogramovaná a nanoprogramovaná).

3.6.3.1 Procesory typu SPARC

Série procesorů SPARC byla odvozena z procesorů RISC-I a RISC-II, které byly vyvinuty na univerzitě v Berkeley na počátku osmdesátých let. Tato architektura definovaná a používaná firmou Sun Microsystems není sedy vlastním firemním návrhem. Procesor SPARC používá 32bitovou architekturu se společnou vyrovnávací pamětí cache umístěnou mimo čip procesoru. Celá procesorová jednotka je tvořena trojicí čipů[1], které poskytují veškeré potřebné funkce - eden čip pro jednotku pracující s celými čísly, eden pro práci s čísly v pohyblivé řádové čárce a poslední zajišťující ostatní funkce, jako je řízení vyrovnávací paměti nebo hlavní paměti. Návrh procesoru SPARC používá propojení s koprocesorem, kde jednotka pracující s čísly v pohyblivé čárce a jednotka pracující s celými čísly si vyměňují informace prostřednictvím paměti a několika řídicích signálů. Procesor SPARC nepoužívá "prefetch buffer" a všechny jednotky sdílejí obecnou sadu 32bitových registrů. Instrukce jsou dekódovány paralelně, jak v jednotce pro zpracování celých čísel, tak v jednotce pro zpracování čísel v pohyblivé řádové čárce. Práce koprocesoru v pohyblivé řádové čárce začíná na povel z funkční jednotky zpracovávající celá čísla. Nastavení podmínkových kódů a rozvržení práce s registry je plně v režii kompilátoru.



Obr. 62 Parametry procesoru SPARC

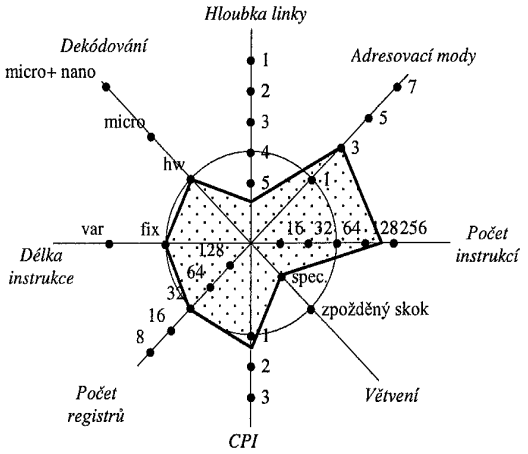
[1] Firma Cypress realizuje tento procesor pouze na dvou čipech.

Základní parametry procesoru SPARC jsou shrnuty v Kiviatově grafu na obr. 62. Přestože procesor SPARC je typickým návrhem ve třídě RISC, existují dvě zvláštnosti, které jej odlišují od ostatních procesorů této třídy. V prvé řadě, procesor SPARC používá koncepci **rozdělení registrů do oken**, aby se minimalizovaly nároky na ukládání a zpětné zavádění hodnot registrů při volání procedur. Tato koncepce staví na skupinách registrů, které jsou označeny pro danou fázi programu jako **pracovní registry**. Při volání procedury se argumenty, místo toho, aby byly předávány prostřednictvím zásobníku v paměti, uloží do pracovních registrů. Součástí volání procedury je přesunutí ukazatele na skupinu pracovních registrů tak, že argumenty budou pro volanou proceduru přístupné - vytvoří se registrové okno. Po návratu z procedury se ukazatel navrátí do původního stavu, tedy okno se opět přesune do původní polohy.

Tento princip registrových oken je hardwarově orientovaná metoda sloužící k optimalizaci alokace registrů. Někteří kritici poukazují na to, že stejného výsledku by mohlo být dosaženo správným rozvržením registrů v době kompilace, což autoři architektury přiznávají. Druhou zvláštností návrhu procesoru SPARC jsou tagované instrukce. Deklarativní jazyky, jako je např. Lisp nebo Prolog používané zejména pro řešení úloh umělé inteligence, používají velmi často tagovaná data. Procesor SPARC poskytuje instrukce, které umožňují snadno pracovat s dvoubitovým tagem spojeným s každým slovem v paměti. Tato vlastnost může zrychlit vykonávání programu v jazyce Lisp o několik procent. Poslední zajímavé architektonické parametry procesoru SPARC jsou počet cyklů na instrukci, který u procesorů SPARC dosahuje hodnoty 1,6 a dva adresovací mody.

6.6.3.2 Procesor IBM - RS/6000

Procesor IBM - RS/6000 využívá architekturu Power[1], která obsahuje celou řadu inovací ve srovnání s procesorem typu SPARC. Základem procesoru RS/6000 je 32bitový procesor s externími vyrovnávacím pamětmi, který využívá hardwardskou architekturu s oddělenými sběrnicemi pro instrukce a data. Výrazným znakem procesoru je velká šířka vyrovnávací paměti instrukcí, do které je současně načteno 128 bitů, uložených ve čtyřslovní předsunuté vyrovnávací paměti (prefetch buffer). Datová sběrnice je široká 64 bitů, aby bylo možno číst a ukládat 64bitová čísla v pohyblivé řádové čárce v jednom cyklu. Architektura procesoru RS/6000 používá vícenásobné funkční jednotky a sestává se ze tří hlavních bloků. Jeden z nich je určen pro řízení a zpracování větvení programu, další pro operace s celými čísly a poslední pro zpracování operací s čísly v pohyblivé řádové čárce. Funkční jednotka predikující větvení programu, využívá pro včasnou detekci větvení obsah předsunuté vyrovnávací paměti - prefetch buferu, který prohledává a pokouší se určit, zda-li příslušný skok bude realizován či nikoliv. Tato jednotka předchází svými operacemi ostatní jednotky procesoru a v mnoha případech je schopna "absorbovat" instrukci skoku, a tak zajistit plynulý chod linky proudového zpracování. Tato technika se nazývá, jak již bylo uvedeno, "větvení s nulovým cyklem" (viz str. 101).



Obr. 63 Parametry procesoru RS/6000

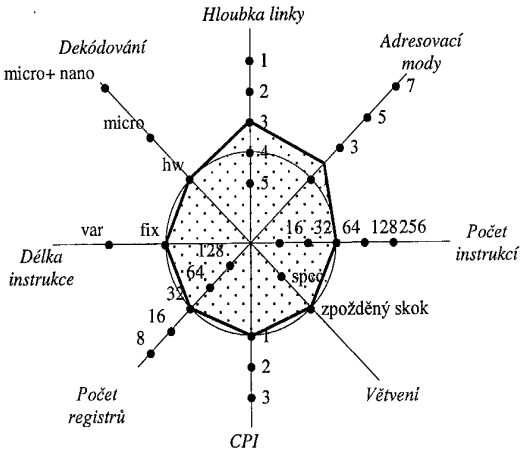
Jednotka pohyblivé řádové čárky poskytuje dvaatřicet 64-bitových registrů. Tyto registry mohou být pevně přiřazeny jedné právě probíhající operaci s pohyblivou řádovou čárkou. Současně může být spuštěna jedna operace sčítání a jedna operace násobení. Procesor je také schopen vykonávat sloučenou operaci násobení a sčítání ve čtyřech cyklech. Tato vlastnost je důležitá při výpočtech skalárních součinů vektorů a jiných obecných matematických funkcí. Všechny jednotky v pohyblivé řádové čárce vykonávají operace s čísly podle standardu IEEE-754.

[1] Performance Optimisation With Enhanced RISC

Podrobnější hodnocení vyjadřuje Kiviatův graf na obr. 63. Procesor IBM - RS/6000 pracuje se dvěma známými linkami proudového zpracování. Jednu pro zpracování čísel v pevné řádové čárce a druhou pro práci s čísly v pohyblivé řádové čárce. První dva stupně obou těchto linek proudového zpracování se nacházejí v jednotce pro zpracování větvení programu. Jednotka pro práci s celými čísly pak přidává další čtyři stupně a jednotka pro práci s čísly v pohyblivé řádové čárce dalších šest. Díky speciální jednotce, zajišťující separátní zpracování větvení programu, nepotřebuje procesor IBM - RS/6000 zpožďovací intervaly. Tato jednotka však sama používá své vlastní registry pro počítání iterací. S pomocí těchto registrů nemusí se zbývající exekuční jednotky zabývat počítáním iterací v cyklech, což je zajištěno uvnitř jednotky pro zpracování větvení programu, která ostatním funkčním jednotkám předá pouze údaj o právě prováděné iteraci. Nezvyklé u tohoto procesoru je dekódování některých operací, zejména operací v pohyblivé řádové čárce, mikrokódem i když délka instrukce je pevná. Práce se soubory registrů je zajišťována zvláštním hardwarem, který používá metodu přejmenování registrů v jednotce pohyblivé řádové čárky. Díky tomu je procesor schopen souběžně zpracovávat instrukce uvnitř smyčky podobně, jako to činí vektorové procesory. Vzhledem k tomu, že funkční jednotky pro zpracování čísel v pohyblivé řádové čárce a pro zpracování celých čísel mohou pracovat současně, lze říci, že návrh procesoru RS/6000 je vlastně superskalární. Průměrná hodnota počtu cyklů na jednu instrukci dosahuje 1,1 a přes svoji poměrně složitou implementaci obsahující daleko více tranzistorů než podobné procesory RISC té době, lze tvrdit, že svými parametry je zařaditelný do třídy procesorů RISC.

6.3.3 Procesor Intel 860

I když procesor **80860** vyvinutý firmou Intel nebyl příliš obchodně úspěšný, jednalo se o první procesor s architekturou RISC vyrobený na jednom čipu. Základem je 32-bitový procesor s harwardskou architekturou, kde sběrnice k vyrovnávací paměti instrukcí je široká 32 bitů a sběrnice k vyrovnávací paměti dat používá šířku 128 bitů, což umožňuje vyzvednout současně čtyři slova. Obě vyrovnávací paměti jsou umístěny na čipu. Podobně jako architektura IBM - RS/6000 i tento procesor používá vícenásobné funkční jednotky a poskytuje pro programátora jednu jednotku pro sčítání čísel v pohyblivé řádové čárce, jednotku pro násobení čísel v pohyblivé řádové čárce a jednu speciální grafickou jednotku. Jádro procesoru obsahuje třicet dva 32-bitových registrů a jednu aritmeticko-logickou jednotku. Alokace těchto registrů je řízena hardwarem pomocí tabulky využití registrů. Soubor registrů pro uložení čísel v pohyblivé řádové čárce obsahuje třicet 64-bitových registrů, které mohou být použity jako patnáct 64-bitových registrů. Pro zpracování součtu součinů, které je často třeba v úlohách lineární algebry a grafiky mohou být sčítací a násobící jednotky vzájemně propojeny. Procesor používá pevnou délku instrukce, která se dekóduje pomocí hardwaru a poskytuje pouze dva adresovací mody. Počet instrukcí je omezen šestibitovým kódem instrukce. Intel udává průměrný počet cyklů na instrukci v okolí 1,1, avšak nezávislá literatura toto číslo upřesňuje na 1,6. Použité linky proudového zpracování dat nejsou příliš dlouhé a obsahují maximálně tři stupně. Speciální grafická jednotka poskytuje programátorovi některé obecné operace potřebné ke zpracování dat v počítačové grafice. Charakteristiky tohoto procesoru jsou shrnuty v Kiviatově grafu na obr. 64.



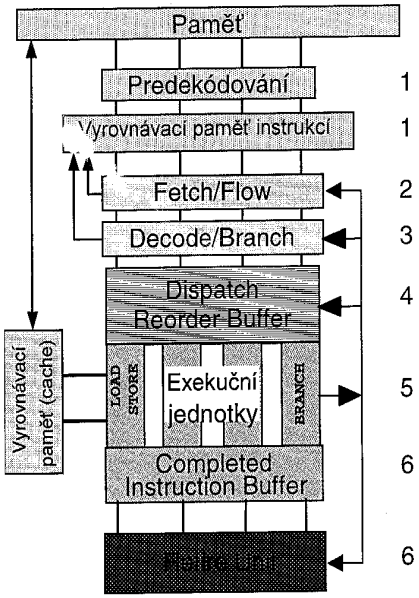
Obr. 64 Parametry procesoru Intel 80860

6.6.4 Architektury Post-RISC

Nárůst výkonů procesorů od doby, kdy byly uvedeny poprvé procesory typu RISC souvisel se změnami technologie směrem k využití výkonnějších a rychlejších implementací, ale i se změnami v architektuře a organizaci procesoru, která se soustředila zejména na **proces zpracování instrukce** a na **charakter použitého instrukčního souboru**. Při bližší analýze architektury a organizace těchto procesorů můžeme zjistit, že výrazné navýšení výpočetního výkonu bylo mnohdy učiněno díky vlastnostem, které již nepatří do architektury RISC. To je důvodem, proč se řadí tyto procesory mezi architektury označované jako post-RISC, i když někteří autoři vytrvale řadí tyto procesory do velké rodiny procesorů RISC.

Nejvýznačnější změnou v oblasti post-RISC procesorů je způsob, jakým je navržená architektura implementována. Superskalární procesory RISC spoléhají na kvalitní optimalizovaný kompilátor, který setřídí a uspořádá instrukce s ohledem na dosažení maximálního výkonu při provádění programu, a hardware procesoru pouze kontroluje přípustnost vícenásobných současných operací s instrukcemi. Procesory, které řadíme mezi architektury post-RISC, jsou mnohem agresivnější v realizaci instrukcí na implementaci hardwaru. Výrazným znakem těchto procesorů je **větší množství paralelních operací** a **provádění operací a instrukcí v jiném pořadí**, než byly generovány překladačem. Tento způsob provádění instrukcí mimo pořadí není nová koncepce. Existovala již na přelomu sedmdesátých a osmdesátých let v počítačích IBM a CDC. Nová je však její implementace na jednom čipu. Nicméně i v této třídě procesorů zůstaly zachovány některé rysy koncepce RISC, například vykonávání instrukcí během jednoho cyklu.

Na obr. 65 je ukázán obecný pohled na strukturu zobecněného procesoru typu post-RISC. Tento teoretický model nereprezentuje žádný konkrétní procesor a použijeme jej pro analýzu procesorů typu post RISC. Pro označení některých částí ve struktuře procesoru jsou použity anglické názvy, zejména pro jejich délku a jednoznačnost. V podrobnějším popisu dále se na ně budeme odvolávat. Podobně jako procesory RISC i uvedená struktura procesoru post-RISC je v zásadě čtyřcestný superskalární procesor, kde každý stupeň proudového zpracování zpracovává skupinu čtyř instrukcí. Instrukce jsou nejdříve vyzvednuty, dekódovány a potom jsou uloženy do vyrovnávací paměti. Na základě dostupnosti potřebných zdrojů a operandů rozhoduje dispečerská jednotka o přidělení instrukce ke zpracování příslušné výkonné jednotce. Z vyrovnávací paměti mohou být instrukce vyzvednuty a distribuovány na spekulativním základě, využívajícím predikci očekávaných skoků. Výsledkem je množina instrukcí v různých stupních zpracování, z nichž ani jedna nedokončila svůj úkol zapsáním konečného výsledku. Jakmile jsou instrukce zpracovány, jsou odloženy do odkládací paměti, kde jsou zpět uspořádány do tvaru, jak byly předloženy procesoru ke zpracování. Jednotlivé části struktury procesoru vykonávají následující funkce:



Obr. 65 Obecná struktura procesoru třídy post-RISC

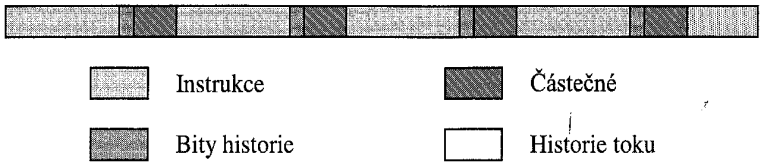
6.6.4.1 Jednotka predekódování a vyrovnávací paměti instrukcí (1)

Instrukce jsou z paměti vyzvedávány ve skupinách, většinou po čtyřech, a během jejich čtení se uskutečňuje proces předběžného dekódování. Na základě této operace je každá instrukce spojena s několika charakteristickými bity a je uložena do vyrovnávací paměti instrukcí, která se obvykle nazývá I-cache. Tyto dodatečné bity slouží

- k identifikaci instrukce, zejména zda-li se jedná o instrukci skoku nebo ne
- k indikaci typu funkční jednotky, kterou bude tato instrukce potřebovat při zpracování
- k určení, zda-li tato instrukce používá či nepoužívá paměťovou referenci (tedy přístup do místa v hlavní paměti).

Jednotka předběžného dekódování (predecode) zvyšuje výkon vlastní dekódovací jednotky instrukce tak, že redukuje logickou složitost řádného dekódovacího stupně jednotky proudového zpracování, a poskytuje

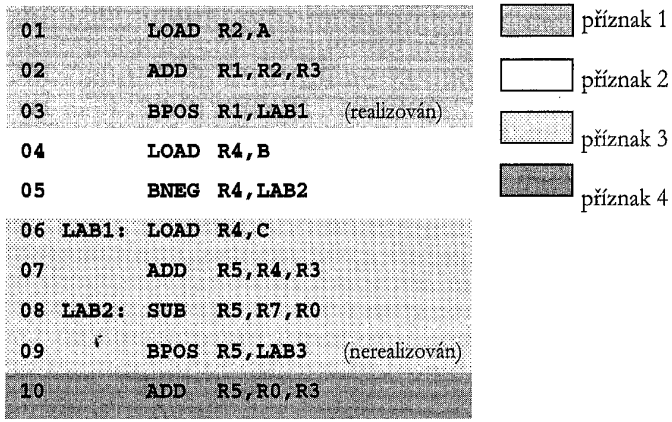
dodatečné informace o dekódování instrukcí stupních, které se nacházejí ještě před vlastním dekódováním. Vyrovnávací paměť instrukcí je rovněž použita k uložení dalších informací o instrukcích. Např. každá skupina čtyř instrukcí má přiděleno několik dalších bitů, které slouží k zaznamenání postupu při zpracování této skupiny. Rovněž každá instrukce obsahuje bity, do kterých se zaznamenává, zda-li u ní došlo v minulosti k větvení či nikoliv.



Obr. 66 Typická položka v předsunuté paměti instrukcí (I-cache).

3.6.4.2 Predikční jednotka (Fetch/Flow) (2)

Ve většině případů proudového zpracování instrukcí je rozhodnutí o tom, která instrukce má být vyzvednuta z paměti instrukcí, obtížné a špatný odhad následující instrukce se odrazí v prodloužení doby zpracování programu. V procesorech typu post-RISC se predikce skoku nerealizuje natolik včas, aby mohla být v únce proudového zpracování použita pro určení místa, ze kterého bude vyzvednuta následující instrukce. Typicky je predikce skoku prováděna až jako součást operace dekódování instrukčního kódu. Avšak jednotka pro vyzdvižení instrukce musí znát instrukci, kterou má v dalším kroku vyzvednout, daleko předtím, než vlastní fáze dekódování začne probíhat.



Obr. 67 Přiřazení příznaků k jednotlivým instrukcím

K tomu, abychom umožnili jednotce pro vyzdvižení instrukcí získat následující instrukci bez čekání na dekódování probíhající instrukce, jsou ke každé skupině čtyř instrukcí přiřazeny bity, ve kterých je zachycena historie jejich zpracování, tedy průběhu této části programu v minulosti[1]. Tyto bity ukazují na další skupinu čtyř instrukcí, které by měly být vyzvednuty. Jestliže budou tyto bity ukazovat na špatnou skupinu instrukcí, budou v následujícím stupni změněny jako důsledek práce predikčního algoritmu nebo vlastního vykonání skoku.

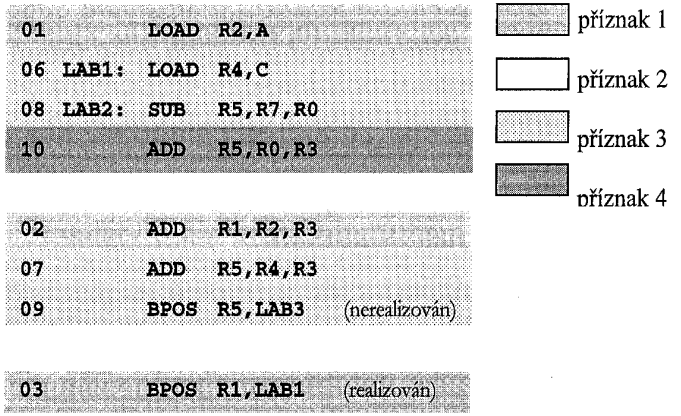
3.6.4.3 Dekódování a skoky (Decode/Branch) (3)

V tomto stupni linky proudového zpracování u procesorů post-RISC je instrukce dekódována a současně je zpřesněna predikce očekávaného skoku. S každou instrukcí skoku se program rozpadne do dvou oddělených

[1] Tzv. "flow hitory bits". Odtud i anglický termín označující tuto část struktury procesoru.

toků instrukcí. Do té doby, než je určena adresa skoku, je správná posloupnost instrukcí, které mají b vykonány, neznámá a procesor použije informace, které má k dispozici (např. informace o chování této instruk v minulosti) k tomu, aby upřesnil původní predikci. Např. bit predikce skoku v tabulce predikcí skoku můž indikovat, jestli je větší pravděpodobnost skok uskutečnit či nikoliv. Podle tohoto upřesnění se mění příznaky následujících instrukcí. Jestliže se později ukáže, že predikce byla chybná, s použitím příznaků instrukcí budo všechny výsledky spekulativně prováděných instrukcí za skokem zrušeny.

Proces postupného upřesňování predikce skoku vede k tomu, že současně existuje větší množství dosu nevyřešených spekulativních skoků. Např. obr. 67 ukazuje instrukce s indikací predikce skoku. Skok v instruk na řádku 03 je predikován jako skok, který se uskuteční, a instrukce skoku na řádku 09 očekává, že se sko neuskuteční. Během dekódovací fáze jsou instrukce zpracovány maximální rychlostí a současně jsou ji přiřazeny příznaky podle předpokládaného dalšího průběhu programu. Např. instrukce na řádku 10 může b spekulativně provedena kdykoliv, avšak jestliže predikce skoku na řádku 09 bude nesprávná a skok se proved pak všechny instrukce označené příznakem 4 (instrukce na řádku 10 a následující, až do **LAB3**) budou zrušeny



Obr. 68 Pořadí provádění instrukcí

6.6.4.4 Přidělování instrukcí a změna uspořádání (4)

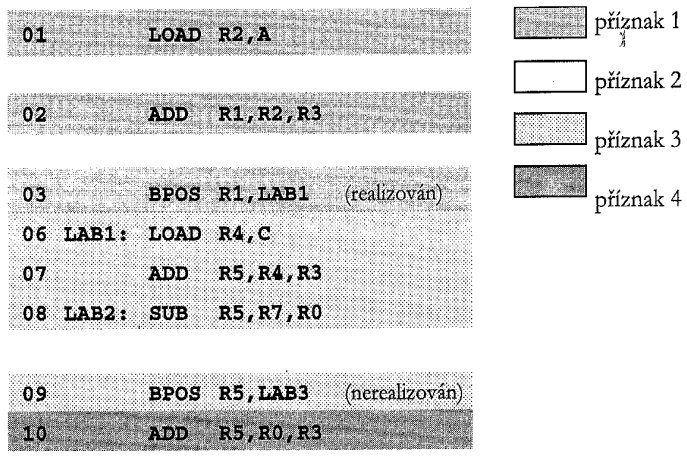
Zvláštní část vyrovnávací paměti, v anglické terminologii nazývaná "dispatch and reorder buffer", slouží uložení instrukcí do front, kde čekají na své zpracování. K tomu může dojít tehdy, když všechny vstupn hodnoty pro zpracování instrukce jsou k dispozici, je uvolněn výstupní registr pro zapsání výsledku operace příslušná výkonná jednotka. Prioritu při zpracování dostávají instrukce, které jsou v této vyrovnávací paměti nejdéle, a instrukce typu **load**. Jakmile se jednou instrukce dostane do této vyrovnávací paměti, stává s původní uspořádání instrukcí v programu bezpředmětným.

Integrální součástí procesu zpracování instrukce je použití mechanismu přejmenovávání registrů, kd instrukce používá při svém zpracování zástupné označení registru ve kterých se uchovávají výsledky tak dlouho než je instrukce po zpracování odložena. V okamžiku odložení instrukce se ze zástupných registrů okopíruj hodnoty, které jsou v nich uloženy, do odpovídajícího registru v architektuře – tedy registru označeného v instrukci programu, nebo je upraven záznam v tabulce přiřazení registrů. Mechanismus přejmenovávání registrů eliminuje závislost toku instrukcí na registrech architektury. Instrukce mohou číst a zapisovat d přejmenovaných registrů a jejich vykonání může pokračovat, zatímco dřívější instrukce obsadily původní registr architektury. Zároveň tento mechanismus umožňuje jednodušší implementaci spekulativního vykonávání instrukcí. V okamžiku, kdy je určena cílová adresa skoku při větvení programu, všechny přejmenované registr alokované pro nerealizovanou cílovou adresu jsou uvolněny a do odkládací paměti jsou uloženy pouze instrukce ve správné větvi programu (tedy instrukce se správným příznakem)

Na obr. 68 je rozveden předcházející příklad z obr. 67, ukazující pořadí provádění instrukcí hypotetického programu. Po vyzvednutí, dekódování a predikci skoku je naplněna vyrovnávací paměti realizující spekulativn provádění instrukcí. V prvním cyklu budou provedeny instrukce 1, 6, 8 a 10, v druhém 2, 7 a 9 a ve třetím cyklu instrukce 3.

6.6.4.5 Exekuční jednotky (5)

Výkonné - exekuční jednotky jsou v obecné post-RISC architektuře podobné jednotkám používaným v architektuře RISC. Stejně jako u procesorů RISC, většina jednotek provádějící práci s celými čísly dokončí operaci během jednoho cyklu. Ostatní jednotky s latencí větší než je jeden cyklus používají proudové zpracování. Tedy každá procesorová jednotka je připravena k tomu, aby akceptovala novou operaci na počátku každého cyklu. Některé exekuční jednotky zpracovávají instrukci ve více cyklech, na což může být z hlediska organizace jednotky nazíráno jako na cyklus jednocyklových instrukcí[1]. Jednotky, které připravují operace pro zpracování nebo které ukládají výsledky do paměti (load/store unit), jsou obvykle schopny pracovat současně na desítkách instrukcí. Tyto instrukce mohou být ukončeny v jakémkoliv pořadí v závislosti na lokálnosti příslušných dat a připravenosti jednotek vyrovnávací paměti cache nebo příslušného bloku hlavní paměti.



Obr. 69 Postup odkládání instrukcí

Jednotka řešící větvení programu předává výsledky vyhodnocení příslušného skoku ostatním jednotkám v lince proudového zpracování a zejména musí

- informovat predikční jednotku (Fetch/Flow), která se zabývá vyzvednutím instrukce a sledováním plynulého toku instrukcí, o chybné predikci v předcházejícím bloku tak, aby tato jednotka mohla upravit tok instrukcí ve vyrovnávací paměti instrukcí (I-cache) a zároveň vyzvednout nové instrukce na správné adrese,
- informovat jednotku dekódování a větvení programu (Branch/Decode) o tom, že došlo k větvení programu tak, aby mohla nastavit tabulky historie větvení programu. Tyto tabulky se nastavují jak v případě špatné predikce, tak i v případě správně predikovaných skoků,
- předat jednotce pro přidělování instrukcí a změnu uspořádání informaci o tom, které instrukce nebudou vykonány a tudíž mohou být z paměti vymazány,
- upozornit odkládací jednotku a vyrovnávací paměť pro dokončené instrukce (viz dále) které instrukce se vzhledem k realizovanému skoku nevykonaly a tudíž se nebudou sestavovat a navracet zpět.

Speciální případy nastanou, když

- jednotka pro zpracování instrukce ve více cyklech právě vykonává instrukci, jejíž výsledky nebudou zapotřebí vzhledem k výsledku vyhodnocení instrukce skoku. Tato jednotka musí obdržet informaci o vyhodnocení výsledku skoku, aby mohla svoji operaci ukončit,

[1] Sem patří např. dělení nebo přesun bloků dat.

- instrukce, která generuje "trap" – vnitřní přerušení, které je uloženo jako příznak u této instrukce. Tento příznak je vyhodnocen ve chvíli, kdy instrukce je zpracována odkládací jednotkou, tedy kdy je vlastní ukončena. Pokud instrukce generující příznak "trapu" nebude díky vyhodnocení instrukce větven programu odložena, zcela přirozeně toto vnitřní přerušení, resp. žádost, nebude vyhodnocena.

6.6.4.6 Odkládací jednotka a vyrovnávací paměť pro ukončené instrukce (6)

Vyrovnávací paměť pro ukončené instrukce obsahuje instrukce, které byly provedeny spekulativním způsobem. Ke každé instrukci je ve vyrovnávací paměti přiřazen její výsledek uložený v přejmenovaných registrech a všechny případné příznaky nebo požadavky na vnitřní přerušení – "trap". Odkládací jednotka pak vybírá všechny provedené instrukce z vyrovnávací paměti v tom sledu, v jakém původně formovaly program, a odkládá je tak, že během jednoho cyklu odloží současně maximálně čtyři instrukce. Současně s odložením instrukcí uloží i výsledky jejich provedení do odpovídajícího registru architektury, které přepíše obsahem přejmenovaných registrů.

Spekulativní provádění instrukcí, tedy jejich provádění v jiném pořadí než jsou předkládány programem vede ke vzniku problému ve chvíli, kdy výsledkem operace instrukce je požadavek na přerušení nebo trap. Z principu návrhu architektury procesoru vyplývá, že jakmile nastane přerušení, pak procesor se musí zastavit právě v místě vzniku tohoto přerušení. To znamená, že instrukce, které se v programu vyskytují po instrukci která způsobila toto přerušení, resp. jejich výsledek, se nemůže odrazit ve stavu stroje. Podobně ve chvíli, kdy se program dostane k instrukci způsobující přerušení, nemohou existovat nevykonané instrukce, které v programu následují před touto instrukcí. Tuto charakteristiku procesoru označujeme jako "precizní přerušení" nebo "precizní trap". Tím, že budeme odkládat instrukce v pořadí, ve kterém byly načteny, docílíme správného zpracování přerušení. Lze říci, že v procesorech třídy post-RISC je v zásadě každá instrukce prováděna spekulativně. Toto je činěno s předpokladem, že žádná instrukce před nimi nezpůsobila přerušení. Instrukce je odložena teprve ve chvíli, kdy odkládací jednotka zjistí, že tento předpoklad je pravdivý.

Při porovnání architektur a organizací třídy post-RISC a RISC je důležité si uvědomit, že měřítkem výkonnosti třídy post-RISC je počet instrukcí, které jsou odkládány po zpracování. Oproti tomu u procesorů třídy RISC, je měřítkem výkonnosti počet instrukcí předaných k paralelnímu zpracování. Procesor třídy RISC vykonává instrukce, resp. je předkládá k paralelnímu zpracování se stejnou rychlostí, jako tyto instrukce odkládá zpět, což neplatí u třídy post-RISC, která využívá spekulativního provádění instrukcí.

Budeme-li pokračovat ve stejném příkladu, který jsme použili pro objasnění operace odkládací jednotky můžeme předpokládat, že všech deset instrukcí bylo provedeno a jsou uloženy ve vyrovnávací paměti provedených instrukcí v odkládací jednotce. V obr. 69 tak může být odložena v prvním cyklu pouze instrukce 1. Ostatní zůstávají ve vyrovnávací paměti a během druhého cyklu bude odložena instrukce 2. Ve třetím cyklu bude uložena instrukce 3 a za předpokladu, že predikce skoku byla správná, mohou být odloženy čtyři následující instrukce - tedy 6, 7, 8 a 9. Zároveň ale je stanovena správná hodnota pro instrukci skoku 9, a tedy v následujícím cyklu bude odložena instrukce 10.

6.6.4.7 Proudová linka zpracování instrukcí u procesorů třídy post-RISC

Základem linky proudového zpracování v architektuře Post RISC jsou tři propojené komponenty (viz obr. 70).

1) Sekce zabývající se vyzdvižením a dekódováním instrukce

2) Exekuční jednotka

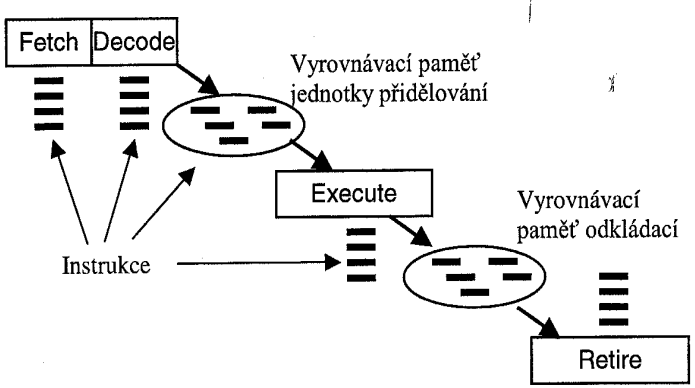
3) Odkládací jednotka

Mezi těmito částmi je vybudováno propojení flexibilními frontami instrukcí čekajících na zpracování. Vyrovnávací paměť pro přidělování instrukcí propojuje jednotku vyzdvižení a dekódování instrukce a exekuční jednotku. Vyrovnávací paměť provedených instrukcí je umístěna mezi exekuční jednotkou a odkládací jednotkou.

6.6.5 Třída post-RISC jako procesor řízený daty

Při analýze architektury a organizace procesorů typu post-RISC bychom mohli tvrdit, že se jedná v zásadě o architekturu procesorů řízených daty (podrobněji viz kap. 11.2.1). Z obr. 70 je zřejmé, že jakmile se instrukce

ocitnou ve vyrovnávací paměti jednotky přidělování instrukcí, jsou předkládány exekučním jednotkám ke zpracování na základě dostupnosti zdrojů, to znamená operandů a potřebných exekučních jednotek. Pokud je k dispozici exekuční jednotka, pak instrukce buď má své operandy k dispozici a její zpracování pokračuje nebo jsou nedefinovány a pak nemůže být zpracována. V procesoru řízeném daty mohou být instrukce provedeny v jakémkoliv pořadí, pokud zdroje požadované pro zpracování dané instrukce jsou k dispozici. Zároveň může existovat velké množství výkonných jednotek, které tyto instrukce budou provádět. Tak, jak se jednotlivé instrukce provádějí, produkují buď výsledek nebo umožňují, aby mohly být provedeny další instrukce. Instrukce, jejichž provedení je umožněno vykonáním předchozích instrukcí, jsou přidány do fronty instrukcí připravených k provádění.



Obr. 70 Linka proudového zpracování v procesorech třídy post-RISC

Klasický procesor řízený daty však musí řešit dva problémy, které nejsou natolik palčivé v procesorech typu post-RISC. Prvním z nich - problém určení, kdy příslušné operandy jsou k dispozici[1] je u procesorů třídy post-RISC zjednodušen, neboť všechny operandy jsou uloženy v registrech. Tedy určení, zda operandy jsou k dispozici nebo ne, se degraduje na určení přítomnosti nebo nepřítomnosti přejmenovaného registru a příslušného příznaku. Druhý problém je zpracování složitějších struktur, např. pole dat. Vzhledem k tomu, že procesor v architektuře post-RISC zpracovává vždy pouze jednoduchý datový typ (jeden typ v jednom registru), je tento nesmírně komplikovaný problém pro obecné procesory řízené daty velmi zjednodušen. Výsledkem celého návrhu pak je, že procesor může využít flexibility stroje řízeného daty v malém měřítku, a tak odstranit jeho nevýhody.

Lze konstatovat, že procesor typu post-RISC se odlišuje od obecného procesoru řízeného daty ve dvou hlavních směrech:

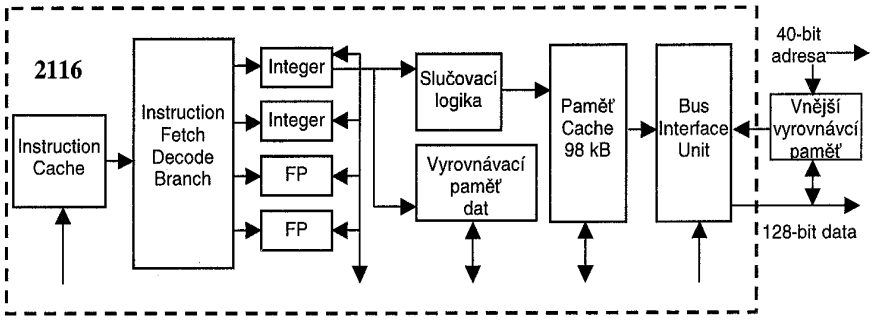
- vzhledem k tomu, že vyrovnávací paměť v jednotce přidělování instrukcí ("Instruction Reorder Buffer") má pevnou délku, všechny aspekty řízení procesoru se omezují na zpracování pevného počtu aktivních instrukcí. V organizaci obecného procesoru řízeného daty toto omezení na počet aktivních instrukcí neexistuje. Tím je umožněno, že procesor třídy post-RISC snadněji identifikuje instrukci připravenou pro zpracování.
- po zpracování instrukcí příslušnou exekuční jednotkou jsou výsledky ukládány v pořadí, které odpovídá původnímu sekvenčnímu charakteru programu. Tím zůstává charakteristika řízení práce jádra procesoru daty skryta.

6.6.6 Příklady vybraných procesorů

V následujících kapitolách je uvedeno několik typických procesorů, které jsou vybudovány na bázi tradiční architektury RISC a vykazují jistou úroveň implementace prvků architektury post-RISC. Typickým představitelem třídy RISC s minimálním vlivem post-RISC organizace je procesor DEC Alpha 21164 a jeho následovník Alpha 21264, který již má výrazné rysy post-RISC návrhu. Procesory jako např. Power PC 604 nebo HP PA-8000 jsou typickým představitelem post-RISC organizace, zatímco procesor Intel Pentium P6 poskytuje post-RISC jádro pro zpracování instrukcí cílového stroje CISC. V této části budou popsány některé typy procesorů, které vykazují výše uvedené charakteristiky.

6.6.6.1 DEC Alpha 21 164 (21 264)

Procesor DEC 21164 je tradiční čtyřcestný superskalární procesor architektury RISC s minimálními prvky post-RISC organizace. Základním znakem tohoto produktu bylo při jeho uvedení na trh použití vysokého hodinového kmitočtu. Tím tento typ procesoru byl konkurencí na trhu pro procesory ostatních výrobců, neboť obvyklý hodinový kmitočet několikrát převyšoval. S ohledem na výši hodinového kmitočtu řádu stovek MHz, tedy odpovídající rychlosti zpracování v exekučních jednotkách, je vynechání některých paralelizmů pochopitelné.



Obr. 71 Vnitřní struktura procesoru DEC Alpha 21164

Tabulka historie skoků, kterou používá procesor DEC Alpha, je udržována dekódovací jednotkou. Obsahuje 2048 jednobitových položek a každá z nich indikuje výsledek posledního rozhodnutí o instrukcích skoku. Tato tabulka je jediná a v procesoru nejsou realizovány další nástroje, které by sloužily k predikci toku programu. Procesor provádí instrukce v pořadí, v jakém jsou vyzvedávány a při provádění instrukcí nepoužívá ani kombinování následujících instrukcí do bloků. Tedy, jestliže má být vyzvednuta instrukce, pak v následujícím cyklu hodin bude pouze tato jedna instrukce zpracovávána. Ačkoliv instrukce jsou zpracovávány v pořadí, v kterém jsou zapsány, je možno nalézt některé typy instrukcí, které se vykonávají ve více cyklech a mimo pořadí. Umožnění operací mimo pořadí vede k jednoduché proudové lince s rychlým zpracováním instrukcí a s nízkou latencí, avšak nelze zajistit precizní přerušení. Takový přístup k návrhu zpracování proudu instrukcí ze strany jádra procesoru je typický spíše pro architekturu typu RISC, která svůj výkon staví na schopnosti kompilátoru rozvrhnout instrukce tak, aby byly nejlépe využity jednotlivé funkční jednotky, aby bylo zabráněno vzniku závislostí mezi daty, a tedy i výpadku linky proudového zpracování.

Zatímco procesor Alpha 21164 stavěl na hodinovém kmitočtu větším než 600 MHz, jeho později modifikace, procesor Alpha 21264, již používala spekulativní provádění instrukcí. Tak bylo demonstrováno, že provádění instrukcí spekulativně neimplikuje použití nízkého hodinového kmitočtu. Procesor 21264 používá hodinový kmitočet 500 MHz a podobně jako většina procesorů post-RISC byl navržen jako čtyřcestný superskalární procesor se spekulativním prováděním instrukcí. Používal jednoduchou predikci skoku s přidaným prediktorem toku programu (predikce toho, co bude v následujícím cyklu vyzvednuto) ke každé vyrovnávací paměti. S ohledem na použití principu přejmenovávání registrů procesor implementuje 80 registrů pro celá čísla a 72 registrů pro čísla v pohyblivé řádové čárce.

Vyrovnávací paměti umožňují, aby ve frontě bylo uloženo 20 instrukcí pro čísla celá a 15 instrukcí očekávajících zpracování v pohyblivé řádové čárce. V jednom cyklu mohou být předloženy ke zpracování čtyři instrukce pracující s celými čísly a dvě instrukce požadující zpracování v pohyblivé řádové čárce. Jednotka proudového zpracování instrukcí pro pohyblivou řádovou čárku a pro sčítání a násobení pracují se čtyřmi cykly. Současně může být aktivních až 32 instrukcí typu load/store. Operace s vnější pamětí využívají velké šířky datové sběrnice a procesor 21 264 umožňuje až osm současných zápisů do paměti mimo čip a osm současných

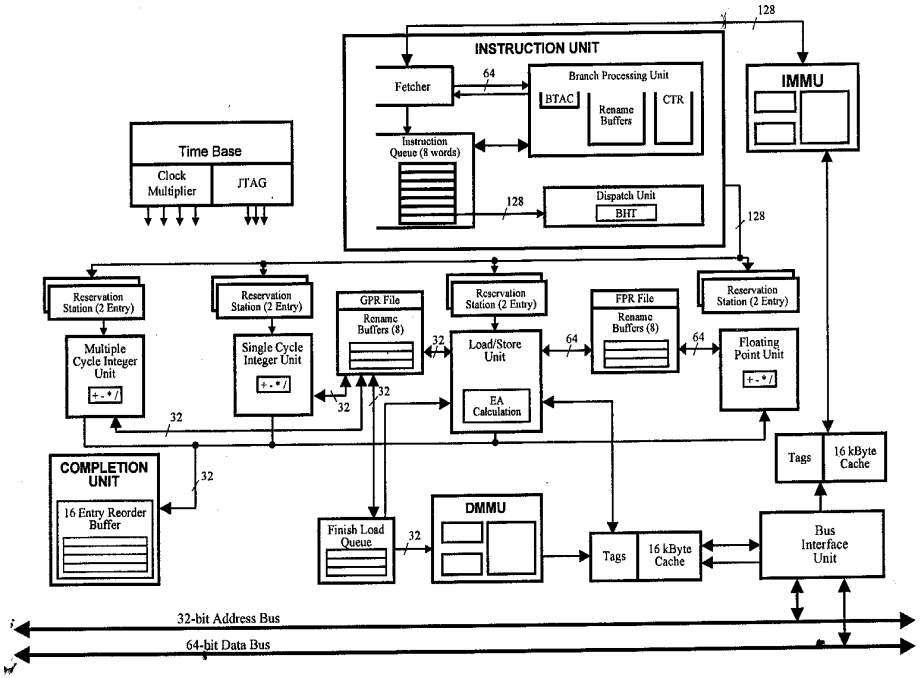
[1] V terminologii počítačů řízených daty se tento stav operace nazývá "matching".

Ústřední knihovna

čtení. Zavedením architektury post-RISC se při stejném hodinovém kmitočtu podařilo zvýšit výkon procesoru Alpha 21264 oproti procesoru 21164 více než 2,5krát.

6.6.6.2 IBM PowerPC 604

Procesory řady PowerPC implementují většinu prvků post-RISC organizace a architektury. Na obr. 72 je znázorněna jeho vnitřní struktura, čtyřcestný superskalární návrh, kde instrukce jsou po čtveřicích vyzvedávány z vyrovnávací paměti na čipu. Instrukce jsou před-dekódovány a uloženy do vyrovnávací paměti instrukcí. Během vyzdvižení instrukce a jejího před-dekódování je provedena předběžná predikce skoku s použitím vyrovnávací paměti cílových adres (Branch Target Address Cache), která může uchovat až 256 položek. Upřesňování predikce v následujících stupních nahrazuje uložené údaje ve vyrovnávací paměti cílových adres při předběžné predikci.



Obr. 72 Bloková vnitřní struktura procesoru PowerPC 604

Vyzvednuté instrukce přicházejí do vyrovnávací paměti, kde tvoří frontu maximálně osmi instrukcí a zde jsou dále dekódovány a přiděleny ke zpracování. V jednom cyklu je možno uložit do fronty až čtyři instrukce a současně jiné čtyři instrukce z fronty mohou být předloženy ke zpracování. Během dekódování instrukcí je upřesňována predikce skoku z předcházejícího stupně a je doplněna o predikční mechanismus založený na dvoubitovém schématu (viz strana 95). Predikční bity jsou ukládány do tabulky historie větvení (Branch History Table). Od tohoto okamžiku jsou instrukce prováděny spekulativně, přičemž jsou do registrů podmínkových kódů doplňovány příznaky, vyplývající z jejich zpracování. Procesor PowerPC 604 obsahuje osm vyrovnávacích pamětí s přejmenovanými podmínkovými registry, které jsou k dispozici po dobu spekulativního provádění instrukce.

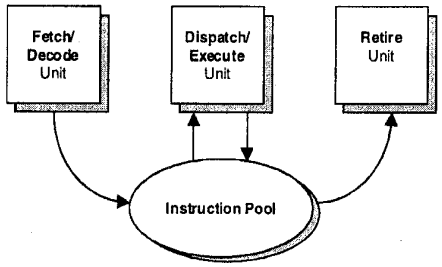
Přidělovací jednotka směruje instrukce z fronty instrukcí k příslušným funkčním jednotkám ve spolupráci s vyrovnávací pamětí pro přerovnání instrukcí. Tato vyrovnávací paměť umožňuje přístup k šesti funkčním jednotkám, kde každá z nich má vlastní "rezervační stanici" ve které mohou být současně umístěny dvě položky – instrukce. Tyto instrukce mohou být předloženy funkční jednotce ke zpracování v jakémkoli pořadí a rozhodujícím momentem pro jejich předání je existence potřebných zdrojů (dat, registrů apod.). Jakmile je instrukce přidělena ke zpracování, je jí přidružen přejmenovaný registr pro uložení výsledku operace. Procesor PowerPC 604 má k dispozici osm přejmenovávaných registrů pro operace v pohyblivé řádové čárce a dvanáct obecných přejmenovávaných registrů. Přidělovací jednotka rovněž rezervuje místo pro zpracovanou instrukci ve

vyrovnávací paměti[1] odkládací jednotky[2]. Z vyrovnávací paměti v odkládací jednotce jsou dokončené instrukce odstraňovány podle jejich pořadí v programu a současně jsou přepisovány hodnoty z přejmenovaných registrů do registrů architektury. Architektura procesoru PowerPC 604 se poněkud vymyká procesorům třídy post-RISC implementací instrukce "branch-on-counter"[3], která patří do skupiny instrukcí obvykle implementovaných v procesorech s úplným souborem instrukcí. Tato instrukce však umožňuje velmi efektivní provádění čítaných cyklů v programu a jsou pro ni vytvořeny i předpoklady v organizaci procesoru, např. je akceptována při predikci skoku.

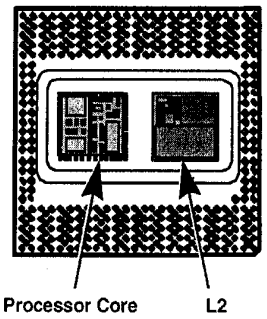
6.6.6.3 Intel Pentium P6

Procesor Intel Pentium P6 je zajímavou směsí různých architektur. Z obavy, aby nebyla narušena instrukční sada původních úspěšných procesorů x86, a tím ohrožen trh, implementuje úplnou instrukční sadu x86 na procesorovém jádře, což není nic jiného než výkonný procesor třídy post-RISC. Vnitřní struktura procesoru Pentium P6 je navržena způsobem velmi blízkým modelu procesoru třídy post-RISC, který byl popsán v kapitole 6.6.4, jako třícestný superskalární procesor. Nejzajímavější na této směsici architektur je, že vlastní procesor nevykonává přímo instrukční sadu x86, ale interpretuje je pomocí skutečných instrukcí post-RISC procesoru nazývané v terminologii Pentium P6 "mikrooperace". Každá instrukce cílového stroje x86 je rozložena a buď nahrazena jednou mikroinstrukcí nebo je interpretována sekvencí několika mikrooperací, které jsou následovně prováděny spekulativním způsobem se všemi aspekty organizace post-RISC.

Z hlediska implementace procesoru je konstrukčně zajímavé umístění vyrovnávací paměti druhé úrovně tzv. L2 cache, do stejného pouzdra s procesorem, i když na jiném čipu (viz obr. 74). Tento způsob sestavení procesoru a vyrovnávací paměti umožňuje, aby vyrovnávací paměť pracovala se stejným hodinovým kmitočtem jako procesor, a tak bylo docíleno vyšší prostupnosti mezi procesorem a touto vyrovnávací pamětí. Navíc výměna dat mezi těmito dvěma prvky probíhá po zvláštní sběrnici, a tak je základní sběrnice uvolněna pro nekolizní spolupráci s hlavní pamětí a zařízeními vstupu a výstupu.



Obr. 73 Třícestný model procesoru Pentium



Obr. 74 Uspřádání čipů v pouzdře procesoru Pentium

Instrukce v procesoru Pentium P6 probíhají třemi základními stupni, které si můžeme představit jako tři nezávislé jednotky vzájemně propojené společným zásobníkem instrukcí (viz obr. 73). V prvním je instrukce vyzvednuta a dekódována (Fetch and Decode Unit), druhý stupeň zajišťuje spekulativní provedení mikrooperace (Dispatch/Execute Unit) a třetí stupeň sestavuje instrukce do původní sekvence (Retire Unit). S ohledem na složitost instrukční sady je dekódování instrukce na rozdíl od ostatních procesorů třídy post-RISC velmi složité tak podstatnou část linky proudového zpracování zabírá jednotka pro vyzvednutí a dekódování instrukce. I když průměrný počet mikrooperací, kterými jsou interpretovány instrukce z instrukční sady x86 je menší než tři některé složitější instrukce generují až stovky mikrooperací. Jednotka pro vyzvednutí a dekódování instrukc

---

[1] Completed Instruction Buffer

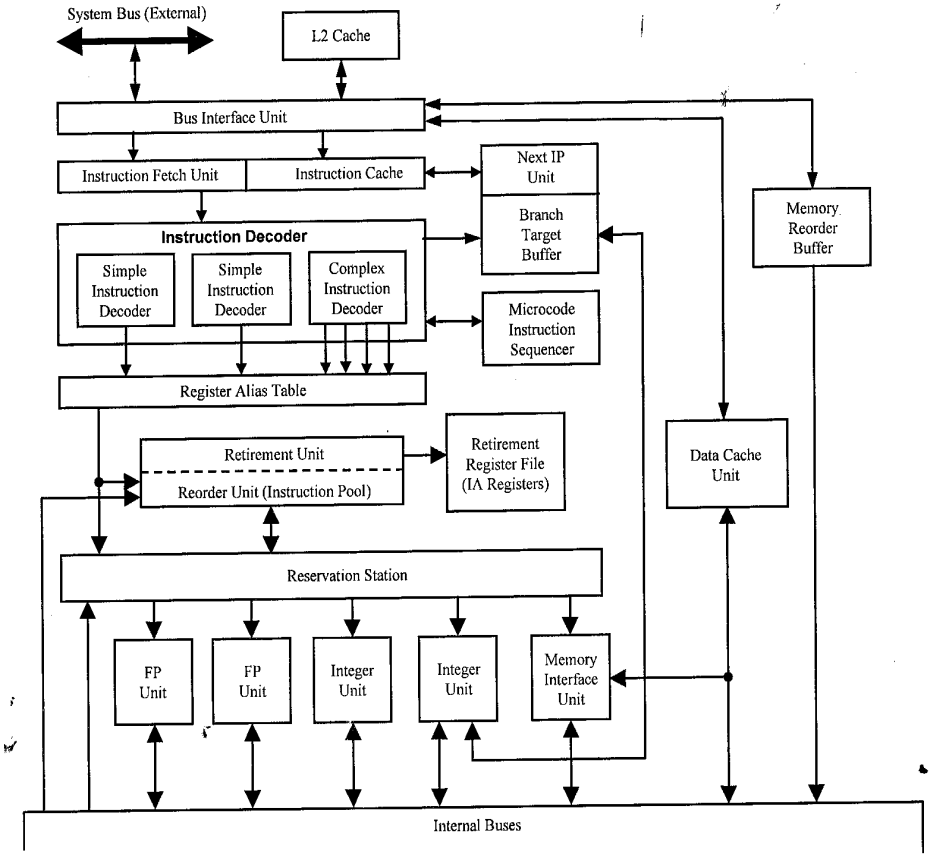
[2] Completion Unit

[3] Větvení podle obsahu čítače.

zpracovává instrukce v pořadí, v jakém byly předloženy ke zpracování procesoru. Popišme nyní některé prvky architektury podrobněji.

Jednotka vyzdvižení a dekódování instrukcí (Fetch/Decode)

Z paměti instrukcí je vyzvednuta odpovídající skupina instrukcí podle toho, jak je nastaven ukazatel další instrukce. Jeho hodnota je výsledkem zpracování vstupu z vyrovnávací paměti cílových adres (Branch Target Buffer), výsledku vyhodnocení přerušení nebo "trapu" a indikace kvality predikce z exekuční jednotky. Vyzvednutá skupina 16 bytů je předložena k dekódování následujícímu stupni.



Obr. 75 Vnitřní bloková struktura procesoru Intel Pentium P6

Paralelní dekodéry konvertují instrukce architektury x86 do tříadresových mikrooperací (jedna adresa pro uložení výsledku, dvě adresy pro vstupní operandy). Instrukce jsou převáděny do jedné až čtyř mikrooperací. Složitější instrukce jsou předloženy ke zpracování mikrooperačnímu sekvenceru, který vygeneruje předprogramovanou sekvenci mikrooperací. Takto zpracované instrukce, nyní již ve formě mikrooperací, jsou předány do jednotky přidělování přejmenovaných registrů (Register Alias Table), kde reference vztahující se k registrům architektury jsou převedeny na přejmenované – fyzické registry pro mikrooperace. Alokátor v následujícím stupni přiřadí každé mikrooperaci stavovou informaci, a takto upravenou mikrooperaci předá do vyrovnávací paměti (tzv. "Reorder Buffer"), která odděluje tuto jednotku od přidělovací a exekuční jednotky.

Přidělovací jednotka vybírá mikrooperace z vyrovnávací paměti podle jejich stavové informace. Jestliže ta stavová informace indikuje, že mikrooperace má k dispozici všechny potřebné operandy, je uložena rezervační stanice a přidělovací jednotka následovně zjišťuje, zda jsou k dispozici další zdroje pro zpracová mikrooperace (např. volné funkční jednotky). Pokud jsou jak operandy tak další zdroje k dispozici, mikroopera je předána z rezervační stanice příslušné exekuční jednotce, kde je zpracována. K rezervačním stanicím js exekuční jednotky připojeny prostřednictvím pěti bran a přidělovací mechanismus má na výběr z exekuční jednotek pro operace typu **`load/store`**, jednotek pro práce s čísly v pohyblivé řádové čárce i jednotek p práci s celými čísly nebo speciálních jednotek pro zpracování mikrooperací skoku (Jump Execution Unit) specifických mikrooperací reprezentujících instrukční podmnožinu MMX[1].

Odkládací jednotka (Retire Unit)

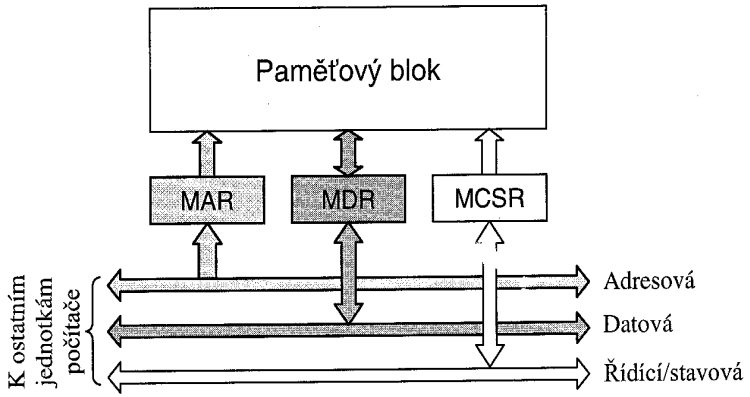
Mikrooperace provedená exekuční jednotkou mimo pořadí je předána do vyrovnávací paměti odkládá jednotky, kde vyčkává, než dorazí mikrooperace, které měly být vykonány podle pořadí programových krol dříve. Ve chvíli, kdy k tomu dojde, je sada mikrooperací odložena a přejmenované registry jsou přepsány registrů architektury. Poměrně dlouhá linka proudového zpracování dat, 12 stupňů, má jednotlivé stup zjednodušeny, takže potřebný čas na jeden stupeň je asi o 30% kratší než u procesoru Pentium. To umožní zvýšení hodinového kmitočtu u tohoto typu procesoru.

[1] **M**ulti**M**edia e**X**tension - rozšíření instrukční sady o speciální instrukce umožňující rychlejší práci v multimediálních aplikacích; poprvé použito firmou Intel

7 Paměti

nezapomeňte poznamenat kam.
A. Bloch: "Murphy's Law"

Paměť slouží k ukládání veškerých dat a programů, používaných procesorem a ostatními součástmi systému. Ve struktuře počítače tvoří neoddělitelnou složku, obvykle propojenou nejen s procesorem, ale i s periferními zařízeními počítače. V mnoha případech jsou malé paměťové oblasti přímo součástí procesoru v jeho návrhu, nebo jsou technologicky integrovány s procesorem na jednom čipu či jedné desce. Vysoká rychlost výměny dat mezi procesorem a pamětí, vyvolaná požadavkem na maximální výkon systému, vede k minimalizaci doby potřebné k předání dat ostatním součástem systému, a tak rychlost paměti a její umístění výrazně ovlivňuje výsledné charakteristiky celého systému.



Obr. 76 Model paměťové jednotky počítače

Paměť je s ostatními součástmi systému propojena pomocí sběrnic, které slouží k zajištění přenosu adresových a datových položek. Z hlediska organizace je možno na paměťový blok nahlížet jako na jednotku systému (viz obr. 76), která je s ostatními jednotkami propojena pomocí dvou bran, a tyto brány jsou připojeny ke dvojici registrů – adresovému registru paměti (**MAR** – **M**emory **A**ddress **R**egister) a datovému registru paměti (**MDR** – **M**emory **D**ata **R**egister). Tato jednotka se chová tak, že pokud se do registru MAR zapíše hodnota odpovídající adrese v paměti, pak v registru MDR se objeví hodnota uložená na této adrese, nebo se hodnota uložená v registru MDR zapíše na výše uvedenou adresu. Pro stanovení způsobu práce paměti je zapotřebí dalších řídících signálů, a proto se model paměťové jednotky doplňuje obvykle třetím registrem – řídícím registrem paměti (**MCSR** – **M**emory **C**ontrol/**S**tatus **R**egister). Tento registr akceptuje potřebné signály, které specifikují způsob práce paměti v daném cyklu (čtení nebo zápis) a s očekáváním většího počtu paměťových jednotek v systému i požadavek na výběr příslušné paměťové jednotky. V některých případech je funkce registru MCSR rozšířena o ukládání stavové informace, a tak lze jeho prostřednictvím předávat potřebné informace o postupu operace paměťové jednotky, což je důležité pro synchronizaci práce s ostatními jednotkami počítače nebo pro rozhodnutí o dalším postupu operace s pamětí[1].

7.1 Charakteristiky paměti

Můžeme ji charakterizovat třemi základními parametry - **funkcí, kapacitou a časovou odezvou**. Základní operace paměti nazývané **čtení** a **zápis** přenášejí obsah z paměti nebo do paměti na základě požadavků vznikajících mimo vlastní paměť. Přitom paměť sama může umožňovat pouze operaci čtení jejího obsahu. Nejmenší jednotka informace je **1 bit**, který může nabývat dvou různých hodnot. **Kapacita** jednotlivých paměťových obvodů je často udávána jako počet bitů. Při měření celkové kapacity systému je preferována větší

[1] Např. u tzv. asociativních pamětí se pomocí stavového údaje, obsaženého ve zvláštním stavovém registru předává informace o tom, zda hledaná položka byla nalezena či nikoliv.

desítek či stovek gigabytů.

Výkon paměťového systému může být vyjádřen pomocí dvou různých měřítek - doby přístupu a doby cyklu paměti. **Vybavovací doba**, která určuje dobu přístupu k datům, je také známa jako **latence** nebo **doba odezvy**[1] a udává, jak rychle paměť bude reagovat na požadavek čtení nebo zápisu. Požadavek na krátkou vybavovací dobu vzrůstá s tím, čím blíže je paměť umístěna ku procesoru. **Cyklus paměti** odpovídá minimální periodě mezi dvěma následujícími požadavky. Je obvykle delší než doba přístupu a zahrnuje nezbytné operace paměti potřebné k přístupu k jednotce uložených dat a časování fyzikálních procesů, které jsou s touto operací spojeny.

Všechny výše uvedené charakteristiky udávají základní parametry paměťové jednotky a jsou jedním ze základních parametrů počítačového systému. Údaje o latenci paměti a o cyklu paměti jsou do značné míry závislé na formě implementace paměti a na způsobu jakým je k datům přistupováno

7.1.1 Rozlišení pamětí podle přístupu k datům

Nejběžněji používanou pamětí je typ umožňující čtení uložené informace z libovolné adresy v kterýkoli okamžik, tedy paměť s náhodným přístupem (**RAM** – **R**andom **A**ccess **M**emory). Doba odezvy u paměti RAM je konstantní a nezávislá na adrese, která je použita pro čtení nebo uložení potřebné informace, tedy platí podmínky

$$\langle příští\ adresa \rangle = \langle libovolná \rangle$$

$$t_a = \langle konst \rangle$$

Rovněž následující adresa, která bude použita nezávisí na předchozí adrese a jejich vztah neovlivní dobu odezvy paměti. Příkladem těchto pamětí mohou být čipy používané v hlavních pamětech personálních počítačů.

Na rozdíl od paměti RAM, paměti se sériovým přístupem (**SAM** – **S**erial **A**ccess **M**emory) vykazují výraznou závislost na lokalizaci dat v paměťové jednotce. Při čtení nebo zápisu musí být u paměti se sériovým přístupem splněna podmínka

$$\langle příští\ adresa \rangle = \langle současná\ adresa \rangle + 1$$

a doba odezvy paměti se sériovým přístupem je

$$t_a = \langle konst \rangle * \langle vzdálenost\ adres \rangle$$

Tyto paměti nejsou pouze historické – např. magnetické bubny nebo ultrazvukové zpožďovací linky, ale v polovodičovém provedení jsou takové integrované obvody součástí i moderních počítačů, zejména rastrových displejů nebo podobných zařízení, kde způsob získávání informací má výrazně sériový charakter.

Pokud budeme k datům přistupovat přímo, avšak čas potřebný k přístupu bude složitější funkcí vzdálenosti adres, označujeme tyto paměti jako paměti s přímým přístupem (**DAM** - Direct Access Memory) a platí

$$\langle příští\ adresa \rangle = \langle libovolná \rangle$$

$$t_a = f(vzdálenost\ adres)$$

Příkladem paměti s přímým přístupem může být disková paměť, používající k záznamu informace rotující magnetické kotouče. Již intuitivně je zřejmé, že přístupová doba bude složitou funkcí okamžitého vydání požadavku na přístup k datům, okamžité polohy kotouče vůči hlavě a vzdálenosti dat od sebe.

Zvláštním případem bude situace, kde základním parametrem přístupu při čtení nebude adresa místa, kde je uložena informace, ale vlastní položka dat a úkolem paměťové jednotky bude zjistit, zda takovou datovou položku obsahuje, a kde je uložena. Tento druh paměti, kde se vyhledává podle obsahu paměťového místa a výsledkem hledání je adresa na níž je hledaný údaj uložen, nazýváme **asociativní pamětí**, která se též označuje

[1] Anglický termín "access time" vyjadřuje tutéž veličinu.

jako paměť adresovaná obsahem (**CAM** – Content Addressable Memory). Základní její charakteristikou je konstantní doba vyhledání položky v paměti. Jednotlivé datové položky v asociativní paměti jsou doprovázeny dodatečnými bity, které určují zda příslušná položka – slovo, obsahuje relevantní informaci (tzv. *empty bit*) a zda má být zahrnuta do procesu vyhledávání (tzv. *skip bit*). Pro případ, kdy se vyskytne více stejných datových položek v asociativní paměti, je vyhledání spojeno se stavovým údajem, který na tuto skutečnost upozorní[1]. Pro poměrně obtížnou konstrukci asociativních pamětí se někdy používají paměti s omezenou asociativitou, jejichž návrh spočívá v rozdělení ukládaných informací do bloků podle nějakého společného znaku, např. stejného řetězce bitů v části datové položky. Práce uvnitř těchto bloků je pak již plně asociativní. Tento druh pamětí se používá všude tam, kde je zapotřebí vyhledávat umístění předmětné informace, jako např. pro mapovací tabulky virtuálních pamětí (viz strana 143)

7.1.2 Rozlišení pamětí podle implementace

Rozlišení podle způsobu, jak jsou paměti implementovány je velmi blízké předcházejícímu, avšak je postaveno blíže oblasti implementace než organizace. V první řadě se jedná o paměti typu RAM, které je však nutno v tomto případě chápat ve smyslu rychlého přístupu k dané adrese pro operaci zápisu i čtení, tedy spíše jako paměti pro čtení i zápis na libovolnou adresu - **RWM** (**R**ead/**W**rite **M**emory). Paměti označované ROM (Read Only Memory) odpovídají rovněž charakteristice pamětí RAM, avšak umožňují pouze čtení již uložené informace. Je zde tedy nejednoznačnost v názvosloví, avšak pro účely této kapitoly se přikloníme k označování s ohledem na technologické charakteristiky a předpokládejme dvě základní skupiny pamětí RAM a ROM.

Paměti typu RAM obvykle rozdělujeme do dvou základních skupin – statické paměti označované **SRAM** (Static **R**andom **A**ccess **M**emory) a dynamické paměti **DRAM** (**D**ynamic **R**andom **A**ccess **M**emory). **Statické paměti** se skládají z polí klopných obvodů s přidruženou adresovací logikou složenou z dekodérů a multiplexorů a mohou jednou uloženou informaci uchovat libovolně dlouho, pokud je jim dodáváno napájení. Poměrně složitý návrh jedné buňky paměti SRAM vede k jejich vysoké ceně a poměrně malým kapacitám[2]. Tato nevýhoda byla odstraněna u **dynamických pamětí**, kde paměťový element je reprezentován kondenzátorem, vytvořeným ve struktuře integrovaného obvodu. Jestliže náboj, který tento kondenzátor získá při přiložení elektrického napětí je úměrný vkládané informaci – např. 0 odpovídá nulovému náboji a 1 nejvyššímu náboji[3], pak přidáním adresovacích a nabíjecích obvodů získáme paměťovou buňku dynamické paměti. Její realizace je mnohem jednodušší než buňky statické paměti, a tedy i hustota paměťových elementů může být větší. Zásadní nedostatek takové buňky však spočívá v tom, že kondenzátor po čase svůj náboj ztratí, a proto je nutno zajistit, aby pomocí dalších obvodů bylo zajišťováno pravidelné obnovování informace[4], které spočívá ve čtení a opětovném zápisu do stejné buňky. Tento postup, který je nutno u dynamických pamětí opakovat v intervalech několika desítek milisekund, vede k tomu, že dynamické paměti jsou obecně pomalejší – mají delší vybavovací dobu než paměti statické.

Rozvoj paměťových systémů však vyžadoval stále větší paměti a stále rychlejší přístup při akceptovatelných cenách. To vedlo konstruktéry k úpravám v oblasti technologií výroby pamětí DRAM, které se staly nejobecnějším paměťovým prvkem v hlavních pamětech současných počítačů. Nejčastěji se setkáme s několika technologickými variantami, které slouží ke zrychlení přístupu k uloženým informacím a mezi které patří:

- dynamické paměti s rychlým stránkovým přístupem (Fast Page Mode DRAM), které zkracují potřebný čas zjednodušením přístupového mechanismu. V pamětech DRAM se používá dvoufázový přístup k datům uloženým v matici adresovatelné řádky a sloupci. Nejdříve je nutno nastavit adresu řádku, a pak adresu sloupce matice, kde jsou data uložena. Paměti DRAM s rychlým stránkovým přístupem však nastaví adresu řádku a pak umožní přečíst několik libovolně umístěných sloupců v této řádce – stránek dat. Jejich použití je zejména při konstrukci větších sestav těchto pamětí, kde umožní přečíst

[1] Řešení této situace je závislé na návrhu paměti. Obvykle je zprostředkována pouze informace o tom, že došlo k několikanásobnému výskytu stejných dat a předána krajní (nejvyšší či nejnižší) adresa datové položky. Další akce jsou přenechány uživatelskému programu.

[2] Tedy i k vysokému poměru "cena/výkon", který je základním měřítkem v počítačových systémech.

[3] Velikost náboje je úměrná kapacitě kondenzátoru a přiloženému napětí.

[4] Tento postup se nazývá občerstvením paměti – anglický termín "refresh".

najednou několik slov během jednoho přístupu k paměťovému bloku (tzv. burst transfer – burst přenos).

- dynamické paměti s potlačeným výstupem dat (EDO – "extended data out") používají dodatečného signálu (Output Enable), který potlačí data na výstupu paměti v okamžiku, kdy již nejsou potřeba. Tak se zkrátí doba, kdy je obsazeno propojení mezi pamětí a požadující jednotkou a je možno zrychlit práci celé sestavy.

- synchronní dynamické paměti (synchronous DRAM) jsou optimalizovány pro blokové operace. Obvykle umožňují současný přístup k několika řádkům paměti současně – prokládání paměti na čipu a jejich řízení je již velmi dokonalé, včetně autonomních obnovovacích mechanismů. Tento typ paměti je již přechodem k samostatným komunikujícím paměťovým jednotkám.

Další varianty paměťových modulů nebo čipů jsou uživatelsky orientovány na konkrétní použití. Např. tzv. VideoRAM je vybavena posuvným registrem na výstupu a umožňuje paralelní zápis dat a jejich sériové čtení, což je výhodné pro zobrazovací jednotky, které používají časový rozklad vícerozměrného obrazu. Výrobci uvádějí na trh i různé modifikace pamětí doplněné např. zvláštním jednoúčelovým procesorem ke každému čipu DRAM, který zajišťuje "inteligentní zpracování požadavku na výběr dat – **IRAM** (Inteligent RAM).

Pro počítačový systém je však nutno zajistit, aby v něm byla obsažena nějaká pevná, zárodečná informace, jejíž obsah by byl stálý a nepotřeboval by ke svému udržení vnější zdroj energie, který je pro výše uvedené typy pamětí nezbytný. Takové pevné paměti, jejichž obsah je určen jenom ke čtení, obecně nazýváme **ROM** (**R**ead **O**nly **M**emory) a rozlišujeme mezi

- pevnou pamětí ROM, do níž jsou uložena požadovaná data již při výrobě a nemohou být nikdy změněna. Technologie uložení této informace se různí, nejčastěji se jedná o použití specifických "masek" při výrobě integrovaného obvodu, tzv. "masked ROM". Cena těchto obvodů je relativně malá, pokud jsou vyráběny ve velkém množství.

- jednorázově programovatelné paměti **PROM** (**P**rogrammable **R**ead **O**nly **M**emory), které se mnohdy označují též jako **OTP ROM** (**O**ne **T**ime **P**rogrammable **R**ead **O**nly **M**emory). Tyto paměti se chovají podobně jako pevné paměti ROM, které obsahují ve všech bitech jedničky. Pomocí zvláštního zařízení, programátoru, se elektrickým impulsem odstraní na požadovaných adresách tzv. pojistky, čímž se hodnota odpovídajícího bitu změní na nulu. Do pamětí PROM je tak možno uložit požadovanou informaci, která zůstane zachována i při přerušení napájení a bude vnějším jednotkám kdykoli dostupná. Uloženou informaci již není možno změnit.

- přepisovatelné paměti ROM, do kterých může být informace uložena, poté vymazána a nahrazena jinou. Způsobů implementace těchto pamětí je celá řada a nejčastěji se setkáme s typy, které pro vymazání informace je nutno ozářit ultrafialovým světlem (EPROM) nebo je možno použít zvláštní kombinace elektrických impulsů na vstupech obvodu (EEPROM, FlashROM apod.).

Důležitým parametrem paměťových bloků je jeho **kapacita**, tedy objem dat, který je schopen paměťový blok pojmout, a **šířka pásma** paměti. Ta vychází z doby přístupu paměťového elementu k datům, která je dána cyklem paměti (viz strana 118), a šířky přenosové cesty - sběrnice. Udává se obvykle v MB/s a představuje buď špičkovou nebo zaručenou rychlost předávání dat z paměťového bloku. Špičková šířka pásma reprezentuje jedno slovo za jeden cyklus, čili může být snadno vyčíslena ze šířky sběrnice, kterou paměť poskytuje a cyklu paměti. Zaručená šířka pásma je ovlivněna konstrukcí a složitostí paměťového systému a její vyčíslení je složitější.

Příklad

Paměť zbudovaná z prvků DRAM s rychlým stránkovým přístupem má šířku datové sběrnice 64 bitů, dobu cyklu pro čtení jednoho slova (nastavení řádku a prvního sloupce paměti) 110 ns a dobu cyklu pro další slova (mění se jenom nastavení sloupce) 40 ns. Šířka pásma pro jedno slovo bude

$$B = \frac{8\ bytů}{110\ ns} = 69{,}35\ [MB/s]$$

šířka pásma pro dvě slova

$$B = \frac{16\ bytů}{(110+40)\ ns} = 101{,}73\ [MB/s]$$

Bude-li paměť umožňovat přečíst maximálně 256 slov (sloupců) na jedno nastavení řádku pak můžeme obdržet obvykle udávané údaje pro špičkovou šířku pásma

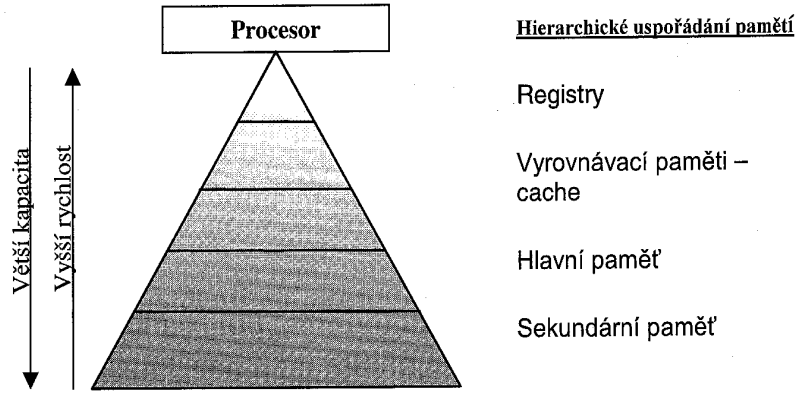
$$B = \frac{8\ bytů}{40\ ns} = 190{,}73\ [MB/s]$$

a pro zaručenou šířku pásma

$$B = \frac{256\ slov * 8\ bytů}{110\ ns + 256 * 40\ ns} = 188{,}71\ [MB/s]$$

7.1.3 Rozlišení pamětí podle umístění v systému

Počítačový systém při své práci vyžaduje uložení celé řady informací, některé pouze na nepatrný zlomek sekundy, jiné trvale. Z hlediska organizace systému bude tedy výhodné vytvořit hierarchii pamětí, kde k uložení různých typů informací budou použity různé úrovně hierarchie. Nejrychlejší, ale zároveň i nejdražší paměti budou umístěny co nejblíže procesoru a vzhledem k jejich ceně budou mít co nejmenší možný objem. Se vzdalováním od procesoru bude narůstat objem realizované paměti tak, jak bude klesat její poměrná cena, ale zároveň se bude snižovat i její rychlost. Jednotlivé stupně hierarchie paměti budou realizovány na různých technologiích a měřítkem kvality jsou parametry související s cenou za jeden uložený bit a s rychlostí paměti.



Obr. 77 Hierarchie pamětí v paměťovém systému

Obvykle používáme základní dělení na **vnitřní paměti**, kam patří

- **registry** s vysokou rychlostí přístupu jak pro čtení tak i pro zápis, většinou realizované přímo jako součást procesoru. Počet registrů není vysoký a pohybuje se od desítek do několika stovek paměťových míst. Uvnitř procesoru vytváří registrové pole, které se někdy označuje jako zápisníková paměť[1].Vybavovací doba těchto pamětí je řádu jednotek až desítek ns,

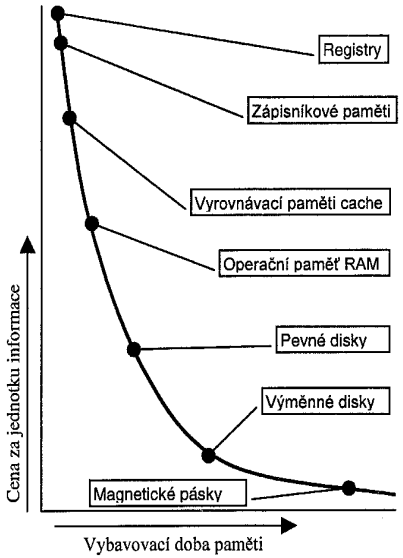
- paměť, která je nejblíže k procesoru je známa jako **vyrovnávací paměť** neboli **cache**. Některé systémy mají zvláštní paměť cache pro instrukce a jinou paměť cache pro data, a vždy se umisťují co nejblíže k procesoru, aby se eliminovalo zpoždění signálu ve vodičích. Vybavovací doby vyrovnávacích pamětí jsou desítky ns,

[1] Anglický název "scratchpad store".

- **hlavní paměť**, která tvoří vlastní fyzickou paměť počítače ve které jsou uložena data a instrukce prováděných programů. Její kapacita dosahuje řádu desítek MB až několika GB a vybavovací doba je řádu několika desítek až stovek ns,

a **vnější paměti**, jejichž označení vychází z fyzického oddělení od procesoru a které zahrnují nejenom typické paměťové bloky, ale i vnější paměťová zařízení. Mezi vnější paměti řadíme

- **stránkovací paměti**, které slouží pro ukládání stránek (bloků dat) z hlavní paměti, pro které už v hlavní paměti není místo, avšak předpokládá se, že budou v krátké době opět potřeba. Tento typ pamětí býval realizován diskovou mechanikou s krátkým přístupovým časem (několik ms) nebo klasickou pamětí RAM s vybavovací dobou stovek mikrosekund[1],
- **sekundární paměti**, kam zahrnujeme vnější paměti s náhodným přístupem, tedy magnetické disky, disková pole nebo i diskety. Vyznačují se vysokým objemem uchovávané informace při relativně rychlém přístupu k datům,
- **archivní paměti**, které se používají pro trvalé uchování velkých objemů dat, ke kterým se přistupuje pouze zřídka. Mezi tyto paměti patří především magnetické pásky, ať už klasického typu s podélným záznamem informace a šířkou půl palce nebo kazety používající diagonální záznam rotačními hlavami podobně jako u záznamu televizního signálu. Optické disky a optická média vůbec představují samostatnou rychle se rozvíjející oblast archivních pamětí a rozmach používání optických disků pro distribuci programů a dat je důkazem vitality tohoto typu záznamu. Přechod na holografické typy záznamu do optického média umožní ještě daleko vyšší hustotu záznamu v jednotce objemu.



Obr. 78 Vztah ceny za uloženou jednotku informace a rychlosti přístupu k informaci

Určujícím prvkem při hierarchickém návrhu paměťového systému bude jeho konečná výkonnost a svoji roli zde sehrají nejenom objemy a rychlosti pamětí, ale i způsob jejich propojení, úroveň vzájemné komunikace a další faktory, které ovlivňují četnost této komunikace. K těmto faktorům přispívá i organizace paměťové jednotky na příslušné hierarchické úrovni, způsob jakým je navržena, aby byly eliminovány nebo alespoň sníženy ztráty způsobené sčítáním latencí jednotlivých bloků a přenosovými charakteristikami propojovacích cest.

7.2 Vnitřní organizace paměťových jednotek

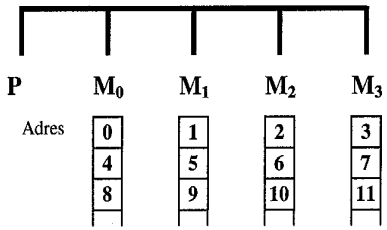
Vysoký výkon paměťového systému, který je dán jeho organizací a který poskytuje potřebné zdroje procesorové jednotce souvisí úzce i s organizací výstavby jednotlivých bloků uvnitř paměťových jednotek. Při posuzování výkonu počítačového systému nezáleží ani tak na tom, jak rychle bude pracovat procesorová jednotka, pokud paměťový systém nebude schopen dodat včas instrukce a data a výkonný procesor bude neustále na data čekat. Přitom hlavní problém, který je nutno vyřešit při přizpůsobování rychlosti paměti a procesoru, je čas mezi dvěma následujícími operacemi paměti, tedy při sekvenčním zpracování programu nejčastější variantou operace paměťové jednotky. Obecně jsou pracovní cykly procesoru typicky kratší než pracovní cykly paměti. Jestliže procesor inicializuje přenos z paměti v čase t_0, pak paměťový systém bude aktivní

[1] V hierarchické pyramidě pamětí není tento typ uveden a nalézal by se mezi hlavní pamětí a sekundární pamětí.

až do času $t_0 + t_c$, kde t_c označuje jeho cyklus. Po tuto periodu nebude paměťova jednotka, resp. její dotcen blok dostupný žádnému jinému zařízení a rovněž procesor, který inicializoval požadavek, nebude moci tuto čás paměti využít, neboť ta bude zpracovávat právě jeho požadavek. Technologické omezení prvků z nichž jso bloky pamětí sestavovány vedlo k hledání metod organizace, které by umožnilo z relativně pomalých prvk sestavit dostatečně rychlou paměť.

7.2.1 Paměť se širokým slovem

Jak již sám název říká, paměťový blok propojený s vnějším obvody počítače používá pro vnitřní přenos větší šířku sběrnice. Do datového registru paměti se předává najednou několikanásobek potřebné informace z tohoto registru je postupně odebírán spolupracujícími obvody. Bude-li paměť spolupracovat např s procesorem požadujícím operace s 32 bitovým slovem a její blok bude používat pro spojení s datovým registrem sběrnici se šířkou 128 bitů, pak na jedno naplnění registru je možno počítat s dobou čtyřnásobně delš než je cyklus procesoru. Tento způsob organizace však předpokládá, že bude docházet ke čtení nebo zápisu vždy na adresách, které za sebou následují. V opačném případě bude procesor pozastaven na dobu nutnou k provedení cyklu paměti.



Obr. 79 PMS diagram paměti s prokládanými cykly pro čtyři bloky

7.2.2 Paměť s prokládanými cykly

Paměti s prokládanými cykly[1] využívají rozdělení paměti na moduly, nazývané **"bank"** a možnosti přístupu procesoru do jednoho modulu, zatímco druhý modul zpracovává předcházející požadavek na přístup do paměti Paměť, která obsahuje n modulů, se nazývá n-cestnou pamětí s prokládanými cykly. Jednou z možných ces alokace adresy pro paměťový modul je rozdělení paměťového prostoru, tedy množiny všech možných adres které procesor může generovat, do spojitých bloků. Použijeme-li n bloků, pak slovo s adresou i bude ležet v paměťovém modulu $i \bmod n$.

Příklad

Předpokládejme, že existují čtyři bloky, každý z nich obsahující 256 bytů. Kdybychom použili rozdělení paměti do lineárně adresovatelných bloků, pak první adresy 0 - 255 by ležely v prvním bloku, 256 - 511 v druhém atd. Při použití technologie pamětí s prokládanými cykly se přiřadí adresy 0, 4, 8 atd. do prvního bloku, adresy 1, 5, 9 do druhého bloku atd. (viz obr. 79).

Tímto rozdělením a odpovídajícím hardware zajistíme, že procesor může vznášet požadavky na operace s jednotlivými bloky paměti nezávisle, a tudíž je paměť může provádět současně. Pro přiblížení předpokládejme že procesor bude požadovat přenos z adresy i v jednom cyklu a v následujícím cyklu přenos informace z adresy j. Jestliže adresy i a j jsou v různých blocích, pak informace bude, za předpokladu, že latence jednotlivých požadavků[2] je v tomto teoretickém případě zanedbatelná, navrácena v následujícím cyklu. To se bude opakovat do té doby, než procesor pošle požadavky na přístup do stejného bloku, tedy využijeme-li předchozího příkladu budou-li i a j např. dělitelné čtyřmi. V tomto případě budou dvě následující operace provedeny v takovém čase

[1] Poprvé byl tento přístup do paměti realizován v počítači IBM 7094 (1962).

[2] Latence značí dobu, po kterou musí procesor čekat předtím, než obdrží požadovanou informaci zpět.

ako kdyby technologie prokládání paměťových cyklů nebyla použita. Nicméně tento způsob organizace paměti možňuje výrazné zvýšení šířky pásma, a pokud přístupová doba k paměti bude srovnatelná s cyklem procesoru, ak paměťový systém zajistí poskytování informace stejnou rychlostí, jak je procesor požaduje, bez ohledu na to, aký je celkový cyklus paměti.

K rozhodnutí, jak alokovat adresy, zda ve spojitých blocích nebo použít technologii prokládání aměťových cyklů, záleží na tom, jaký učiníme předpoklad přístupu k těmto vloženým informacím. Obvykle sou programy kompilovány tak, že instrukce jsou uloženy postupně za sebou, takže existuje velká ravděpodobnost, že procesor vykoná instrukci na adrese i a následovně bude vykonávat instrukci na adrese $+1$. Rovněž kompilátory mohou alokovat jednotlivé složky vektoru do následujících adres, takže operace oužívající celý vektor využije výhody prokládání.

Pokud se místo na instrukce soustředíme na způsob uložení dat a na přístup k těmto datům, pak můžeme nalyzovat elementární příklad, kdy pole 4x4 je uloženo do čtyřcestné paměti s prokládanými cykly. Jestliže ačteme jednotlivé elementy sekvenčně po sloupcích, pak budou uloženy následovně

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|c|} \hline \textbf{M0} & \textbf{M1} & \textbf{M2} & \textbf{M3} \\ \hline a_{11} & a_{21} & a_{31} & a_{41} \\ a_{12} & a_{22} & a_{32} & a_{42} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{14} & a_{24} & a_{34} & a_{44} \\ \hline \end{array}$$

ekvenční přístup ke sloupcům pole bude tedy bezkonfliktní, stejně tak jako k diagonálám. Pokud však oužijeme sekvenční přístupy k řádkům pole, dojde v případě, kdy rozměr pole je dělitelný počtem modulů banků) ke konfliktu na stejném modulu, a doba potřebná ke čtení bude stejná, jako kdybychom princip rokládané paměti nepoužili.

7.2.2.1 Techniky redukce konfliktů

K minimalizaci konfliktů při čtení nebo zápisu polí do buněk prokládaných pamětí se používají jak softwarové echniky tak i speciální hardwarová opatření. Programové opatření vychází ze skutečnosti, že konflikt při čtení ádků nastane jen tehdy, pokud rozměr pole bude dělitelný počtem modulů paměti. Nejjednodušší cestou bude, aby při kompilaci programu kompilátor rozšířil pole tak, aby tato podmínka nebyla splněna. Pole v paměti počítače se promítne do tvaru

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \\ a_{51} & a_{52} & a_{53} & a_{54} \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|c|} \hline \textbf{M0} & \textbf{M1} & \textbf{M2} & \textbf{M3} \\ \hline a_{11} & a_{21} & a_{31} & a_{41} \\ a_{51} & a_{12} & a_{22} & a_{32} \\ a_{42} & a_{52} & a_{13} & a_{23} \\ a_{33} & a_{43} & a_{53} & a_{14} \\ a_{24} & a_{34} & a_{44} & a_{54} \\ \hline \end{array},$$

kde prvky posledního řádku nebudou při výpočtu používány. Potom sice nedojde ke konfliktu při čtení řádků nebo sloupců, avšak sníží se využití paměti a při čtení diagonálních prvků budou zatíženy jen některé moduly. Opatření, která jsou realizována v hardware zahrnují

- **příčné adresování**, kde hardware mapuje adresy do různých paměťových modulů tak, že prvky pole, které původně tvořily řádky budou uspořádány diagonálně. Pole uvedené výše se pro koeficient příčnosti[1] 1 zobrazí

[1] Koeficient příčnosti nebo též příčnost je posun, ke kterému dochází při rotujícím číslování modulů.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|c|} \hline \textbf{M0} & \textbf{M1} & \textbf{M2} & \textbf{M3} \\ \hline a_{11} & a_{21} & a_{31} & a_{41} \\ a_{42} & a_{12} & a_{22} & a_{32} \\ a_{33} & a_{43} & a_{13} & a_{23} \\ a_{24} & a_{34} & a_{44} & a_{14} \\ \hline \end{array}.$$

Budeme-li číslovat adresy ve směru sloupců, tedy $a_{11} \rightarrow 0$, $a_{21} \rightarrow 1$, $a_{31} \rightarrow 2$ ..., pak příslušné transformační rovnice budou

$$\langle Adresa\ slova\ v\ modulu \rangle = \frac{\langle adresa \rangle}{\langle počet\ modulů \rangle}$$

$$\langle Číslo\ modulu \rangle = \left(\frac{\langle adresa \rangle}{\langle počet\ modulů \rangle} * \langle příčnost \rangle + \langle adresa \rangle \right) \textbf{MOD} \langle počet\ modulů \rangle$$

Těmito transformačními rovnicemi dojde k posunutí tak, že např. pro koeficient příčnosti 1 bude vzestupnému adresování slov 0,1,2 ... odpovídat posloupnost umístění slov v modulech 0,0,0,0, 1,1,1,1, 2,2,.... a posloupnost modulů 0,1,2,3, 1,2,3,0, 2,3 ... .

Příklad

Pro pole uvedené výše můžeme zkonstruovat uložení pro příčnost 2 takto

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|c|} \hline \textbf{M0} & \textbf{M1} & \textbf{M2} & \textbf{M3} \\ \hline a_{11} & a_{21} & a_{31} & a_{41} \\ a_{32} & a_{42} & a_{12} & a_{22} \\ a_{13} & a_{23} & a_{33} & a_{43} \\ a_{34} & a_{44} & a_{14} & a_{24} \\ \hline \end{array}$$

- **prvočíselné prokládání**, které se často používá pro prvočíselné počty modulů, které zároveň splňují podmínku, že prvočíslo lze zapsat ve tvaru 2^N+1 nebo 2^N-1. V takových případech se transformace budou řídit vztahy

$$\langle Adresa\ slova\ v\ modulu \rangle = \langle adresa \rangle\ \textbf{MOD}\ \langle počet\ slov\ v\ modulu \rangle$$

$$\langle Číslo\ modulu \rangle = \langle \textbf{adresa} \rangle\ \textbf{MOD}\ \langle počet\ modulů \rangle$$

a prvky pole budou uloženy ve tvaru

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \Rightarrow \begin{array}{|c|c|c|} \hline \textbf{M0} & \textbf{M1} & \textbf{M2} \\ \hline a_{11} & - & a_{13} \\ a_{23} & a_{21} & - \\ - & a_{33} & a_{31} \\ a_{41} & - & a_{43} \\ a_{14} & a_{12} & - \\ - & a_{24} & a_{22} \\ a_{32} & - & a_{34} \\ a_{44} & a_{42} & - \\ \hline \end{array}$$

[illegible] a tak i využití paměti bude nízké. Jestliže počet slov v modulu bude mocninou 2, pak operace vyhledání adresy modulu se zjednoduší na operaci posunu.

Další metody, které jsou pro zrychlení přístupu k blokům dat u pamětí s prokládáním používány jsou většinou modifikacemi uvedených postupů. Sladění cyklu sběrnice, která spojuje paměťový modul s procesorem a cyklu paměti je základním předpokladem výkonnosti paměťového systému.

7.2.3 Kontrola a opravy chyb dat v pamětech

Nároky na spolehlivost operační paměti vyžadují ošetření dat uložených v paměti tak, aby při špatném čtení došlo k signalizaci chyby nebo aby chyba byla automaticky ošetřena. To vyžaduje vložení dodatečné – redundantní informace, která takové odhalení umožní.

Nejjednodušším způsobem řešení je přidání tzv. **parity,** což je redundantní informace využívající vlastnost dvojkového slova, která umožňuje vyjádřit počet jedniček v něm obsažených jako sudé nebo liché číslo (zda je možno z nich vytvořit "páry"). Určujeme-li paritu dvojkového kódového slova $\boldsymbol{a} = (a_1,a_2,...,a_n)$, kde prvky a_i (i = 1, 2, ..., n) nabývají hodnoty 0 nebo 1, lze pro výpočet parity použít vztah[1]

$$p(\boldsymbol{a}) = (((..(a_1 \oplus a_2) \oplus a_3) \oplus ... \oplus a_n)$$

Hodnota parity je tedy také rovna 0 nebo 1 a ke kódovému slovu se připojuje jako kontrolní znak (kontrolní bit, paritní bit). Paritu vypočtenou podle uvedeného vztahu obvykle označujeme jako sudou, protože počet jedniček ve slově je po přidání paritního bitu vždy sudý. Kromě toho se používá též lichá parita, která vznikne inverzí sudé parity. Přidáním paritního bitu ke všem kódovým slovům vzroste **kódová vzdálenost** o jedničku, takže nebyl-li kód původně vůbec zabezpečen, vznikne přidáním paritního bitu kód s minimální kódovou vzdáleností rovnou dvěma, který je schopen detekovat lichý počet chyb v jednom slově.

Poznámka

Kódovou vzdáleností nazýváme číslo, udávající počet míst, v nichž se liší dvě dvojková slova. Např. 0011 a 0111 mají kódovou vzdálenost rovnu jedné, protože se liší pouze v jednom bitu, a to na druhém místě. Na počest autora jedné ze základních prací o kódování se někdy označuje jako Hammingova vzdálenost.

Parita, zabezpečující jedno slovo se často označuje jako příčná parita, a umožňuje detekovat lichý počet chyb v každém slově. Kromě toho se někdy vytváří bloková paritní kontrola tak, že na konec bloku, sestaveného ze slov zabezpečených příčnou paritou se připojuje kontrolní znak, v němž je hodnota každého bitu vypočtena jako parita všech bitů stejné váhy z předcházejících slov. Tato parita, označovaná jako **podélná parita**, umožňuje v kombinaci s příčnou paritou opravu jedné chyby v bloku a detekci všech dvojic a trojic chyb.

Pro obecný kód, v němž číslice a_i (i = 1, 2,..., n) mohou nabývat hodnot $a_i = 0, 1,..., r - 1$, je parita definována vztahem

Chyba! Objekty nemohou být vytvořeny úpravami kódů polí.

Pro výpočet parity se obvykle volí m = r, takže výpočtem parity vznikne jedna nová číslice. Zabezpečení pomocí desítkové parity bylo použito v hlavní paměti prvního československého elektronického počítače EPOS 1, kde ke dvanácti významovým desítkovým číslicím byla vypočtena třináctá kontrolní číslice jako desítkový doplněk - parita.

Příklad:

Pro číslo 153708900515 je desítková parita [44] mod 10 = 4 a kontrolní číslice je 6.

Samočinná oprava chyb však s sebou vždy nese nebezpečí nesprávné opravy, tj. opravy shluku chyb, mylně považovaného za jednu chybu, na nesprávnou hodnotu. Zamezit tomuto nežádoucímu jevu lze pouze vhodnou volbou kódové vzdálenosti bezpečnostního kódu. Kód musí být zvolen na základě důkladné poruchové analýzy, tj. na základě možností výskytu všech jednotlivých i vícenásobných poruch v libovolném místě paměťového bloku a jejich možných důsledků z hlediska chyb ve čtené informaci. Podle reakce na chybu, je každý samoopravný kód charakterizován dvěma parametry

- maximálním počtem chyb v jednom slově, které je kód schopen objevit a opravit,
- maximálním počtem chyb v jednom slově, které je kód schopen spolehlivě detekovat a hlásit jako neopravitelné.

Výsledkem poruchové analýzy má být taková volba hodnot parametrů, která zajišťuje správnou opravu často se vyskytujících chyb a zároveň vylučuje jejich záměnu s neopravitelnými chybami.

Klasickým příkladem samoopravného kódu, který pracuje s minimální nutnou redundancí a umožňuje poměrně snadné detekování chyb, je **Hammingův kód**. Hammingův kód patří do skupiny lineárních[1] kódů s minimální kódovou vzdáleností $d_{min} = 3$, takže umožňuje samočinnou opravu jedné chyby v kódovém slově. Jako každý lineární kód je i kód Hammingův ekvivalentní kódu systematickému[2], v němž z celkového počtu n bitů ve slově je počet informačních bitů k a počet kontrolních bitů $n - k$ vázán vztahem

$$2^{(n-k)} - 1 \geq n$$

Každý z kontrolních bitů je určen jako parita vhodně zvolené podmnožiny informačních bitů. Hammingův kód je obvykle zadáván kontrolní maticí $(n, n-k)$, která má n sloupců a $n - k$ řádků. Zvolíme-li např. Hammingův kód (7,4) můžeme kontrolní matici navrhnout ve tvaru

$$H = \begin{array}{c} \begin{matrix} a_4 & a_3 & a_2 & a_1 & p_3 & p_2 & p_1 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Tuto matici pak můžeme použít jak k zakódování čtyřbitové informace ($a_4\ a_3\ a_2\ a_1$) tak i ke kontrole správnosti přečteného slova. Zakódování se provádí na základě vzorců symbolicky zapsaných do této matice tak, že jednotlivým řádkům matice odpovídají vztahy :

$$p_3 = a_3 \oplus a_2 \oplus a_1$$
$$p_2 = a_4 \oplus a_2 \oplus a_1$$
$$p_1 = a_4 \oplus a_3 \oplus a_1$$

Příklad

Zakódujme podle předcházející kontrolní matice Hammingova kódu slovo $[a_4\ a_3\ a_2\ a_1]$ = **0011**. Odpovídající hodnoty budou $p_3 = 0$, $p_2 = 0$ a $p_1 = 1$, takže výsledkem kódování bude slovo **0011001**.

Hlavní nevýhodou Hammingova kódu je neschopnost detekovat neopravitelnou chybu, takže se snaží opravit libovolné chybné kódové slovo, bez zřetele na to, jak vzniklo. Pro použití v praxi se zlepšuje detekční schopnost Hammingova kódu, např. přidáním parity odvozené z celého zakódovaného slova včetně paritních bitů. Tak vznikne tzv. rozšířený Hammingův kód, jehož kódová vzdálenost je rovna čtyřem. Tento kód umožňuje jednu chybu ve slově opravit a dvě chyby v jednom slově spolehlivě detekovat jako neopravitelné. Na trojici chyb nereaguje kód jednoznačně, tj. některé chyby detekuje jako neopravitelné, jiné chyby nesprávně opraví. V praxi se Hammingův kód používá v pamětech, kde je výskyt chyb nezávislý, tzn. tam, kde současný výskyt k chyb v jednom slově je přibližně roven k-té mocnině pravděpodobnosti výskytu jedné chyby.

---

[1] Operace ⊕ značí nonekvivalenci (exclusive OR).

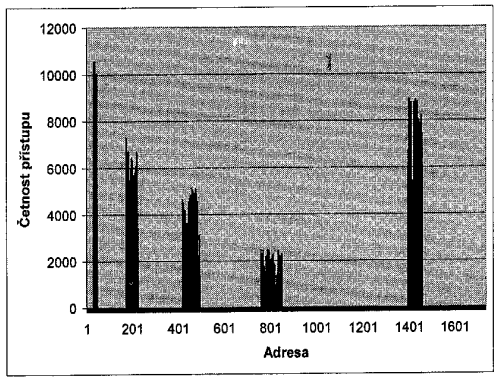
[1] U lineárního kódu tvoří kódová slova lineární podprostor prostoru všech posloupností kódových symbolů délky n.

[2] Kód, v němž lze každé slovo rozdělit na informační a kontrolní část.

7.3 Hierarchické paměťové systémy

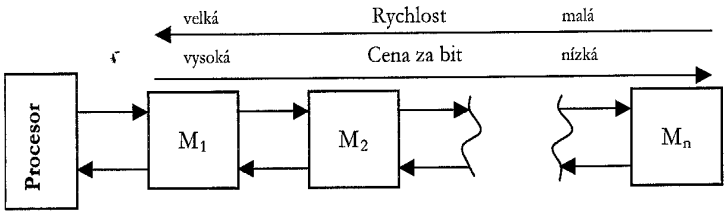
Ačkoliv procesor má volnost ve volbě adresy, na kterou bude v paměti přistupovat, obvykle se její hodnota pohybuje ve vyčleněných oblastech paměťového prostoru. Např. ten fakt, že instrukce jsou ukládány v paměti za sebou[1] je jedním ze zdrojů této regularity. To znamená, že pokud procesor vyžaduje v čase t instrukci z adresy X, je vysoce pravděpodobné, že následující instrukce v nejbližším časovém okamžiku bude požadována z adresy $X+1$. Odkazy na data vykazují podobné chování. Např. jestliže program změní hodnotu jednoho elementu uvnitř pole, pak je velmi pravděpodobné, že následující operace bude měnit další element tohoto pole. Skutečnost, že přístupy do paměti mají tendenci se shlukovat do skupin, se v počítačové architektuře označuje jako **lokalita referencí**.[2]. Rozeznáváme dvě hlediska lokality referencí

- **časovou lokalitu**, tedy že reference, která byla právě použita bude použita v krátké době znovu a
- **prostorovou lokalitu**, která vychází z předpokladu, že pokud byla použita nějaká paměťová reference, pak je nejpravděpodobnější, že následující paměťová reference bude ležet v okolí právě použité reference



Obr. 80 Snímek četnosti přístupů do paměti

Základní myšlenka hierarchického paměťového systému vychází právě z lokality referencí, a tedy z pravděpodobnosti, že pokud budu požadovat data nebo instrukce z adresy X, pak s vysokou pravděpodobností bude následující požadavek směřován na adresy v okolí původního požadavku. Vytvoříme-li krátkodobý snímek práce programu, který bude sledovat četnost přístupu na jednotlivé adresy pak obdržíme graf dle obr. 80, který graficky zachycuje koncentraci přístupů v přesně lokalizovaných oblastech. Z této analýzy vyplývá, že budeme-li přenášet mezi jednotlivými úrovněmi hierarchie větší bloky dat, lze s velkou pravděpodobností předpokládat, že instrukce, následující po instrukci, jejíž požadavek nebyl na nejvyšší úrovni splněn, již plně využije rychlosti malé paměti na vrcholu hierarchie, neboť hledaná položka tam již byla přenesena.



Obr. 81 Hierarchické uspořádání paměťového systému

Uvedená analýze vedla k myšlence automatického dynamického přidělování paměti, která se objevila na počátku šedesátých let. Tato činnost je koordinována operačním systémem počítače a požadavky na využívání operační paměti obvykle vyřizuje správa operační paměti v jádru operačního systému. Rozhodování správy paměti ovlivňuje řada faktorů, mezi než patří např. mechanismus adresace, způsob řízení hardware pamťového

[1] Předpokládejme, že v paměti není uložena instrukce skoku a tedy programový čítač je vždy zvýšen o jedničku v každém instrukčním cyklu.

[2] Lokalita referencí byla a je dosud rovněž využívána při tvorbě tzv. "overlay" segmentů nebo dynamických knihoven, které jsou do paměti zavedeny jen v případě potřeby.

systému nebo organizace sběrnic. Přidělování paměti je založeno na dvou základních principech, které využívaných současně, a to na

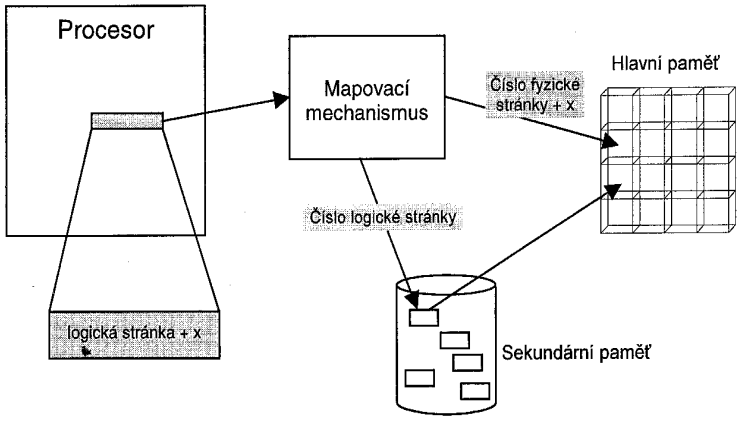
- rozdělení paměti na úseky, tvořené posloupnostmi adresovatelných míst v operační paměti,
- výměnách obsahu úseku hierarchicky nižší paměti s vybranou oblastí hierarchicky vyšší paměti (např úseků hlavní paměti a sekundární paměti).

Obecná struktura hierarchického paměťového systému je na obr. 81. Na vrcholu je umístěna malá paměť s krátkou přístupovou dobou, jejíž cena vztažená k jednomu bitu je vysoká. Do nejnižší části této hierarchie budou umístěny největší, ale také nejlevnější a nejpomalejší paměti. Procesor může přistupovat pouze na vrchol této hierarchie, kde jeho požadavek bude směřovat na nejmenší, avšak nejrychlejší paměť. Jestliže adresovaná položka bude obsažena v této paměti, tato malá paměť velmi rychle reaguje a splní požadavek procesoru. Jestliže se hledaná položka nebude v této paměti nacházet, požadavek se rozšíří do dalších úrovní hierarchie.

Použití kombinace malých, drahých a rychlých paměťových modulů spolu s levnými a objemnými pamětm vedlo k návrhu systémů využívajících spolupráci prostředků hardware paměti a software operačního systému, jenž je často označován jako **virtuální paměti**. Tato technologie umožňuje, že zpracovávaný proces nemusí nutně být celý umístěn v hlavní paměti počítače. Řízení činnosti virtuální paměti je soustředěno do jádra operačního systému a hardware počítače mu poskytuje celou řadu prostředků pro efektivní realizaci mechanismů virtuální paměti. Základním mechanismem je **mapování** neboli způsob, jak je logický paměťový prostor mapován do fyzického adresového prostoru

7.3.1 Mapovací mechanismy

Rozhodnutí o způsobu, jakým se bude zobrazovat větší logický paměťový prostor do menšího fyzického adresového prostoru patří k základním rozhodnutím při návrhu architektury počítače. Přitom zřejmou základní podmínkou je, aby zpracovávaná instrukce byla umístěna v paměti, která je nejblíže procesoru. Jeden ze způsobů, který umožňuje realizaci virtuálního paměťového systému je zjednodušeně uveden na obr. 82. Procesor, který má provádět instrukci, jež se nenachází ve fyzickém adresovém prostoru, informuje o této skutečnosti operační systém např. pomocí přerušení. Operační systém zjistí, kde se stránka s požadovanou instrukcí nachází a zajistí její přenos do volného rámce ve fyzické paměti. Způsoby mapování a výměny oblastí paměti mezi jednotlivými úrovněmi paměťové hierarchie jsou předmětem následujících kapitol.

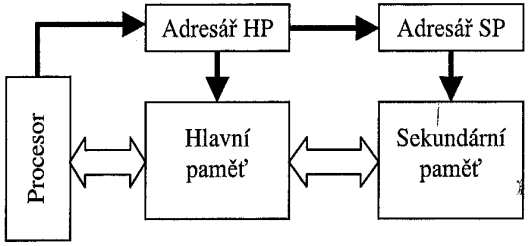


Obr. 82 Příklad mechanismu výměny paměťových oblastí (stránek) v hierarchickém systému

7.3.2 Dvojúrovňový paměťový systém

Nejjednodušší hierarchický paměťový systém, jenž je znázorněn na obr. 83, je dvojúrovňový, . Obsahuje dva typy pamětí - jednu paměť vnitřní, reprezentovanou hlavní pamětí a jednu vnější, většinou sekundární

paměť. Paměťové prostory v těchto dvou nezávislých paměťových jednotkách jsou si vzájemně jednoznačně přiřazeny. Abstrakcí paměťového prostoru do zdánlivého – virtuálního rozměru ve dvojúrovňových paměťových systémech umožňuje velká sekundární paměť, která z hlediska programu zvětšuje hlavní paměť procesoru. Uživatel - programátor pracuje s pomyslnými adresami, aniž by se staral o to, jakým způsobem jsou data vyhledávána ve skutečném paměťovém systému.



Obr. 83 Dvojúrovňový hierarchický paměťový systém

Data jsou podle potřeby přenášena v blocích mezi fyzickou pamětí procesoru a sekundární pamětí a nahrazované bloky v hlavní paměti procesoru jsou zapisovány zpět do sekundární paměti. Překlad "virtuální" adresy se provádí pomocí tabulky, což umožní zobrazit souvislý logický adresový prostor do obecně po částech souvislého úseku fyzického adresového prostoru[1]. Aby tato tabulka nebyla příliš velká, logická adresa se dělí na dvě části, z nichž jedna se překládá a druhá zůstává nezměněna. Proces vyhledání a náhrady příslušného bloku dat se řídí různými **strategiemi výměny bloku**, které určují na základě sledování časových nebo statistických parametrů, který z bloků dat může být s nejmenším rizikem zaměněn. Jestliže bloky dat přenášené v obou směrech budou konstantní délky pak hovoříme o **stránkování paměti**, pokud bude mít přenášený souvislý blok dat délku proměnou v daných mezích, pak se jedná o **segmentování paměti**. V některých případech jsou obě metody kombinovány a paměťový systém používá jak stránkování tak i segmentování paměti.

7.3.2.1 Stránkovaná paměť

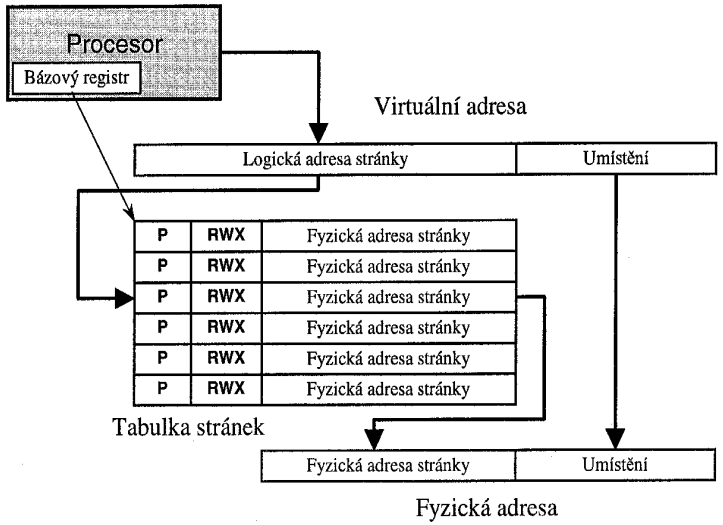
V tomto případě jsou hlavní i sekundární paměť rozděleny na stejné úseky pevné délky. Úsek v hlavní paměti nazýváme rámec, úsek v sekundární paměti stránka, a vždy platí *(počet rámců)* << *(počet stránek)*. Mapovací mechanismus zajišťuje přenos zvolené stránky ze sekundární paměti do vybraného rámce. Na programem generovanou "logickou adresu" je možno nahlížet jako na spojení čísla logické stránky a relativního umístění položky na této stránce (angl. displacement). Takto konstruovaná adresa je mapovacími mechanismy převedena na skutečnou fyzickou adresu. Jestliže se při mapování ukáže, že adresa je uložena v hlavní paměti, nahradí se pouze číslo logické stránky číslem fyzické stránky a pro operaci s hledanou položkou se použije takto vzniklá fyzická adresa. Pokud se naopak zjistí, že příslušná stránka neleží v hlavní paměti, detekuje se tzv. **výpadek stránky** a odpovídající stránka musí být přenesena do rámce hlavní paměti z paměti sekundární vyvolanou akcí operačního systému. Po přenesení stránky se operace dokončí stejným postupem, jako by k výpadku stránky nedošlo.

Mapovací mechanismus je součásti systému a převádí logické adresy stránek na fyzické adresy uvnitř adresového prostoru hlavní paměti. Tento převod se provádí tak, že z tabulky v níž jsou zapsány fyzické adresy všech stránek uložených v hlavní paměti (viz obr. 84) se přečte položka, která je adresována polem čísla logické stránky ve virtuální adrese vzhledem k bázovému registru ukazujícímu na začátek tabulky stránek. Položka tabulky stránek obvykle obsahuje tři pole

- příznak platnosti stránky **P**, který signalizuje, zda je stránka uložena v hlavní paměti a zda její obsah je platný,
- autorizační pole, které určuje oprávněnost přístupu (**RWX** - Read/Write/Execute) operace a které se při čtení z tabulky stránek porovná s oprávněními nastavenými systémem,
- fyzickou adresu stránky v hlavní paměti.

[1] Používají se i jiné metody např. zobrazení pomocí speciálního mapovacího registru; blíže viz např. [P1]

Způsob implementace tabulky stránek je rozhodující pro výkon dvojúrovňového paměťového systému, neboť každý přístup do paměti vyžaduje referenci z tabulky stránek. Pro rozsáhlejší hierarchické paměti j tabulka příliš velká, než aby se vešla do sady registrů a zároveň je její zaplnění minimální (počet stráne umístěných v hlavní paměti je většinou pouze zlomkem celkového počtu stránek). Tabulka je obvykle uložena hlavní paměti, která se adresuje přímo fyzickou adresou, aby nevznikl cyklus v požadavcích na překlady adres. K tomu, aby bylo možno umístit tabulku do hlavní paměti a přitom se minimalizovaly nároky na paměťov prostor, se používají speciální techniky, které umožňují redukovat rozměry tabulky stránek, např. víceúrovňov překlad. Víceúrovňový překlad má ovšem za následek několik přístupů do hlavní paměti při překladu adresy tedy zpomalení překladu adresy. Opětného zrychlení překladu se dosahuje tím, že pro část tabulky stránek s použije samostatná paměť, rychlejší než hlavní paměť.



Obr. 84 Mapování adres s použitím tabulky stránek

7.3.2.2 Segmentovaná paměť

Výhoda použití stránek stejné velikosti spočívající v libovolné záměně stránek je znehodnocena v případě když bude výrazný nepoměr mezi objemem přenesených a skutečně použitých dat ze stránky. Bez ohledu na to jaká část bude ze stránky využita, musí se z vnější do vnitřní paměti přenést vždy celá stránka. Segmentovan paměť odstraňuje tento nedostatek tak, že umožňuje realizovat segmenty proměnné délky, které jsou přenášen ze sekundární paměti v objemu potřebném pro efektivní zpracování programu, až do stanovené maximáln hodnoty, vyplývající z vlastností organizace počítače (adresový prostor, šířky sběrnic apod.). Segment přenesený vnější paměti zaujímá v hlavní paměti souvislý paměťový prostor, na jehož počátek ukazuje bázově adresovan logická adresa segmentu. Mapování segmentových adres do fyzického paměťového prostoru se provádí stejným způsobem jako při použití stránkování paměti – pomocí tabulky segmentů, jejíž položky obsahují příznal platnosti segmentu, přístupová práva a fyzickou adresu začátku segmentu v hlavní paměti.

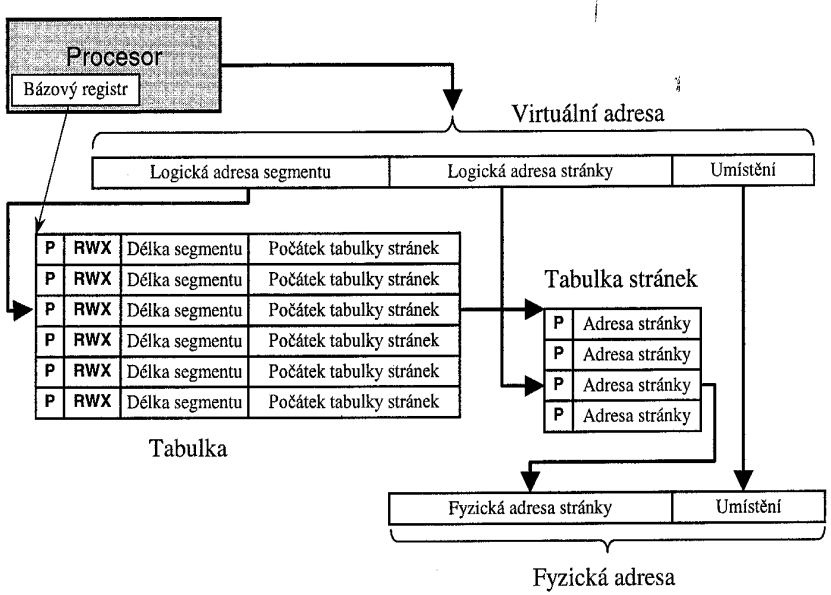
7.3.2.3 Stránkovaná segmentovaná paměť

Nevýhodou segmentované paměti je poměrně složitý mechanismus vyhledávání odpovídajícího místa pr umístění segmentu v hlavní paměti. Konstruktéři proto hledali optimální řešení odstraňující nevýhody obou metod a dospěli ke kombinaci obou metod, kde segmenty jsou rozděleny na menší bloky – stránky, jim odpovídají rámce v hlavní paměti. V kombinovaném systému stránkované segmentované paměti je pak nutn logickou adresu rozdělit v nejjednodušším případě na tři pole – ukazatel do tabulky segmentů, ukazatel d tabulky stránek a umístění položky na stránce (viz obr. 85). Pro překlad adresy pak je zapotřebí dvou tabulek tabulky segmentů a tabulky stránek. Nejdříve pomocí pole logické adresy segmentu relativně adresujeme tabulku segmentů a určíme adresu začátku tabulky stránek příslušející tomuto segmentu. Při tomto přístupu se obvykle provádí kontrola oprávnění a platnosti segmentů. Následovně se s použitím zjištěného počátku tabulky stránek

pole logické adresy stránky relativně adresuje tabulka stránek odkud se zjistí fyzická adresa počátku stránky. Tento proces probíhá sekvenčně a tedy přístupové časy do jednotlivých tabulek budou do značné míry určovat výkonnost celého paměťového systému.

Při použití stránkované segmentované paměti mohou být v hlavní paměti přítomny pouze tabulky stránek aktivních procesů a segment nemusí nutně být uložen do kompaktního bloku hlavní paměti. Umístění jednotlivých stránek do volných rámců v hlavní paměti je pak snadnější.

Metoda víceúrovňového překladu se často používá i pro stránkovanou paměť a v zájmu úspory místa se někdy používá i víceúrovňový překlad virtuálních adres (viz např. překlad adres u procesoru SPARC na 133).



Obr. 85 Mapovací mechanismus pro segmentovanou stránkovanou paměť

7.3.3 Efektivní přístupová doba

Nosným principem práce paměťové hierarchie je umístění vybraných bloků dat v paměti nejblíže u procesoru, a tak poskytnutí co nejkratšího přístupového času pro procesor. Ve chvíli, kdy je potřeba nový blok dat, dojde k jeho zavedení do hierarchicky vyššího paměťového modulu. Pokud není k dispozici volné místo, je nutno jej uvolnit. Způsoby používané pro nahrazení příslušného bloku dat jiným blokem dat ve vyšší hierarchii paměti nazýváme **strategií výměny bloku**[1] a jsou podrobněji uvedeny v kapitole 7.3.6. Existují dva případy, ke kterým může dojít při požadavku na čtení nebo zápis jednotky dat – datová jednotka je v příslušné paměti přítomna, nebo tam není a požadavek na operaci s daty nemůže být splněn.

Pokud požadavek procesoru na vyzvednutí jednotky dat je splněn na odpovídající úrovni hierarchie paměti, nazýváme tuto skutečnost obvykle anglickým termínem **hit**. Pokud tento požadavek splněn není, označujeme jej jako **výpadek bloku**. Poměr splněných požadavků k celkovému počtu požadavků v procentním vyjádření nazýváme **mírou úspěšnosti** (hit ratio) Tato míra úspěšnosti závisí na velikosti a organizaci paměti a není neobvyklé, aby při dobré konstrukci paměťového systému se přiblížila až k 99 %. Výkonnost hierarchického paměťového systému můžeme definovat pomocí **efektivní přístupové doby**, což je funkce spojující procento úspěšnosti přístupů a relativní přístupový čas mezi následujícími úrovněmi hierarchie. Název parametru vychází z toho, že hierarchický systém pamětí se jeví procesoru jako jedna velká paměť s přístupovým časem rovným

[1] Pod pojmem blok je chápán celistvý úsek dat, tedy jak stránka tak i segment.

právě této efektivní přístupové době. Při definici efektivní přístupové doby vycházíme z předpokladu, že součinitel výpadku bloku

$$\beta = \frac{N_{výp}}{N_{celk}}$$

kde β je součinitel výpadku bloku,

$N_{výp}$ počet neúspěšných pokusů o přístup do paměti,

N_{celk} celkový počet pokusů o přístup do paměti,

je konstantní. Tato "konstantnost" může být chápána jako statistický parametr při "ustáleném" stavu stroje. K přesunu bloku mezi sekundární pamětí a hlavní pamětí je potřeba čas t_B a tak efektivní doba přístupu do paměti je dána vztahem

$$t_{ef} = \beta \cdot (t_p + t_B) + (1 - \beta) \cdot t_p$$

kde t_{ef} je efektivní doba přístupu do paměti,

t_B doba přenosu bloku mezi hlavní pamětí a sekundární pamětí včetně přístupové doby k sekundární paměti

t_p doba přístupu procesoru do hlavní paměti.

Příklad

Předpokládejme například, že nejvyšší úroveň hierarchie má přístupový čas t_p =100 ns a sekundární paměť přístupový čas t_B = 10 ms včetně potřebného času na zpracování požadavku. a doby přenosu bloku mezi pamětmi Součinitel výpadku bloku je $\beta = 2.10^{-5}$. Potom efektivní přístupová doba bude 298 ns. Je zřejmé, že součinitel výpadku bloku výrazně ovlivňuje efektivní přístupovou dobu, a proto je záměrem jej snížit na minimum.

Přestože paměťová hierarchie zvyšuje složitost paměťového systému, nemusí nutně přispívat ke zvýšení latence požadavku. Existují velmi efektivní algoritmy hardware, realizující jak vyhledávání adres, tak náhradu jednotlivých bloků a pracující paralelně s ostatními obvody, takže celkový čas v cyklu vyzdvižení, dekódování a provedení instrukce není výrazně prodloužen. Pokud dvojúrovňový systém je realizován se sekundární pamětí na disku, což je nejběžnější řešení, je nutno do doby přenosu bloku započítat potřebný čas pro zpracování přerušení způsobeného detekcí výpadku stránky a dobu potřebou k načtení a přenesení dat z disku do hlavní paměti.

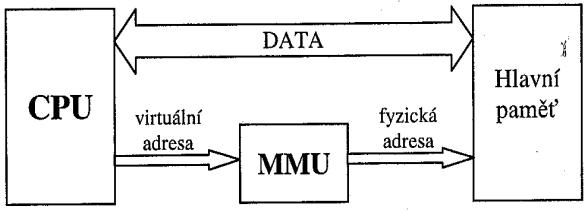
7.3.4 Příklady realizace virtuálního adresování

Při reálném návrhu virtuálních adresových systémů používají konstruktéři celou řadu vylepšení nebo doplnění, která slouží ke zvýšení efektivity práce virtuálního paměťového systému. V této souvislosti se mnohdy jednoduché principy ztrácí v nepřehledné soustavě překladových tabulek, jejichž význam není na první pohled zřejmý. Uveďme pro názornost dvě nejběžnější řešení použitá v paměťových systémech Sun SPARC a Intel Pentium.

7.3.4.1 Sun SPARC

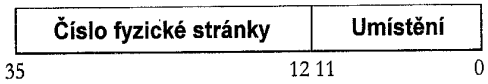
Konstruktéři procesoru SPARC, který je popsán v části 6.6.3.1, předpokládali, při návrhu architektury a organizace počítače, existenci speciální jednotky, která bude podporovat práci s virtuálními adresami. Svá doporučení shrnuli v návrhu referenční MMU (Memory Management Unit) tak, aby výrobci hardware a programátoři mohli optimalizovat charakteristiky navrhovaného počítačového systému. Hardware referenční jednotky MMU pracuje s 32 bitovým virtuálním adresovým prostorem, 36 bitovou fyzickou adresou, stránkami délky minimálně 4 kB s ochranou před neoprávněným přístupem a používá tří úrovní stránkovacích tabulek pro převod adresy. Referenční MMU SPARC specifikuje jak funkce hardware tak i organizaci a obsah překladových tabulek v hlavní paměti. Z hlediska programátora poskytuje základní funkce, nutné pro realizaci virtuální paměti:

- realizuje převod z virtuální adresy požadované běžícím procesem do fyzické adresy v hlavní paměti. Mapování je prováděno do 4 kB stránek, tedy proces požadující větší paměť nemusí být nutně umístěn ve spojité oblasti hlavní paměti. Kterákoli virtuální adresa může být mapována do jiné dostupné fyzické adresy.
- poskytuje nezbytné nástroje pro ochranu bloků paměti (stránek), takže proces nemůže číst z adresy nebo zapisovat na adresu, která je alokována jinému procesu.
- implementuje funkce nutné pro realizaci virtuálního paměťového systému. Pomocí tabulky stránek je sledována přítomnost potřebných stránek v paměti a pokud stránka není v hlavní paměti přítomna, je operačnímu systému signalizován výpadek stránky.

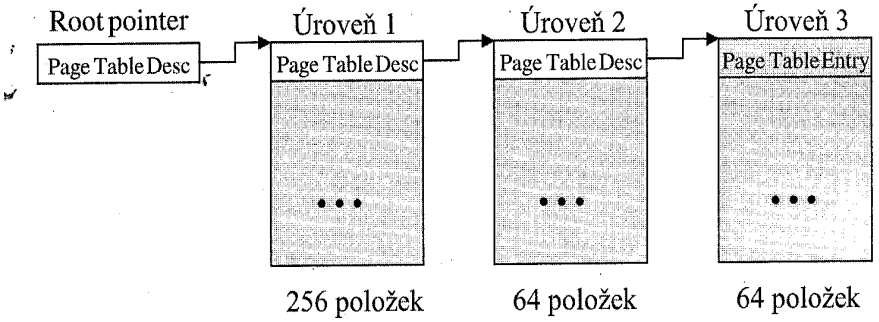


Obr. 86 Blokové schéma systému s reálnou MMU

Reálná jednotka MMU, začleněná do cesty mezi procesor a hlavní paměť na obr. 86, překládá virtuální adresy generované procesorem do fyzických adres tak, že 32-bitová virtuální adresa je překládána do 36-bitové fyzické adresy. Fyzická adresa, skládající se adresy fyzické stránky a umístění položky na této stránce, má formát



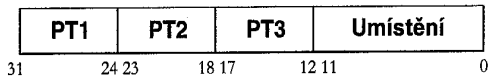
Vzhledem k tomu, že adresa počátku fyzické stránky je vždy celistvým násobkem 4kB, nejnižších 12 bitů fyzické adresy se shoduje s nejnižšími 12 bity virtuální adresy a tato část adresy se nepřekládá. Pro každou platnou virtuální stránku vyskytující se v hlavní paměti existuje odpovídající položka v tabulce stránek, která obsahuje číslo fyzické stránky pro tuto virtuální stránku. Při překladu se číslo virtuální stránky nahradí číslem fyzické stránky.



Obr. 87 Tříúrovňové mapování adresových odkazů v překladových tabulkách v architektuře SPARC

V architektuře SPARC se předpokládá, že všechny potřebné informace požadované referenční MMU pro překlad adres, jsou uloženy ve fyzicky adresovatelných datových strukturách hlavní paměti. Mapování virtuálních adres se provádí ve třech úrovních tabulek stránek. První a druhá úroveň těchto tabulek typicky, avšak ne nezbytně, obsahuje ukazatele, které se v architektuře SPARC nazývají **PTD** (Page Table Descriptor) a obsahují fyzickou adresu počátku tabulky stránek další úrovně. Na první tabulku ukazuje tzv. "Root Pointer", který je pro každý kontext unikátní a je uložen v tabulce kontextů[1]. Poslední tabulka stránek obsahuje fyzickou adresu hledané stránky a v architektuře SPARC se označuje jako **PTE** (Page Table Entry). PTE je koncipováno tak, že mimo fyzické adresy stránky obsahuje mj. deklarovaná přístupová práva a další příznaková pole. Plné tříúrovňové adresování je znázorněno na obr. 87.

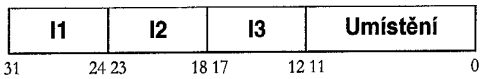
Virtuální adresa je rozdělena do tří polí následovně



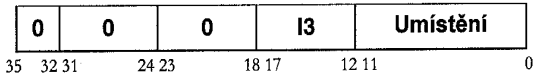
kde pole PT1 až PT3 jsou buď ukazatelé udávající umístění čísla fyzické stránky uvnitř odpovídající tabulky stránek nebo přímo čísla fyzických stránek. Pokud by reference k adrese musela používat všechny tři tabulky, pak by konstrukce adresy byla časově náročná. Avšak způsob tříúrovňového překladu použitý v architektuře SPARC umožňuje, aby ukládané bloky v paměti byly různé délky podle toho na které úrovni bude nalezen PTE, a to tak že pomocí Root Pointeru je dosažitelný lineární adresový prostor 4 GB, ukazatelem z první tabulky lineární adresový prostor 16 MB, z druhé 256 kB a z poslední 4 kB. Překlad adresy je v okamžiku nalezení PTE zastaven a příslušné PTE je použito pro konstrukci fyzické adresy položky.

Příklad

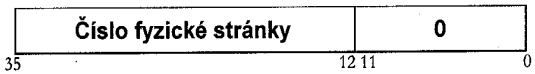
Předpokládejme, že daná virtuální adresa má tvar



a na úrovni **I2** je nalezena položka fyzické adresy PTE (namísto PTD) s číslem stránky fyzické paměti. Potom odpovídající fyzická adresa bude výsledkem operace logického součtu po bitech původní virtuální adresy[2]



a PTE ve tvaru



Poznamenejme, že nejnižších šest bitů čísla fyzické stránky musí být rovno nule, aby byl zachován požadavek na zarovnání do oblasti mapované PTE.

7.3.4.2 Intel Pentium

Architektura procesorů Intel umožňuje práci s pamětí ve dvou základních modech[3] - přímou fyzickou adresaci paměti nebo, při práci procesoru v chráněném modu, vytvoření virtuální paměti. V prvém případě je hlavní paměť obhospodařována jako lineární adresový prostor fyzických adres. Ve druhém případě všechny programové segmenty a tabulky GDT (Global Descriptor Table) a IDT (Interrupt Descriptor Table) mohou být součástí virtuálního adresového prostoru a pouze aktuální stránky budou udržovány v hlavní paměti.

Umístění stránek v hlavní paměti může být realizováno ve dvou typech datových struktur, které se v architektuře Intel nazývají adresář stránek ("page directory") a soubor tabulek stránek ("set of page tables"). Tyto datové struktury jsou umístěny v hlavní paměti. Každá položka v adresáři stránek obsahuje fyzickou adresu

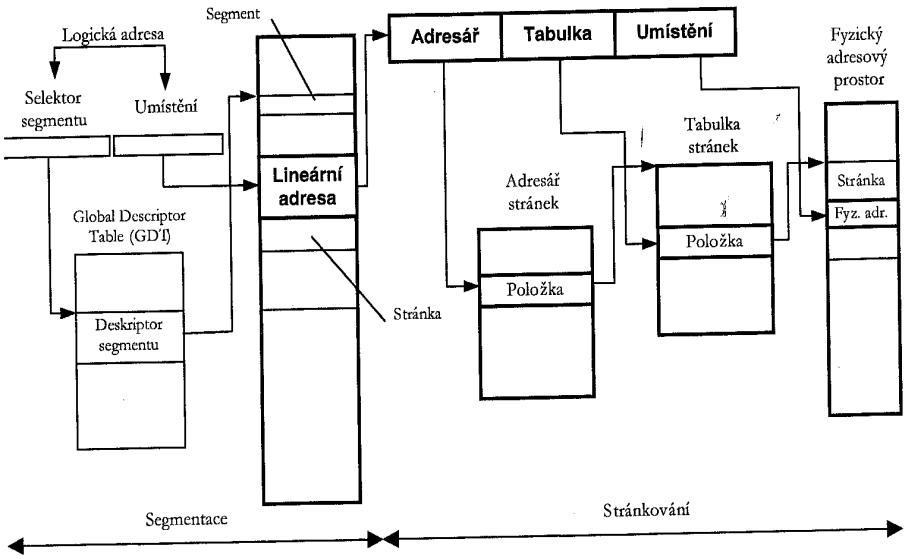
---

[1] Tabulka kontextů slouží k záznamu o parametrech procesu zpracovávaného procesorem. Její součástí je kontextové číslo procesu a ukazatel na základní mapovací tabulku paměti procesu.

[2] Virtuální 32-bitová adresa je rozšířena na 36 bitů pro operaci logického součtu s položkou o velikosti 36 bitů.

[3] Nejedná se o mod práce procesoru, který může pracovat ve čtyřech modech - chráněný mod, reálný mod, virtuální mod 8086 a mod pro řízení systému.

počátku tabulky stránek, přístupová práva a další systémové informace. Bázový ukazatel na počátek adresáře stránek pro aktivní proces je uložen v řídícím registru procesoru. V systému může existovat jeden nebo více adresářů stránek - např. každá úloha může mít svůj adresář stránek.



Obr. 88 Segmentace a stránkování v procesoru Intel Pentium

Správa paměti v architektuře Intel rozeznává dvě základní metody - segmentaci[1] a stránkování. Segmentace poskytuje mechanismus pro oddělení adresových prostorů pro programové, datové a zásobníkové moduly a vytváří v lineárním adresovém prostoru procesoru oblasti - segmenty (viz obr. 88). Stránkování poskytuje prostředky pro implementaci virtuálního paměťového systému. Narozdíl od stránkování není možno při práci procesoru v chráněném modu segmentaci potlačit. Pro přístup k libovolné adrese uvnitř segmentu je použita **logická adresa**, která obsahuje unikátní identifikátor segmentu - selektor segmentu a umístění uvnitř tohoto segmentu. Selektor segmentu obsahuje ukazatel na datovou strukturu uvnitř tabulky deskriptorů, která popisuje daný segment. Součástí této datové struktury je ukazatel na počátek segmentu v lineárním adresovém prostoru (bázová adresa segmentu). Kombinací bázové adresy segmentu a položky "umístění" z logické adresy je vytvářena adresa v lineárním adresovém prostoru procesoru. Nebude-li použito stránkování, pak lineární adresový prostor procesoru bude mapován přímo do fyzického adresového prostoru.

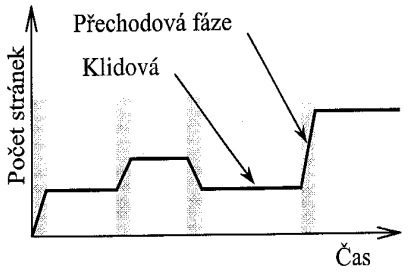
Stránkování poskytuje možnost vytváření virtuální paměti rozdělením segmentů na stránky, které jsou uloženy v hlavní nebo sekundární paměti. Pro správu virtuální paměti udržuje operační systém adresáře stránek a tabulky stránek, pomocí kterých je mapována cesta k žádaným datům. Požaduje-li proces přístup na logickou adresu, procesor převede logickou adresu na adresu v lineárním adresovém prostoru a pak použije adresář stránek a tabulky stránek k překladu lineární adresy na odpovídající fyzickou adresu. Jestliže se požadovaná stránka nenachází v paměti, procesor generuje signál "výpadek stránky", který je pokynem pro operační systém k zavedení požadované stránky ze sekundární paměti do hlavní paměti. Po zavedení stránky operační systém inicializuje opětovné provedení instrukce, která byla příčinou vzniku signálu "výpadek stránky". Aby se minimalizoval počet přístupů do hlavní paměti při překladu logické adresy, jsou položky z adresáře tabulek a z tabulky stránek, které byly v poslední době nejvíce používány uloženy ve zvláštní rychlé paměti v procesoru[2]. Návrh procesoru Pentium umožňuje řízení stránkování pomocí řídících bitů v řídícím registru. Rozšíření přímo adresovatelného fyzického paměťového prostoru na 36 bitů je umožněno nastavením bitu PAE (Physical

[1] Termín segmentace se zde vztahuje pouze k definici úseku paměti, nikoliv k metodě vytváření virtuálního paměťového systému.

[2] Tato paměť nazývaná **TLB** (Transaction Lookaside Buffer) a výrazně přispívá ke zrychlení překladu adres.

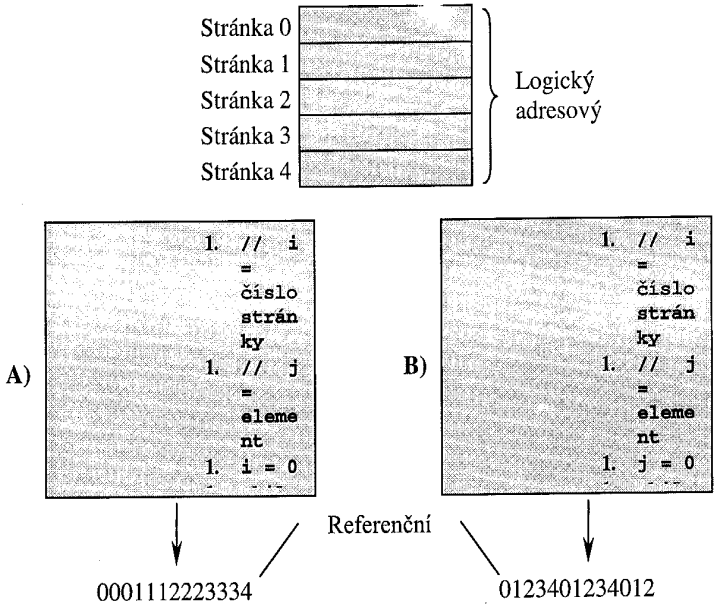
Address Extension - pouze u procesoru Pentium Pro) a bit označovaný jako PSE (Page Size Extension), umožňuje alokovat větší stránky (4 MB nebo 2 MB, podle nastavení bitu PAE).

7.3.5 Struktura programu a virtuální paměťový systém



Obr. 89 Grafické znázornění změn v hlavní paměti.

Struktura programu, a tedy i jeho chování, ovlivňuje účinnost virtuálního paměťového systému. Simulace chování programu i měření prováděná na skutečných paměťových systémech ukazují, že během práce programu existují časové úseky, kdy dochází k minimálním změnám v hlavní paměti – klidové fáze, které se střídají s úseky intenzívní výměny obsahu hlavní paměti – přechodové fáze. Grafické znázornění např. počtu stránek ve fyzické paměti v závislosti na postupu programu může být např. takové jako na obr. 89. Krátk přechodové fáze, které se vyznačují vysokou četností výpadku bloků (stránek), jsou výraznou režijní zátěž paměťového subsystému, a i když jejich celková doba trvání je menší než celková doba trvání klidových fází nelze jejich vliv na výkon celého počítačového systému zanedbat.



Obr. 90 Příklad chování programu při přístupu ke stránkám paměti

Tento způsob chování programu, doprovázený střídáním delších klidových fází a kratších přechodových fází je přirozeným jevem a snahou konstruktérů a programátorů bude ovlivnit chování programu ve prospěch klidových fází. Jedním z prostředků bude reorganizace programu, která bude vycházet z toho, jak kompilátor přiřazuje logické segmenty programu stránkám v adresovém prostoru. Uvažujme dva příklady kódu programu který obsahuje dva vnořené cykly *while*, které zpracovávají matici uloženou na stránkách v paměti dle obr. 9(Posloupnost přístupu ke stránkám v paměti vyjadřuje **referenční řetězec**, který obsahuje časovou posloupnos čísel stránek na které bylo přistupováno a můžeme jej v obecné formě vyjádřit jako

```
w = r(1) r(2) r(3) ... r(t)
```

kde `r(t)` je číslo virtuální stránky v paměti a `t` je časový okamžik zpracování programu. Je evidentní, že program označený jako A) bude vyžadovat daleko menší počet výměn stránek než program B) a tudíž poběží efektivněji ve virtuálním paměťovém systému.

7.3.6 Strategie výměny bloku

Ve virtuálním paměťovém systému povede provádění programu k vyvolání příznaků výpadku bloku, tedy segmentu nebo stránky. Počet těchto výpadků bude úměrný nejenom celkovému počtu stánek, které má systém k dispozici, ale i počtu stránek alokovaných právě probíhajícímu procesu a metodě, která bude zvolena pro jejich výměnu. Vztah mezi referenčním řetězcem, strategií výměny bloku a bloky dat, které zůstanou v paměti, tedy určuje výkonnost virtuálního paměťového sytému. Můžeme rozeznat dva mezní případy

- z celkové množiny bloků paměti bude programu alokován vždy pouze jeden blok (stránka). V tomto případě dojde k maximální frekvenci výskytu příznaků výměny bloku a taky i k maximální režii virtuálního paměťového systému.
- programu je alokována plná množina bloků v paměti a tedy celý program je umístěn v hlavní paměti. V tomto případě nebude během zpracování programu docházet k vyvolání příznaku výpadku bloku a režie virtuálního paměťového systému bude nulová.

Úkolem konstruktéra bude navrhnout takový způsob obsazení a výměny jednotlivých bloků mezi hlavní a sekundární pamětí, aby minimalizoval režii virtuálního paměťového systému v případě, kdy je nutné bloky dat mezi oběma pamětmi vyměňovat. V návrhu virtuálních paměťových systémů se vyskytuje celá řada strategií směřujících k optimalizaci mechanismu výměny bloku, které vycházejí ze základního předpokladu, totiž výměny stránky jako důsledku akce uživatelského programu. To znamená, že operační systém, nikoliv programátor, je zodpovědný za správné obměňování stránek podle toho, jak jsou vyžadovány právě aktivním programem. Při hodnocení strategie výměny bloku budou důležitá dvě hlediska - **časové hledisko**, zohledňující dobu, po kterou je příslušný blok umístěn v hlavní paměti a **četnostní hledisko**, sledující počet přístupů k danému bloku.

Tato hlediska je možno vzájemně kombinovat tak, že jedno z nich bude při volbě strategie hlavní a druhé pomocné. Mezi základní strategie výměny bloku, pomineme-li strategii náhodného výběru, patří:

FIFO (First in First Out) – nahrazuje vždy nejstarší blok v paměti, tedy ten, který byl zaveden nejdříve. Rozhodující je tedy časové hledisko.

LIFO (Last In First Out) – blok, který bude nahrazen je ten, který byl naposledy zaveden do paměti, tedy nejmladší blok. Tato strategie rovněž preferuje časové hledisko a je použitelná pouze ve specifických případech.

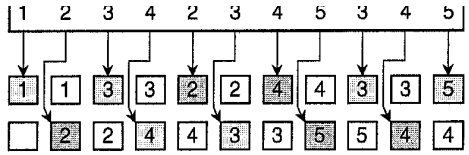
LRU (Least Recently Used) – nahrazuje se blok, který byl nejdéle nepoužit. V této strategii se používá současně hlediska počtu přístupů a stáří bloku v paměti.

LFU (Least Frequently Used) – nahrazen bude blok s nejmenším počtem přístupů, tedy rozhodující je hledisko četnosti přístupů, tedy používání bloku programem.

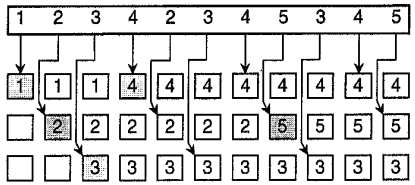
Pro teoretické studie se často používá strategie **OPT** (Optimal), která nahrazuje blok, jenž bude nejméně používán. Vzhledem k požadavku na znalost budoucího vývoje referenčního řetězce, která obvykle není k dispozici, je používána pouze jako referenční při studiu účinnosti různých algoritmů výměny stránek. Existují i modifikace výše uvedených základních strategií - např. v systému Linux je implementována strategie nazývaná "CLOCK", která je vlastně modifikovanou strategií FIFO, doplněnou o bit aktivity. Prakticky používané strategie jsou většinou zjednodušené nebo modifikované algoritmy FIFO nebo LRU, které budou podrobněji probrány dále.

7.3.6.1 Strategie FIFO

Strategie FIFO, jak již bylo uvedeno, nahradí stránku v hlavní paměti, která byla v této paměti nejdéle. Pro ilustraci této strategie předpokládejme referenční řetězec ve tvaru **12342345345**, který reprezentuje požadavky na přítomnost bloku v paměti při běhu hypotetického programu. Alokujeme-li vždy maximálně dvě stránky v hlavní paměti, pak obdržíme následující diagram výměny stránek:

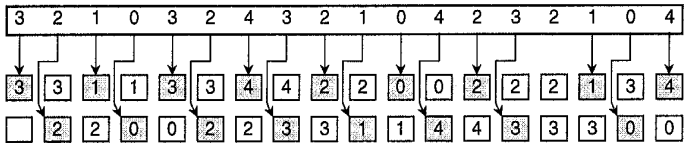


Jak je vidět, během práce programu došlo k 11 výpadkům stránky a stránka v hlavní paměti musela být nahrazena (nahrazující stránky jsou označeny tmavší výplní). Rozšíříme-li počet alokovaných stránek v hlavní paměti na tři, získáme následující průběh výměny stránek během práce hypotetického programu

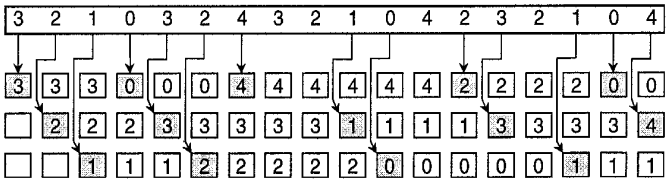


Počet výpadků bloku se díky přidání dalšího bloku v hlavní paměti snížil na 5. Vzhledem k tomu, že celkový počet stránek potřebných pro program je rovněž 5, jedná se o optimální způsob výměny stránek ve virtuálním paměťovém systému.

Strategie FIFO, která je jednoduchá pro pochopení i implementaci, však nemusí vždy pracovat ta optimálně, jako v uvedeném příkladě. Bude-li stránka, která se bude vyskytovat na počátku referenčního řetězc pouze stránkou, která bude souviset s inicializací programu, pak její nahrazení při nejbližším požadavku n výměnu je korektní a logické. Avšak v případě, kdy tato stránka bude obsahovat např. často používaná data, je výměna při nejbližší příležitosti způsobí přídavnou časovou režii na její opětovné zavedení. Strategie FIF(dokonce při některých typech referenčních řetězců způsobí více výpadků stránek při zvětšení počtu alokovanýc stránek v hlavní paměti. Tento fenomén, který se nazývá **Beladiova**[1] **anomálie** [B1], můžeme ukázat příklad který používá referenční řetězec **3210324321042321O4**.

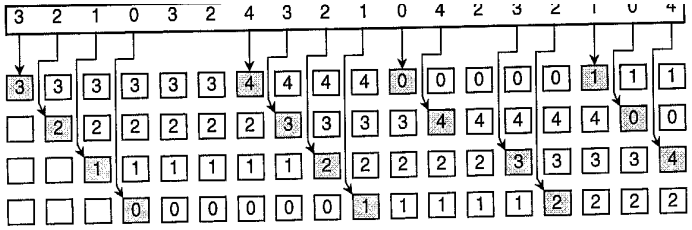


V prvém případě alokace dvou stránek v hlavní paměti došlo k 17 výpadkům stránek při celkově 18 přístupech Zvýšením počtu alokovaných stránek obdržíme diagram



se sníženým počtem výpadků na 14. Další alokací stránek pro hypotetický program v hlavní paměti se vša diagram výměny stránek změní na

[1] Laszlo A. Belady – specialista firmy IBM, posléze šéfredaktor časopisu IEEE Transaction on Software Engineering a ředitel MERL, americké pobočky Mitsubishi Electric, zabývající se výzkumem v oblasti počítačů.

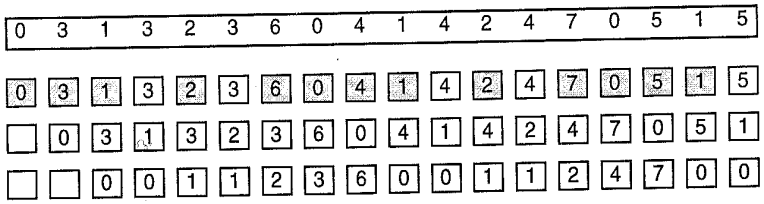


a současně se zvýší počet výpadků na 15. Právě Beladiova anomálie je důvodem častějšího používání strategií, které takové chování nevykazují, jako je LRU.

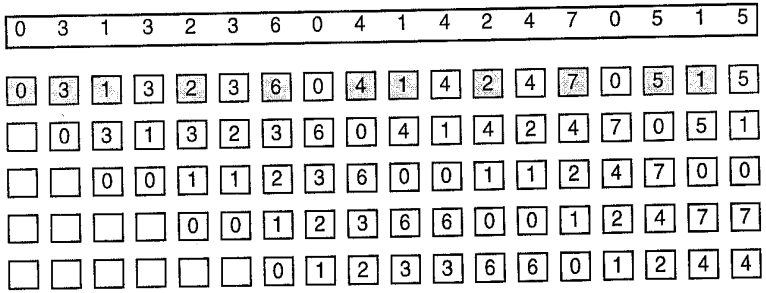
7.3.6.2 Strategie LRU

Strategie výměny stránek nahrazující stránku, která byla v poslední době nejméně používána zohledňuje jak časové měřítko stáří stránky (bloku) tak i četnost přístupu ke stránce. Důležitým předpokladem je, že *"nejbližší minulost je možno aproximovat na nejbližší budoucnost"*. Při této strategii operační systém používá informace o tom kolik času uplynulo od posledního přístupu na stránku. To může být realizováno např. tak, že ke každému rámci je přiřazen čítač, jehož hodnota se v pravidelných intervalech zvětšuje o 1 a při přístupu na stránku umístěnou v rámci se nuluje. Vyměněna bude ta stránka, jejíž čítač bude obsahovat nejvyšší hodnotu.

Použijeme-li grafické zobrazení z předcházejícího odstavce pak pro referenční řetězec **031023604142470515** a tři alokované stránky obdržíme diagram



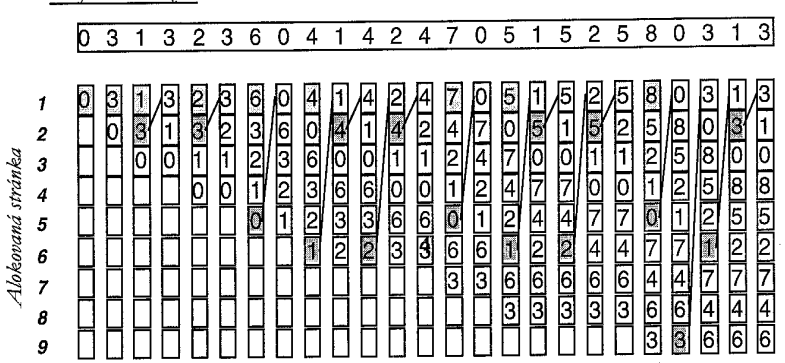
Způsob zobrazení se poněkud liší – čtverečky v tomto diagramu nereprezentují konkrétní stránky alokované v paměti, ale jen počet stránek a nejčerstvěji používaná stránka je vždy v horní řadě. Touto konstrukcí jsme obdrželi jakýsi zásobník, ve kterém se stránky střídají podle toho, která z nich má být použita s tím, že vždy stránka, která byla nejméně v poslední době používána je na dně zásobníku. Hloubka symbolického zásobníku je dána počtem alokovaných rámců. V uvedeném diagramu jsou tmavší výplní označena místa, kde došlo k výměně stránky – celkem 13 výpadků bloku z 18 přístupů. Budeme-li v konstrukci modelu pokračovat, pak zjistíme, že s přidáním alokovaných stránek pro program se nebudou měnit reference v jednotlivých řádcích, ale sníží se počet výpadků stránek. Pro případ pěti alokovaných stránek



se sníží počet výpadků na 11, přičemž jednotlivé řádky z diagramu pro tři alokované stránky zůstanou zachovány. LRU a některé jiné strategie, které mohou být reprezentovány zásobníkovým modelem, se také označují souhrnným názvem **zásobníkové strategie**.

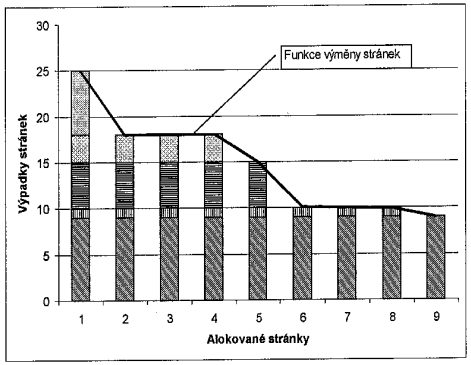
Přijmeme-li výše uvedený pohled na strategii LRU jako na zásobník stránek, pak se můžeme pokusit o určení dalších parametrů, kterými bude možno charakterizovat proces výměny stránek mezi hlavní a sekundární

paměti. Předpokládejme, že referenční řetězec programu, který bude [illegible] **031323604142470515258031313**, pak diagram ukazující práci strategie LRU bude mít tvar



Použijeme-li předcházející graf ke konstrukci funkce, která bude znázorňovat počty výpadků stránek p různém počtu alokovaných stránek, obdržíme histogram podle obr. 91. Sloupce v histogramu reprezentu akumulovaný počet výpadků stránek a každý dodatečný výpadek stránky způsobený snížením počt alokovaných stránek je v histogramu odlišen. Tato závislost je známa jako **funkce výměny stránek** a její normováním k maximálnímu počtu výpadků stránek obdržíme **četnost výpadků stránek** pro daný systén Tato veličina, respektive její průběh v závislosti na počtu stránek je důležitým měřítkem výkonnosti strategi výměny stránek pro daný paměťový subsystém.

Pro případ multiprogramového systému se často používá parametru, který je inverzí četnosti výpadku stránek a je znám jako **funkce délky života stránky**. Optimalizace doby života stránky je možná změnou některého z parametrů, jako např. velikosti stránky, počtu alokovaných stránek pro program nebo změnou strategie výměny stránek. Je zřejmé, že pokud počet stránek alokovaných v hlavní paměti procesu odpovídá počtu potřebných stránek s ohledem na skupiny vytvořené lokalitami referencí procesu, pak doba života stránky v hlavní paměti odpovídá celkové době zpracování procesu. Proces během doby obsadí všechny potřebné stránky (křivka roste) a posléze se ustálí na konečné hodnotě kde setrvá až do ukončení procesu (plató).



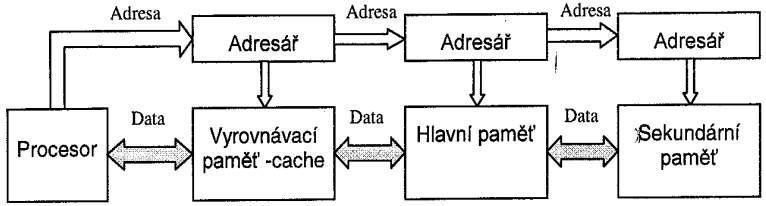
Obr. 91 Průběh funkce výměny stránek k diagramu v textu

7.3.7 Tříúrovňový paměťový systém

Přenosy mezi hlavní pamětí a sekundární pamětí, které byly řešeny v předcházejících kapitolách, s soustředily zejména na zajištění potřebné velikosti paměti a tedy vytvoření virtuálního adresového prostor s relativně nízkou cenou za uloženou jednotku informace a únosnou rychlostí přístupu k uloženým datůn Moderní procesory však požadují od pamětí daleko vyšší rychlosti, než je schopna poskytovat běžná hlavi paměť, a proto se mezi hlavní paměť a procesor vkládá další úroveň paměti, označovaná jako **vyrovnávac paměť** – v počítačové terminologii známá pod vžitým názvem **cache**[1]. Tato paměť bývá asi desetkrát rychlej

[1] Název byl poprvé použit u počítače IBM System/360 model 85. Zdomácněl a používá se v běžném jazyce.

než hlavní paměť a jejím zařazením mezi procesor a hlavní paměť lze výrazně zvýšit výkon systému za předpokladu, že jsou splněny základní podmínky realizace víceúrovňového paměťového systému (viz kapitola 7.3.3), spočívající ve vhodném poměru přístupových dob a nízkém součiniteli výpadku bloku. Pro programátora je tato paměť nedostupná, a tak ji nemůže ani adresovat ani rozhodovat o tom, které informace do ní budou uloženy. Strategie výměny dat a přesuny jednotlivých stránek jsou řízeny přidruženým hardwarem, bez možnosti ovlivnění na úrovni operačního systému. Základní blokové schéma tříúrovňové paměti je na obr. 92, kde jsou znázorněny jednotlivé adresáře i paměťové bloky v sériovém propojení.



Obr. 92 Tříúrovňový paměťový systém

Jak je vidět, některé informace jsou ve tříúrovňovém hierarchickém paměťovém systému uloženy až třikrát a pomocí adresářů je možno zjistit, kde se hledaná informace nachází a jaký je její současný stav – zda je aktuální či nikoliv. Zajištění platných dat na potřebné úrovni bude nemalý problém a jeho řešení vede na složité strategie výměny bloků dat a označování jejich validity.

Obvyklý způsob práce tříúrovňového paměťového systému začíná okamžikem, kdy proces, který je zpracováván procesorem, vydává požadavek na poskytnutí dat pomocí adresových vodičů do adresáře vyrovnávací paměti cache. Zde je porovnána tato adresa s adresami, které se momentálně nacházejí ve vyrovnávací paměti, tedy i v adresáři, a pokud se odpovídající adresa najde, jsou data předána procesoru přímo, aniž by se přistupovalo do hlavní paměti.. Není-li požadovaná adresa nalezena, požadavek postupuje do adresáře hlavní paměti, a pokud je uspokojen zde, hardware řídící spolupráci vyrovnávací paměti a hlavní paměti přesune odpovídající data, vždy celý blok okolo požadované adresy s ohledem na již diskutovanou lokalitu referencí, do vyrovnávací paměti a upraví obsah adresáře vyrovnávací paměti. Pokud není požadavek procesoru uspokojen ani obsahem hlavní paměti, vyvolá se příznak výpadku bloku a operační systém zajistí přesun potřebných dat ze sekundární paměti do hlavní a jejím prostřednictvím i do vyrovnávací paměti. Zároveň jsou odpovídajícím způsobem upraveny adresáře.

Jako adresáře se pro vyrovnávací paměť obvykle používá asociativní paměť (viz strana 118), aby se zkrátila doba potřebná k vyhledání a uspokojení požadavku procesoru. Strategie výměny stránky se volí podobně jako v případě dvojúrovňového paměťového systému, nejčastěji LRU, FIFO nebo náhodný výběr. Přednost se přitom dává výběru takové strategie, která se bude snadno realizovat v hardware počítače.

Při návrhu vyrovnávací paměti jsou použita dvě kritéria, která určují parametry této paměti a vztahují se ke způsobu výběru položky z vyrovnávací paměti, kde rozeznáváme

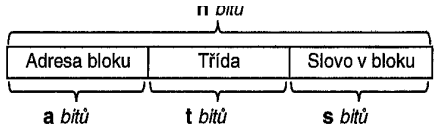
- vyrovnávací paměti s přímým mapováním (direct mapping cache),
- vyrovnávací paměti s asociativním mapováním (fully associative cache),
- vyrovnávací paměti s částečně asociativním mapováním (set associative cache)

a k zapisovací strategii, která zahrnuje vyrovnávací paměti

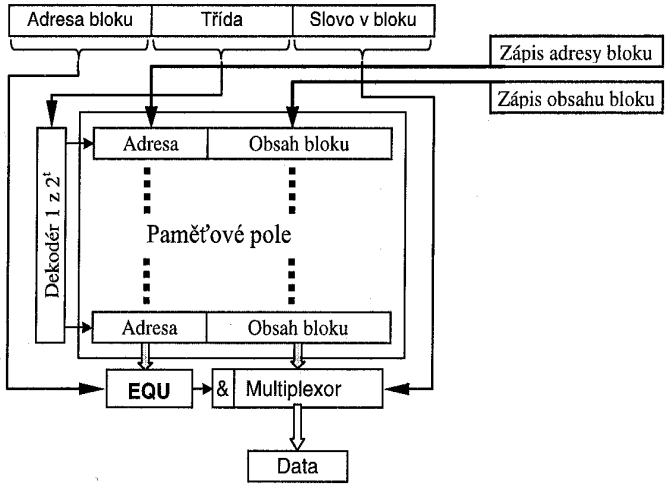
- se současným zápisem do hlavní paměti (write through cache) a
- se zpožděným zápisem (delayed write).

7.3.7.1 Vyrovnávací paměť s přímým mapováním

Předpokládejme, že hlavní paměť má kapacitu 2^n slov, je rozdělena na bloky s velikostí 2^s slov a struktura reálné adresy je rozdělena do tří polí tak, jak je znázorněno na obrázku



Pole nejnižších bitů délky ***s*** označuje polohu slova v bloku, prostřední pole délky ***t*** označuje příslušnost slova d třídy slov a nejvyšší skupina bitů délky ***a*** označuje adresu bloku ve třídě. Adresář vyrovnávací paměti j organizován tak, že jednotlivé části adresy jsou použity pro přímou adresaci položky nebo pro porovnán s obsahem uloženým v adresáři, jak je znázorněno na obr. 93. Při zápisu do vyrovnávací paměti se uloží adres bloku hlavní paměti odkud byla data přepsána a obsah tohoto bloku. Požaduje-li procesor přístup k datům, čás adresy odpovídající třídě je použita pro adresování uložené adresy bloku ve vyrovnávací paměti. Pokud s přečtená adresa bloku shoduje s požadovanou, je otevřen multiplexor, který vybere z datové oblasti příslušn data.

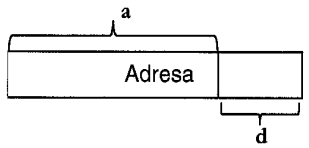


Obr. 93 Blokové schéma vyrovnávací paměti s přímým mapováním

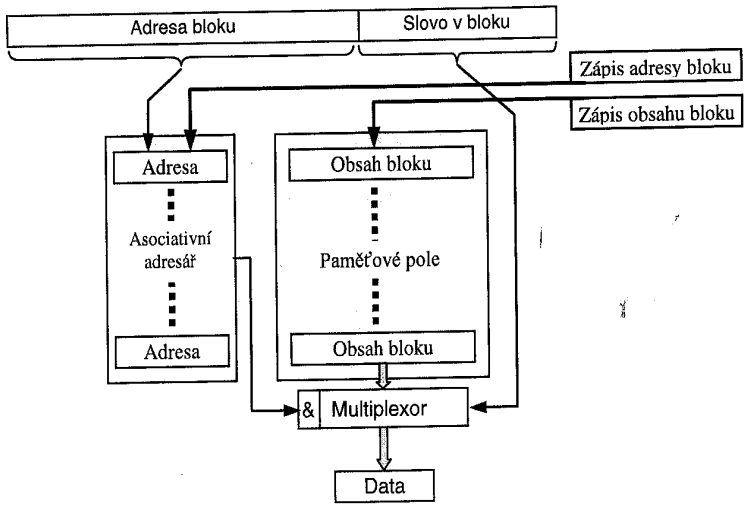
Tento způsob adresace vyrovnávací paměti je na první pohled velmi nevýhodný, protože při každém novém zápisu bloku je nutno uvolnit odpovídající rámec ve vyrovnávací paměti, nicméně pro svoji jednoduchou konstrukci dosáhla vyrovnávací paměť s přímým mapováním značné popularity.

7.3.7.2 Vyrovnávací paměť s asociativním mapováním

Narozdíl od vyrovnávací paměti s přímým mapováním, vyrovnávací paměť s plně asociativním mapováním patří k nepružnějším způsobům organizace adresáře a přilehlé vyrovnávací paměti, neboť každý blok hlavní paměti může být uložen kdekoliv ve vyrovnávací paměti. Jednotlivá slova jsou adresována oddělenými poli adresy pro výběr bloku (a) a pro výběr slova v bloku (d) takto



Ve vyrovnávací paměti s plně asociativním mapováním se při výběru bloku z hlavní paměti uloží do asociativní části adresáře adresa bloku, přičemž je jedno do které položky v adresáři je tato adresa zapsána. Při adresaci procesorem, který požaduje předání dat, je adresová část **a** porovnána s plně asociativní částí adresáře. Pokud je adresa bloku nalezena, otevře se multiplexor, který je adresován **d** bity adresy a podle jejich stavu se vybere příslušné slovo z bloku.



Obr. 94 Vyrovnávací paměť s plně asociativním mapováním

7.3.7.3 Vyrovnávací paměť s částečně asociativním mapováním

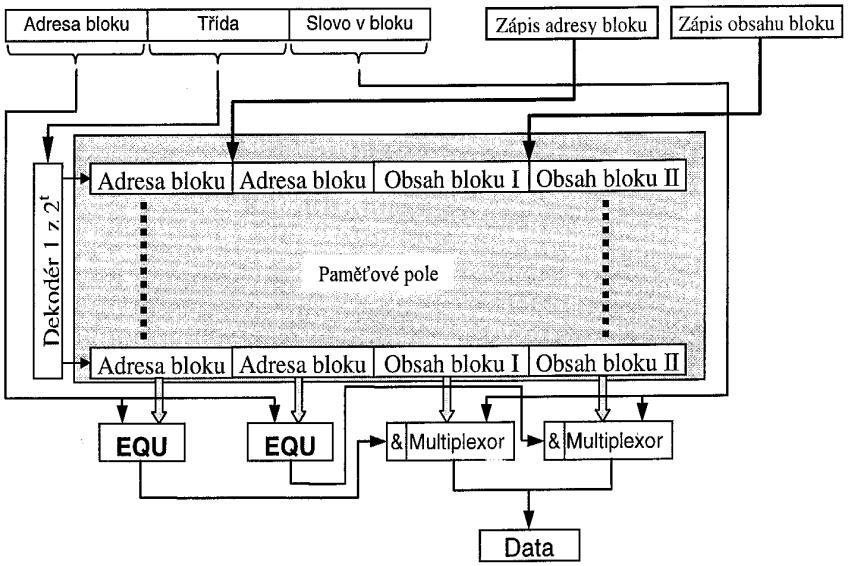
Obvodová složitost asociativních adresářů vedla konstruktéry k hledání kompromisního řešení, jehož výsledkem byl návrh vyrovnávací paměti s částečně asociativním mapováním. Jeho koncepce vychází z návrhu vyrovnávací paměti s přímým mapováním, která je rozšířena o další pole adresáře i vlastní paměti. Důležitým parametrem vyrovnávací paměti s částečně asociativním mapováním je její stupeň asociativity, který udává, kolik slov adresáře je porovnáváno v jednom kroku, neboli kolikrát existuje vedle sebe ta část adresáře, která obsahuje adresy bloku.

Na obr. 95 je blokové schéma vyrovnávací paměti s částečnou asociativitou stupně 2, jejíž adresovací slovo je rozděleno na tři pole, obdobně jako u vyrovnávací paměti s přímým přístupem. Při zápisu jsou ukládány do příslušných oblastí jak adresy bloků, tak i jejich obsahy. Přitom adresy bloků mohou být zapsány jen na určité místo, které je dáno obsahem pole adresy určující třídu. Každá množina tříd však obsahuje více bloků než je možno do vyrovnávací paměti umístit, proto při umisťování bloků ve vyrovnávací paměti dochází ke konfliktům. Aby těchto konfliktů vznikalo co nejméně, volí se při konstrukci adresové pole třídy tak, aby bloky, které za sebou následují v hlavní paměti patřily do různých tříd, neboť se předpokládá, že při práci s hlavní pamětí převládá sekvenční čtení a zápis.

Vyhledání příslušného slova ve vyrovnávací paměti probíhá tak, že na základě dekódování třídy se vyberou adresy bloků patřících do této třídy. Ty jsou na výstupu v obvodech ekvivalence porovnány s hledanou adresou bloku a v případě souhlasu je otevřen příslušný multiplexor. Výsledná data předaná na výstup jsou vybrána adresací multiplexoru pomocí adresy slova v bloku.

Paměti s částečnou asociativitou mají řadu výhod, pro které se staly oblíbeným konstrukčním prvkem vyrovnávacích pamětí. Jejich hlavní výhodou je, že vyhledávání a čtení probíhá současně, takže vybavovací doba je výrazně kratší než u pamětí s plnou asociativitou. Rovněž jejich konstrukce je jednodušší, a tím i levnější. U současných počítačů se používají nejčastěji paměti se stupněm asociativity 2 nebo 4, avšak existují vyrovnávací paměti se stupněm asociativity až 16.

Procesory nejčastěji se vyskytující na běžném trhu obvykle mají několik úrovní vyrovnávacích pamětí s různým stupněm asociativity. Např. procesor firmy DEC Alpha 21164, používá vyrovnávací paměť L1 s přímým mapováním, paměť L2 je konstruována jako vyrovnávací paměť s částečnou asociativitou stupně 3 a vyrovnávací paměť L3 je opět realizována s přímým mapováním. Adresáře hlavní paměti (TLB) jsou navrženy jako plně asociativní a to jak pro data tak i pro instrukce. Procesor Pentium Pro (architektura P6) používá na nejnižší úrovni (L1) vyrovnávací paměť s částečnou asociativitou stupně 2 pro data a stupně 4 pro instrukce, vyrovnávací paměť L2 je realizována s částečnou asociativitou stupně 4 podobně jako adresáře hlavní paměti (TLB).



Obr. 95 Vyrovnávací paměť s částečnou asociativitou stupně 2

7.3.8 Zapisovací strategie

Při použití víceúrovňového paměťového systému vzniká vážný problém souladu dat v jednotlivýc hierarchických stupních paměti. Tento problém není ani tak markantní při čtení z jakékoliv úrovně paměti jak při zpětném zápisu. Navíc, pokud se o tutéž paměť podílí několik procesů, nebo ji sdílí několik jednotek, s z problému souladu dat stává poměrně náročný úkol. Jde o to, aby se změna, kterou provede kterákc z jednotek nebo kterýkoli z procesů co nejrychleji projevila v hlavní paměti, tak aby byla k dispozici ostatní procesům nebo jednotkám[1].

Můžeme rozlišit dva případy, které bude nutno sledovat – koherenci dat a konzistenci paměťových operac Pojem **koherence dat** můžeme definovat jako vlastnost paměťového systému, která zajistí, že při jakémkoli průběhu programu bude každá dotčená paměťová buňka obsahovat data v souladu s průběhem programu a že

- jednotlivé operace kteréhokoliv procesu budou paměťovým systémem zpracovány v pořadí, jak by předány dotčeným procesem
- hodnota přečtená z dotčené paměťové buňky bude poslední hodnotou, která byla do této buňk zapsána v pořadí jak požadavky na zápis přicházely

Ke splnění těchto požadavků je nutné, aby operace paměťového systému spuštěné kterýmkoli procese (jednotkou) byly viditelné pro všechny další procesy a aby operace zápisu byly všemi zúčastněnými proces vnímány ve stejném pořadí.

Koherence dat je tedy jejich autentičností pro danou buňku paměťového systému, avšak klade r paměťový systém podmínky, které jsou z hlediska celé paměti problematické. Jestliže požadujeme aby zápis d buňky byl stejně viditelný pro všechny procesy (jednotky) pak si můžeme klást otázku, kdy bude takový záp "zviditelněn" a jakým způsobem bude zajištěna synchronizace mezi zápisem a čtení z různých proces

[1] Tento problém samozřejmě existuje i v hierarchii hlavní paměť/sekundární paměť, ale tam není tak markantní, neboť výměna dat je v tomto případě zprostředkována operačním systémem, který je schopen koherenci zajistit.

Abychom mohli tyto otázky zodpovědět, musíme mít k dispozici model, který specifikuje požadavky na pořadí, ve kterém bude stanoveno, kdy operace paměti (iniciovaná libovolným procesem) bude zviditelněna ostatním procesům. Takový model nazveme modelem **konzistence paměťových operací**. Můžeme tvrdit, že paměťový systém používaný několika procesy (jednotkami) je konzistentní, jestliže výsledek akce kteréhokoli procesu je vždy stejný i když se operace všech procesů provádí v libovolném pořadí a operace každého procesu se provádí podle jeho vlastního algoritmu.

Pro zajištění souladu dat mezi hlavní pamětí a vyrovnávací pamětí se používá několik strategií zápisu do vyrovnávací paměti, z nichž nejběžnější jsou: současný zápis, zpožděný zápis a předběžný zápis. Obě strategie se používají jak v jednoprocesorových tak i ve víceprocesorových systémech a zajišťují poměrně efektivní proces sladění obsahů vyrovnávací paměti a hlavní paměti.

7.3.8.1 Strategie současného zápisu

Vyrovnávací paměť používající strategii současného zápisu (write-through cache) je propojena s hlavní pamětí a procesorem tak, že je umožněn současný zápis jak do vyrovnávací paměti, tak do hlavní paměti. Tímto způsobem dojde k současnému obnovení informace na obou místech. Nepříjemnou vlastností této strategie je, že hlavní paměť je výrazně pomalejší než procesor a vyrovnávací paměť a procesor musí čekat na splnění požadavku zápisu až do doby, kdy je ukončen zápis do hlavní paměti. Tím se ztrácí výhoda rychlého přístupu procesoru do paměti při zápisu. Aby se eliminoval tento nedostatek, používá se někdy zvláštní sady registrů (write buffer) mezi hlavní pamětí a vyrovnávací pamětí, která má stejnou dobu přístupu jako vyrovnávací paměť. Při operaci zápisu je do ní uložena zapisovaná informace, procesor může pokračovat ve své činnosti a autonomní řídící obvody rozhraní mezi vyrovnávací pamětí a hlavní pamětí zařídí uložení požadované informace na odpovídající místo v hlavní paměti.

V důsledku zpoždění operace zápisu do hlavní paměti dojde ke krátkodobé disproporci mezi obsahem hlavní paměti a vyrovnávací paměti a v případě, kdyby procesor požadoval data ze stejné adresy na jakou ji předtím uložil, došlo by pravděpodobně k chybě paměťového systému. Tato situace se řeší různými metodami, které většinou staví na nastavení příznaku neplatnosti dat (invalidate flag) ve vyrovnávací paměti po dobu, kdy panuje rozpor mezi oběma obsahy. Podobně, pokud je v systému použito více vyrovnávacích pamětí, které spolupracují s jednou hlavní pamětí, mechanismy nastavení neplatnosti dat musí nastavit neplatnost dané oblasti vyrovnávací paměti u všech pamětí, které se procesu zápisu nové informace nezúčastnily.

7.3.8.2 Strategie zpožděného zápisu

Při použití strategie zpožděného zápisu (*write-back cache* nebo též *copy-back*, *store-back*) je zapisovaná informace uložena pouze do příslušného bloku ve vyrovnávací paměti. Tím dojde k tomu, že takto modifikovaný blok existuje jenom ve vyrovnávací paměti a k obnovení informace v hlavní paměti dojde až v okamžiku, kdy bude tento blok při výměně ukládán zpět do hlavní paměti. Procesor při modifikaci bloku nastaví rovněž **příznak modifikovaného bloku** (*dirty bit*), který označuje blok, jenž byl po dobu své existence ve vyrovnávací paměti modifikován, ale nebyl ještě uložen zpět do hlavní paměti. Není-li blok označen příznakem modifikovaného bloku, pak je "čistý", jeho obsah se shoduje s obsahem téhož bloku v hlavní paměti a není nutno jej při výměně zapisovat zpět do hlavní paměti.

Tím že blok bude zapisován do hlavní paměti pouze při výměně bloku, a ne s každým zápisem do bloku, dojde k výraznému snížení komunikace mezi vyrovnávací pamětí a hlavní pamětí, což je výhodné zejména v multiprocesorových systémech. Mimo zápisu bloku, který vyplývá z požadavku na výměnu bloků při práci procesoru, musí mechanismy spolupráce vyrovnávací paměti a hlavní paměti zajistit, aby tento blok byl zapsán do hlavní paměti i tehdy, je-li obsah dotčeného bloku požadován jinou jednotkou v systému. Otázka zajištění koherence dat a konzistence paměťových operací je řešena u strategie zpožděného zápisu mnoha způsoby, počínaje exkluzivitou v přiřazení bloku pouze jednomu procesoru (*Berkeley Ownership Protocol*) a konče monitorováním modifikovaných bloků v systému (snooping).

7.3.8.3 Strategie předběžného zápisu

V roce 1983 navrhl John Goodman [G83] strategii kombinující oba přístupy současného i zpožděného zápisu, které je označována jako strategie předběžného zápisu – *write ones protocol*. Tato metoda používá strategii současného zápisu pro první zápis do vybraného bloku vyrovnávací paměti a všechny další zápisy jsou prováděny pouze do vyrovnávací paměti se strategií zpožděného zápisu. Tedy modifikace učiněné po prvním zápisu do vyrovnávací paměti budou uloženy až společně s výměnou bloku mezi vyrovnávací pamětí a hlavní pamětí. Protokol, zajišťují spolupráci hierarchie pamětí je v tomto případě složitější a je řízen čtyřstavovým automatem, jehož stavy se odvozují od hodnot dvou bitů spojených s každým blokem vyrovnávací paměti. Tyto bity umožňují rozlišit mezi stavy

- neplatný blok (*invalid*), kterým je označen blok vyrovnávací paměti neobsahující žádná data,
- platný blok (*valid*), obsahující data, která byla načtena v předchozím kroku z hlavní paměti a nebyla ještě modifikována – tudíž blok vyrovnávací paměti a hlavní paměti jsou konzistentní,
- rezervovaný blok (*reserved*) je označení, které znamená, že tento blok byl změněn právě jednou a i tomto případě jsou vzhledem k použité metodě dotčené bloky v obou pamětech konzistentní,
- modifikovaný blok (*dirty*) obsahuje více změn než jednu, a tudíž je nekonzistentní s hlavní pamětí - obdoba modifikovaného bloku u strategie se zpožděným zápisem.

Implementace strategie s předběžným zápisem je obtížnější než implementace strategie se současným zápisem, zejména v multiprocesorových systémech. Oproti protokolu se zpožděným zápisem vyžaduje zápis do hlavní paměti i tehdy, když data uložená do hlavní paměti nebudou sdílena s ostatními jednotkami v systému. Koherence dat při použití strategie s předběžným zápisem u multiprocesorových sestav je zajištěna tím, že při prvním zápisu, který prochází až na úroveň hlavní paměti, jsou všechny ostatní vyrovnávací paměti informovány o změně v dotčeném bloku.

7.3.8.4 Porovnání strategií zápisu

Vyrovnávací paměť se strategií současného zápisu patří i v současnosti k nejoblíbenějším konstrukcím zejména pro jednoduchost implementace a proto, že není nutno cokoliv zapisovat do hlavní paměti, dojde-li k výměně bloku mezi hlavní pamětí a vyrovnávací pamětí. Další výhodou vyrovnávací paměti se strategií současného zápisu je trvalá konzistence obsahu vyrovnávací paměti a hlavní paměti, zatímco u pamětí se strategií zpožděného zápisu jsou nejčerstvější data udržována pouze ve vyrovnávací paměti. Jeho podstatnou nevýhodou jsou vyšší požadavky na potřebnou šířku pásma sběrnice mezi vyrovnávací pamětí a hlavní pamětí. To je důvodem vzrůstu popularity pamětí se strategií zpožděného zápisu u multiprocesorových systémů, kde dochází ke značné úspoře potřebné komunikace mezi dvěma paměťovými systémy, a tím k uvolnění sběrnice, neboť k zápisu do hlavní paměti dochází pouze při výměně bloku mezi hlavní pamětí a vyrovnávací pamětí. Bohužel nekonzistence dat v hlavní a vyrovnávací paměti vede k problémům koherence vyrovnávacích pamětí se zpožděným zápisem. Kompromisní řešení spočívá ve strategii s předběžným zápisem, které výrazně snižuje komunikační zátěž mezi hlavní a vyrovnávací pamětí (asi na 50%) a částečně řeší problém koherence dat.

7.3.9 Redukce ztrát výkonu v hierarchických paměťových systémech

Zavedení vyrovnávacích pamětí do organizace počítačového systému výrazně zvyšuje jeho výkon, díky urychlení přístupu procesoru do paměti. Avšak práce vyrovnávací paměti, přenosy bloků mezi jednotlivými hierarchiemi a další režijní aktivity jsou daní, která se projevuje přídavnými ztrátami na výkonu systému. Během doby byla ověřena a zavedena do praxe celá řada technik, snižujících tyto sekundární ztráty. Většina z nich se zaměřila na snížení počtu výpadků bloku, redukci ztrát při výpadku bloku a snížení doby potřebné pro zjištění existence bloku ve vyrovnávací paměti.

7.3.9.1 Snížení počtu výpadků bloku

Při studiu důvodů vedoucích k výpadku bloku můžeme rozlišit tři základní případy:

1. **Nutné výpadky**, které se vyskytují při prvním přístupu na adresu, jež je umístěna v bloku, který není ve vyrovnávací paměti.
2. **Výpadky z důvodu nízké kapacity** vyrovnávací paměti. K těm dochází tehdy, když mezi dvě blízké adresy patřící do téhož bloku je umístěna další adresa patřící do jiného bloku. Pokud vyrovnávací paměť nemá dostatečnou kapacitu, první blok je uložen, nahrazen novým blokem a ten je téměř ihned nahrazen původním blokem.
3. **Výpadky způsobené konfliktem** bloku, které se podobají předcházejícím a jsou důsledkem použití částečně asociativního výběru nebo přímého výběru z vyrovnávací paměti. K výpadku dojde proto, že blok z hlavní paměti musí být uložen na místo, kde je uložen jiný blok, ačkoliv ve vyrovnávací paměti existují dosud volná místa.

Některé jednoduché techniky mohou odstranit některé druhy těchto výpadků, např. výpadky bloku způsobené konfliktem jsou zcela eliminovány u vyrovnávacích pamětí s plně asociativním výběrem. Jejich implementace je však nákladná a vede ke zpomalení výběru dat z vyrovnávací paměti. Výpadky způsobené nízkou kapacitou vyrovnávací paměti lze omezit zvýšením její kapacity. Mezi další používané postupy patří:

- **Zvětšení velikosti bloku** sníží počet nutných výpadků většinou za cenu delší doby potřebné k přenosu bloku z hlavní paměti do vyrovnávací paměti.
- **Zvýšení asociativity** zvýší míru úspěšnosti přístupů do vyrovnávací paměti. Zároveň však, vzhledem ke složitějšímu vyhodnocení shody než u výběru s přímým přístupem, se zvýší doba potřebná k vyhodnocení shody a tím i celková přístupová doba vyrovnávací paměti. Studie ukazují, že vyrovnávací paměť s částečně asociativním výběrem stupně 2 a velikosti N/2 má zhruba stejný výkon jako vyrovnávací paměť s přímým výběrem a s velikostí N. Rovněž zvýšení asociativity nad stupeň 8 většinou již nevede k výraznému zvýšení výkonu paměťového výkonu.
- **Použití odkládací paměti** (*victim cache*), což je malá vyrovnávací paměť, do které je odložen blok, který byl právě nahrazen ve vyrovnávací paměti. Základní myšlenka přidání odkládací paměti je stejná jako u realizace řádné vyrovnávací paměti – lokalita referencí. Při následujícím výpadku bloku se tedy prohlíží nejdříve odkládací paměť, a pokud je hledaný blok nalezen, další hledání v hlavní paměti se zastaví.
- **Pseudoasociativní návrh** umožňuje, abychom v případě výpadku bloku, na rozdíl od adresáře s přímým výběrem, měli další možnosti výběru. To znamená, že návrh adresáře s omezenou asociativitou přinese výhodu oproti adresáři s přímým výběrem, avšak různá doba přístupu, daná různou dobou vyhodnocení shody, může vést k problémům při vazbě vyrovnávací paměti na linku proudového zpracování.
- **Předběžné vyzvednutí bloku** je realizováno tak, že se spolu s blokem, který měl být zaveden do vyrovnávací paměti zavede i další nejbližší blok. Jsou realizovány i konstrukce, využívající předběžného vyzvednutí bloku na základě provedení speciální instrukce, která je vložena do programu kompilátorem. Kompilátor rovněž může uspořádat data tak, aby se minimalizoval počet přesunů mezi hlavní pamětí a vyrovnávací pamětí.

7.3.9.2 Redukce ztrát při výpadku bloku

Ztráty způsobené narušením práce vyrovnávací paměti v důsledku výpadku bloku spočívají zejména v dodatečné časové režii na výměnu bloku nebo v disproporci rychlosti při zápisu a při čtení. K zamezení těchto ztrát se obvykle používají vyrovnávací paměti pro zápis, které zamezí zabrzdění práce linky proudového zpracování při zápisu do paměti v případě použití strategie současného zápisu. Data nebo bloky které mají být zapsány do vzdálenější paměti, jsou odeslány do rychlé vyrovnávací paměti a linka proudového zpracování pokračuje ve své práci. Vyrovnávací paměť pomocí vlastních autonomních mechanismů zajistí zápis příslušných dat do vzdálenější paměti. Problém nastává, pokud procesor žádá o stejná data dříve, než jsou zapsána do paměti.

Při použití malých bloků dochází často k výpadkům bloků, a tím i k vyvolání přenosu bloku do vyrovnávací paměti. Výhoda, kterou získáme úsporou času při přenosu malých bloků je vyvážena častými výměnami a velkou pamětí potřebnou pro konstrukci adresáře. Alternativní řešení spočívá v rozdělení bloků ve vyrovnávací paměti do sub-bloků. Shoda při nalezení bloku ve vyrovnávací paměti pak musí být ještě ověřena kontrolou přidruženého bitu "platnosti sub-bloku", což opět přispívá k větší časové režii.

Ve většině situací dojde k výpadku bloku při nedosažitelnosti konkrétního slova ve vyrovnávací paměti. K tomu, aby procesor nemusel čekat na naplnění celé vyrovnávací paměti se používají dvě metody. První metody "včasného restartu" (*early restart*) spustí procesor ihned, jakmile je požadované slovo k dispozici ve vyrovnávací paměti. Metoda předběžného výběru kritického slova (*critical word first*) přenese nejdříve do vyrovnávací paměti požadované slovo, spustí procesor a pokračuje v přenosu zbytku bloku. Obě tyto metody pracují nejlépe při velkých blocích ve vyrovnávacích pamětech.

Metoda neblokující vyrovnávací paměti je schopna plnit požadavky procesoru i po dobu, kdy se vyřizuje požadavek na výměnu bloku. Tento typ vyrovnávacích pamětí se používá u procesorů se spekulativním prováděním instrukcí a umožňuje, aby instrukce, které následují po instrukci, jež zapříčinila výpadek bloku byly prováděny bez zdržení čekáním na výměnu bloku. V organizaci procesorů se spekulativním prováděním instrukcí existují návrhy, u nichž může být současně aktivních několik požadavků na výměnu bloku, aniž by to brzdilo zpracování instrukcí.

Jednoduchá vyrovnávací paměť je prostředkem, který přizpůsobuje rychlost hlavní paměti rychlosti procesoru. Použitím další sekundární vyrovnávací paměti vložené mezi hlavní vyrovnávací paměť a hlavní paměť a vhodnou volbou poměru velikostí a rychlostí vyrovnávacích pamětí může být dosaženo zvýšení výkonu celého paměťového systému spolu se snížením efektivní doby přenosu bloků.

7.3.9.3 Redukce času potřebného k nalezení shody

I když se zdá, že hledání metod zajištění vysokého výkonu paměťového systému při výpadcích bloků je nejdůležitějším problémem, nezajištění rychlého poskytnutí dat v případě shody může mít stejný, ne-li větší dopad na výkonnost celého systému. Dlouhá doba potřebná k nalezení shody má bezprostřední dopad na délku cyklu procesoru, a tedy i na dobu potřebnou ke zpracování instrukce např. v lince proudového zpracování. Nejjednodušší cestou je zajištění rychlé reakce vyrovnávací paměti je používání malých a jednoduchých vyrovnávacích pamětí. Malá vyrovnávací paměť se může realizovat přímo na čipu procesoru a adresář vyrovnávací paměti s přímým mapováním může reagovat velmi rychle. Druhou metodou je použití několika úrovní vyrovnávacích pamětí.

7.3.10 Virtuální a fyzické vyrovnávací paměti

Většina současných systémů používá speciální jednotky pro spojení procesoru s pamětí[1], která zprostředkuje mimo jiné i mapování virtuálního adresového prostoru do fyzického adresového prostoru. Takto vznikají dvě možnosti, kde může být vyrovnávací paměť umístěna

- mezi procesorem a jednotkou řízení paměti, kdy vyrovnávací paměť pracuje pouze s virtuálními adresami a fyzický adresový prostor je jí neznám,
- mezi jednotkou řízení paměti a hlavní pamětí, kdy vyrovnávací paměti jsou naopak předkládány pouze fyzické adresy a virtuální adresovací prostor zůstává skryt.

První případ umístění vyrovnávací paměti mezi procesor a jednotku řízení paměti nazýváme **virtuální vyrovnávací pamětí**, zatímco druhý případ je označován jako **fyzická vyrovnávací paměť**. Obě řešení mají své výhody a nedostatky. Při přístupu procesoru k virtuální vyrovnávací paměti není nutno, aby adresování procházelo jednotkou řízení paměti, a proto je přístup rychlejší v případě, kdy dojde ke shodě s daty přítomnými ve vyrovnávací paměti. Na druhou stranu virtuální adresa je delší než fyzická adresa a proto i virtuální vyrovnávací paměť bývá složitější. Navíc, řídící logika, která musí pracovat v násobném adresovém prostoru, bude vyžadovat rozsáhlejší obvody a oba uvedené vlivy budou činit virtuální vyrovnávací paměť dražší.

Fyzická vyrovnávací paměť, která je umístěna na společné sběrnici s hlavní pamětí může být zároveň použita pro monitorování aktivit[2] této sběrnice a tedy i paměti, což je výhodné při zajišťování konzistence vyrovnávacích pamětí. Většina dnešních procesorů obsahuje integrované jednotky řízení paměti, a tak externí vyrovnávací paměti jsou vždy fyzické.

---

[1] Tzv Memory Management Unit - MMU

[2] Tzv. bus snooping.

Jestliže systém nefunguje, doplň
další jednotky.

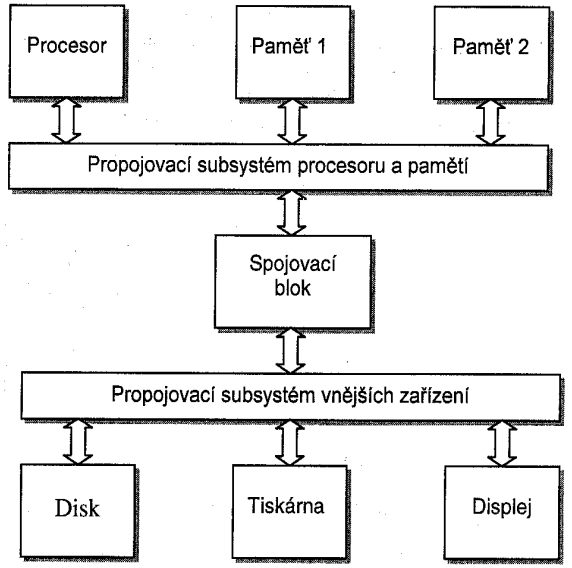
A. Bloch: "Murphy's Law

8 Propojovací subsystémy

V předcházejících kapitolách byly popsány základní výpočetní jednotky počítače – procesor a paměti, jejich vzájemné propojení se předpokládalo. V počítačovém systému však propojení jednotlivých bloků systému tvo velmi náročný subsystém, jemuž se vyplatí věnovat pozornost. Pro předání dat z jedné jednotky do druh budeme muset akceptovat čtyři základní charakteristiky takového přenosu:

- charakter přenášených dat,
- velikost (objem) přenášených dat,
- zdroj (původ) dat,
- cíl (místo určení) dat.

Obvykle výše uvedené charakteristiky jsou doprovázeny dalšími údaji, jako je priorita datové položky, je časování atp. Tyto přenosy bude nutno zajistit jak mezi procesorem a pamětí, tak i k dalším komponentům systému – zařízením vstupu a výstupu. Podle toho můžeme propojovací subsystémy rozdělit na **vnitřní**, slouží většinou k přenosům mezi paměťmi a procesory, a **vnější** použité pro připojování periferních zařízení. Každ propojení bude dále charakterizováno svou rychlostí, šířkou pásma a rovněž cenou, kde parametr <*přenosový výkon*>/<*cena*> bude rozhodující pro použitelnost propojovacího systému v komerčních návrzích. Propojovac systém může být i hierarchický, jak ukazuje obr. 96.



Obr. 96 Hierarchické propojení počítačového systému

Základním rozlišovacím prvkem mezi jednotlivými metodami organizace propojení jednotek počítače je způsob šíření informace mezi jednotlivými jednotkami. Pokud se informace bude pomocí spojového systému šířit tak, že vždy bude aktivována zvláštní cesta, a tedy část propojovacího subsystému se vlastního přenosu konkrétní datové jednotky nezúčastní budeme hovořit o **směrové organizaci** propojovacího systému. Ve druhém případě bude předávaná informace mezi kterýmikoli jednotkami předávána po obecných cestách a bude dostupná všem připojeným jednotkám ve stejném časovém intervalu. Taková organizace aktivuje pro přenos dat vždy celý přenosový systém a informace se šíří všemi směry. Označujeme ji jako sdílenou nebo též **sběrnicovou organizaci** propojovacího systému. Přitom šířka použité datové cesty může být sekundárním parametrem charakteristiky propojení a v těchto případech hovoříme obvykle o **sériových kanálech**, kde informace je přenášena po elementárních jednotkách (bitech), a o **paralelních kanálech**, které umožňují přenos větších jednotek informace. Toto dělení však může narážet na úskalí v případě, kdy např. bude vhodným kódováním

přenášen násobek elementární jednotky pomocí kanálu pro přenos jedné elementární jednotky[1] v tzv. základním pásmu.

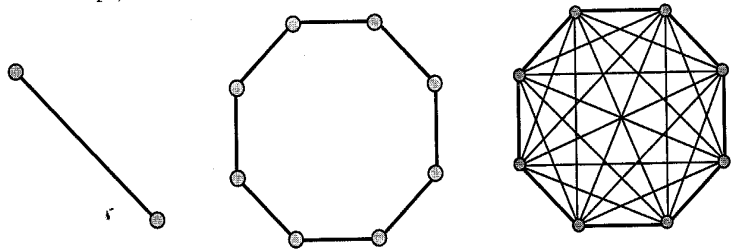
8.1 Směrové organizace

Směrové organizace mají nejrůznější možná topologická uspořádání od lineárních až po složité vícerozměrné topologie. Mezi nejjednodušší patří propojení dvou krajních jednotek, označované jako propojení **bod-bod**. Mezi složitější topologie patří propojení **do hvězdy**, vyznačující se připojením krajních jednotek k centrální jednotce ve středu propojovacího systému nebo **stromové propojení,** ať už realizované jako **binární strom**, kde každý uzel má dvě větve, či jako strom s možností připojení více větví k jednomu kořeni. Tyto topologie spolu s dalšími se zejména používají při návrhu a výstavbě velmi výkonných systémů, kde komunikační složka je podstatným omezujícím prvkem výkonnosti systému.

Vzájemná spojení jednotlivých prvků systému určují způsob propojení sítě, neboli **topologii propojovacího subsystému**. Vlastnosti navržené topologie určují, jakým způsobem budou připojené jednotky sdílet zdroje a co toto sdílení bude stát. Pro obecnější pohled na komunikační subsystém a diskusi jeho vlastností předpokládejme, že propojovací síť je složena z **uzlů**, které komunikují prostřednictvím **spojů**. Ve skutečném systému pak uzly mohou být nahrazeny procesory, paměťmi nebo jinými výkonnými jednotkami. Každý spoj budeme v této diskusi vždy vnímat jako dvoubodové propojení[2], což znamená, že na jednom spoji se nemůže podílet více uzlů.

8.1.1 Základní pojmy

Dva uzly jsou **sousední**, jestliže mezi nimi existuje spoj. **Stupeň uzlu** je dán počtem jeho sousedů. Některé základní topologie, zahrnující spojení **bod-bod, kruhová síť** a **síť s úplným propojením,** jsou ukázány na obr. 97. Jak je vidět, v propojení bod-bod existuje pouze jediný spoj, každý uzel v kruhové topologii je spojen jenom s dvěma okolními uzly a každý uzel v topologii úplného propojení je spojen se všemi ostatními uzly. Prakticky to znamená, že volba topologie má vliv na cenu systému, protože více spojů k uzlu znamená více logiky použité k implementaci těchto spojení.



a) Spojení bod-bod b) Kruhová síť c) Síť s úplným

Obr. 97 Základní topologie sítí

Jestliže uzel není přímo propojen s ostatními uzly, přenášené informace budou muset procházet mezilehlými uzly předtím, než dosáhnou svého místa určení. **Průměrem sítě** nazveme nejkratší cestu ze všech možných cest mezi kterýmikoli dvěma uzly sítě. Je možno ukázat, že kruh a plně propojená síť jsou dva extrémní případy. Kruh ***n*** uzlů má průměr ***n/2***, kdežto plně propojená síť má pevný průměr ***1*** nezávisle na tom, kolik uzlů je připojeno. Je zřejmé, že průměr kruhové sítě poroste s množstvím přidaných uzlů, zatímco průměr plně propojené sítě zůstává stejný. Opačným jevem je, že kruhová síť může být expandována do nekonečna bez změny stupně uzlu, avšak je-li nový uzel přidán do plně propojené sítě, pak je mu nutno přiřadit nový spoj s každým existujícím uzlem - tedy zvýší stupeň uzlu. Průměr sítě prakticky definuje dobu, potřebnou pro přenos informace mezi dvěma nejvzdálenějšími uzly sítě, a tak má vliv na celkovou rychlost systému. Pojem **míry**

[1] To je častý případ komunikačních kanálů využívajících vícestavové modulace (fázové, amplitudové).

[2] Až na některé případy, kdy bude zdůrazněna jiná funkce spoje.

topologie můžeme použít pro označení stupně zvýšení složitosti komunikace v souvislosti s přidáním dalších uzlů. V systémech s velkou mírou topologie může být přidáno mnoho uzlů bez podstatného zvýšení množství logiky požadované pro implementaci topologie a současně bez výrazného zvýšení průměru topologie.

8.1.2 Statické sítě

Výše uvedené rozdělení propojovacích technologií na směrové a sběrnicové může být dále rozveden v případě směrových technologií podle požadavků kladených na síť. Jedním z parametrů, které se používá k taxonomii propojovacích sítí je schopnost sítě měnit svoji konfiguraci, tedy dělení na

- sítě se stálou konfigurací propojení – **statické**,
- sítě s proměnnou konfigurací - **dynamické**.

Otázka kvalifikace nebo definice statického či dynamického chování však nemusí pouze souviset s fyzickou výstavbou sítě, ale může být též specifikována jako její behaviorální prvek nebo logická charakteristika. V našem pojetí však budeme předpokládat, že sítě charakterizované jako statické jsou takto popsány s ohledem na jejich fyzikální konstrukci a jejich fyzické propojení stejně jako směr pohybu zpráv zůstávají neměnné po dobu existence sítě.

Strukturu statické sítě obvykle vyjadřujeme grafem, v němž každý vrchol odpovídá jednotce systému a hrana komunikačnímu spoji. Poměrně velký počet různých druhů statických sítí, který se prakticky používá s ohledem na odvozuje z požadavku přizpůsobení předpokládané činnosti. Pro každý typ systému je třeba, v závislosti na druhu převažující komunikace, použít jiný typ statické sítě. K nejznámějším statickým propojovacím schématům patří lineární sítě, kružnice, hvězda, stromy, různé typy mříží a krychle.

8.1.2.1 Lineární síť

Lineární síť propojující dva uzly je nejjednodušším případem propojení jednotek ve směrové propojovací technologii. Ad hoc propojení každého zařízení s odpovídajícím vstupem nebo výstupem jiného zařízení se charakterizuje též jako propojení **bod-bod**, neboť přenosu dat se účastní pouze tato dvě zařízení. Toto spojení může být realizováno samostatně, nebo může být součástí složitějšího propojovacího schématu. Při propojení bod-bod získáme největší možnost přizpůsobení přenosových charakteristik obou zařízení, a tedy je možno docílit i nejvyšších přenosových rychlostí. Podobně jako u pamětí je možno mluvit o velké **šířce pásma** tohoto způsobu připojení.

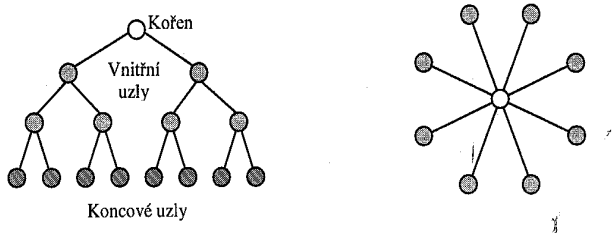
Přes tuto výhodu má propojení bod-bod celou řadu nevýhod. Jeho vysoká cena dále vzrůstá s rozšiřováním počtu propojených zařízení, kde každé další zařízení přispívá do kombinace všech možných propojení. Zároveň návrh zařízení propojovaných bod-bod je pouze velmi obtížně realizovatelný pro případy dalšího rozšiřování systému. O takovém rozšíření musí být již předem rozhodnuto, musí být stanoveny maximální možnosti rozšíření a s tím související omezení. Pokud se síť sestavená ze spojení bod-bod nesestaví zcela, pak náklady na realizaci přípravy dodatečného rozšíření jsou nevyužity.

Lineární síť propojující dva uzly může být dále rozšířena na více uzlů s použitím mezilehlých uzlů, které se účastní komunikace v případě, že přenos probíhá mezi dvěma uzly, které spolu bezprostředně nesousedí. Každý mezilehlý uzel musí tedy zprávu přijmout, na nezbytnou dobu uložit a opět odeslat dalšímu prvku v přenosové cestě. Je zřejmé, že **stupeň uzlu lineární sítě** bude konstantní hodnota **2,** s výjimkou koncových uzlů, a **průměr lineární sítě** bude ***n-1***, kde ***n*** je počet uzlů. Nízký stupeň uzlu lineární sítě zjednodušuje adresaci, avšak zároveň je nevýhodný v případě poruchy, kdy u acyklických lineárních sítí je znemožněno spojení mezi uzly ležícími na opačných koncích od místa poruchy.

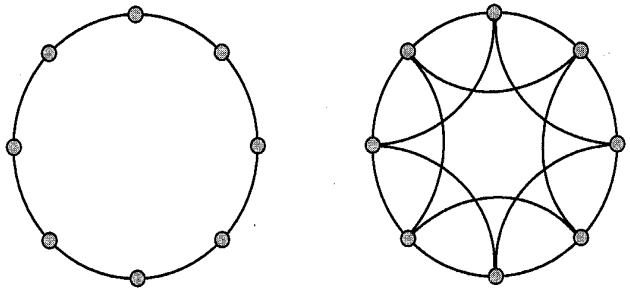
8.1.2.2 Stromové sítě a hvězda

Stromové a hvězdicové sítě mohou být chápány jako speciální případy lineárních acyklických sítí, neboť podobně jako ony obsahují uzly různého stupně a přerušení cesty mezi prvky vede k nepřístupnosti části sítě. Při bližším zkoumání zjistíme, že stromová topologie obsahuje tři různé typy uzlů, kořenový uzel, vnitřní uzel a koncový uzel, kde každý z těchto typů uzlů má různý stupeň. Hvězdicová topologie má odlišný uzel v centru sítě, neboť centrum je propojeno s každým uzlem sítě (viz obr. 98). Pro stromovou topologii používající binární strom můžeme nalézt uzly s maximálním stupněm uzlu 3 a s průměrem sítě $2.[\log_2 (N+1) - 1)]$. Podobně pro hvězdicovou síť lze nalézt maximální stupeň uzlu rovný počtu uzlů v síti zmenšenému o jedna, tedy ***N-1***, což je stupeň středového uzlu sítě. Průměr sítě bude konstantní a roven 2. Z hlediska pohledu na stromovou nebo

hvězdicovou síť jako celek můžeme tvrdit, že uzly jsou **nesymetrické**. V každé takové topologii, kde neexistují symetrické uzly, tedy topologie je z hlediska uzlů asymetrická, se odlišný uzel se může stát úzkým místem komunikačního systému.



Obr. 98 Stromová a hvězdicová topologie



a) Jednoduchý kruh b) Dvojitý kruh

Obr. 99 Propojovací sítě s kruhovou topologií

Pro formalizaci pohledu na definici úzkého profilu komunikace můžeme použít vlastnost známou jako **osová šířka**, což je minimální počet spojů, které musí být odebrány, abychom rozdělili danou topologii do dvou nezávislých sítí stejného rozměru, lišící se maximálně o jeden uzel. Osová šířka stromové topologie je zřejmě jedna, neboť odebráním jednoho spoje připojeného ke kořenovému uzlu se stromová topologie rozpadne do dvou stromových topologií. V některých případech se můžeme setkat s dalším rozšířením tohoto pojmu, na tzv. **osovou šířku pásma systému**[1], což je přenášený informační výkon nebo též šířka pásma kterou poskytoval spoj odebraný během zjišťování osové šířky. Tento parametr je užitečný při stanovení nejnižšího výkonu zpracovávaného analyzovanou sítí, neboť odpovídá nákladům na přenos dat z jedné části systému do druhé.

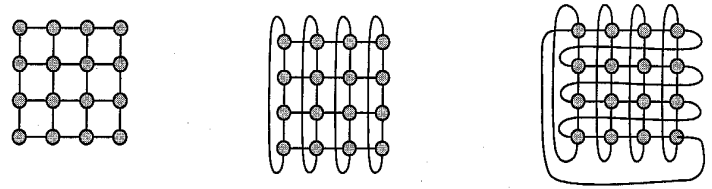
8.1.2.3 Kruhová síť

Asymetrické uzly lineárních, stromových a hvězdicových sítí sice zjednodušují adresování v síti, ale zároveň přinášejí složitější rozhodovací algoritmy, různé pro různé uzly, a potenciální úzká místa v přenosových cestách - to je markantní zejména na hvězdicové síti. Velmi žádoucí vlastností propojovací sítě bude tedy symetrie uzlů. Síť, která obsahuje pouze symetrické uzly, pak nemusí mezi jednotlivými uzly rozlišovat, neboť pohled na zbytek sítě je stejný z každého uzlu. Kruhové sítě, sítě s úplným propojením a hyperkrychle jsou sítě s plně symetrickými uzly. Typické kruhové sítě jsou dnes reprezentovány jednoduchým a dvojitým kruhem jak je ukázáno na obr. 99.

V obou těchto propojovacích schématech existuje několik cest, jak mohou být uzly sítě propojeny a současně všechny uzly mají stejný stupeň. U jednoduché kruhové sítě je stupeň uzlu sítě vždy 2 a průměr sítě N/2, zatímco u dvojité kruhové sítě je stupeň vrcholu dvojnásobný, tedy 4, a průměr sítě dosahuje hodnoty N/4.

[1] Známá též pod anglickým názvem "bisection bandwith"

Planárními topologiemi nazýváme dvourozměrné sítě, které vycházejí ze struktury matice uzlů, kde kažc uzel je spojen pouze se svými sousedy. Příklady planárních sítí jsou uvedeny na obr. 100. Mimo základníh spojení, planární mříže uvedené pod označením a), můžeme též použít zpětné cesty, to znamená spojení, kc uzel na vrcholu sítě je spojen s odpovídajícím uzlem na nejnižším bodě sítě. Pokud provedeme tato propojer pouze např. zespoda nahoru, pak planární síť se změní na síť, která bude umístěna na povrchu třírozměrnéh válce. Doplníme-li tuto síť zpětným propojením v horizontálním směru, pak cylindrický charakter sítě se změní vzniklá síť bude tvořit povrch toroidu. Z tohoto důvodu se síťové topologie, které používají zpětné propojer často nazývají **toroidními topologiemi**. Další úpravou této toroidní topologie může být posunutí zpětnýc propojení o jednu nebo více řad (sloupců), což nebude mít výrazný vliv na geometrický tvar sítě, ale topolog přejde na druh známý jako **zborcený toroid**. Spoj, který začne v uzlu umístěném v levém horním roh pokračuje v této technologii spojitě, prochází každým uzlem pouze jednou a navrátí se opět do téhož uzlu.



a) planární b) válcová topologie c) zborcený toroid

Obr. 100 Příklady planárních sítí

I když se tyto topologie zdají exotické, jejich použití ve výkonných systémech je běžné. Např. topologi uvedená na obr. 100 pod písmenem c), zborcený toroid, je známá také jako mříž ILLIAC, podle počítače, kd byla poprvé použita. Firma Cray, proslulá výstavbou velmi výkonných systémů, používá vlastní topologii pi propojování procesorů, která je tvořena trojdimenzionálními mřížemi, jež jsou propojeny zpětnými spoji a tvoi toroid ve všech třech směrech.

8.1.2.5 Hyperkrychle

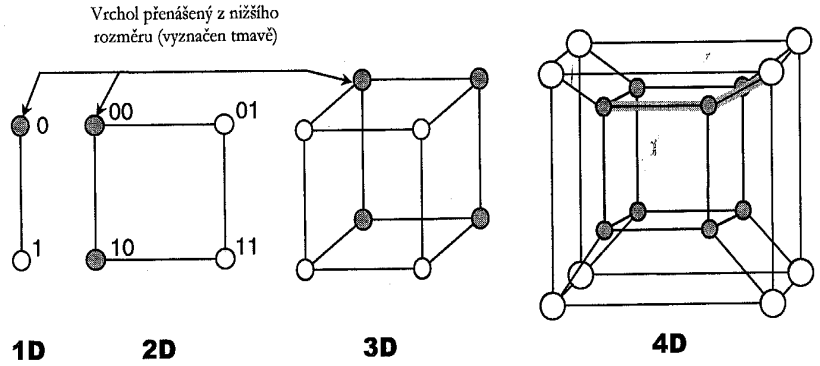
Hyperkrychle patří k důležitým topologiím často používaným při výstavbě výkonných systémů. Způsol propojení jednotek v hyperkrychli je graficky znázorněn na obr. 101. K lepší představě hyperkrychle můžem postupně začít s její konstrukcí tak, že začneme s jednodimenzionální krychlí realizovanou spojem propojujícím dva uzly. Přidáním dalšího rozměru vznikne čtverec, tedy dvojdimenzionální krychle, obsahující čtyři uzly Následující třídimenzionální krychle s osmi uzly je totožná s běžnou geometrickou představou. Opakováním tohoto postupu budeme moci konstruovat n-dimenzionální krychle tak, že začneme vždy od $(n-1)$ dimenzionální krychle, vytvoříme její identickou kopii a přidáme spoje z každého uzlu originálu dc korespondujícího uzlu kopie. Jak je vidět, zvýšení počtu uzlů v hyperkrychli na dvojnásobek zvýšilo počet spojů v každém uzlu pouze o jeden, stejně tak jako průměr sítě. Lze dokázat, že n dimenzionální hyperkrychle bud mít 2^n uzlů, průměr sítě bude n stejně jako stupeň uzlu.

8.1.2.6 Komunikační strategie ve statických sítích

Komunikačními strategiemi obecně nazýváme algoritmy směrování zpráv na jejich cestě od jednoho uzlu sítě ke druhému. Složitost takového algoritmu roste jak se stupněm uzlu sítě tak s jejím průměrem a stanoven optimální komunikační strategie ve složité síti je mnohdy velmi obtížné. Nejjednodušší komunikační strategie existují v lineárních sítích, kde cesta od jednoho uzlu ke druhému je snadno popsatelná a rozhodovací algoritmus je jednoduchý. Uzavřeme-li však lineární síť a tak vytvoříme síť kruhovou, problém komunikační strategie okamžitě naroste i při tak jednoduché síti, mající pouze dvě možné cesty ke každému cíli.

Pro stanovení postupu směrování v síti se používají identifikátory (adresy) jednotlivých uzlů sítě, které nemusí nutně tvořit úplnou množinu avšak musí být snadno "štěpitelné" tak, aby směrovací algoritmus moh snadno z části identifikátoru uzlu usoudit na správné nasměrování přenášené informace. Postup směrování při binární reprezentaci identifikátoru uzlu můžeme osvětlit na příkladu komunikační strategie v hyperkrychli Jednotlivé uzly očíslujeme tak, že dva sousední uzly se ve svém číslování, resp. v jeho binární reprezentaci, budou

lišit pouze v jednom bitu. Např. uzly označené **0110** a **0100** jsou bezprostřední sousedé, zatímco uzly označené **0110** a **0101** sousedy nejsou. Nejjednodušší způsob označení jednotlivých uzlů je přiřazování jejich identifikátorů tak, jak je krychle konstruována. Jestliže kopírujeme ***n***-dimenzionální krychli, pak jednotlivé uzly a jejich kopie budou mít stejný identifikátor. V následujícím kroku rozšíříme tento identifikátor o 1 bit a to tak, že přidáme **0**, jestliže se jedná o identifikátor uzlu originálu, nebo **1** v případě, kdy se jedná o identifikátor uzlu v kopii krychle (viz obr. 101).



Obr. 101 Postup konstrukce čtyřdimenzionální krychle (hyperkrychle)

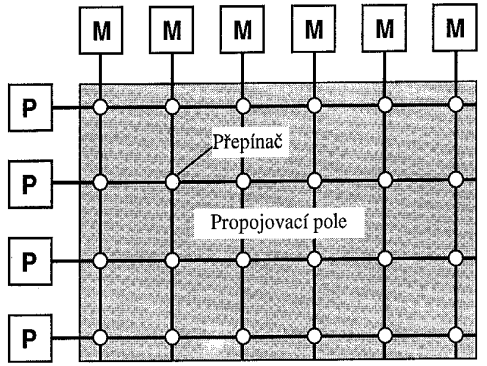
Použijeme-li výše uvedenou konstrukci pro identifikátory resp. adresaci jednotlivých uzlů, pak můžeme navrhnout i jednoduchý algoritmus sloužící ke směrování toků informací v hyperkrychli. Víme, že ***n***–dimenzionální krychle bude mít ***n***-bitové identifikátory uzlů. Vyslání zprávy z uzlu **A** do uzlu **B** může být provedeno v ***n***-cyklech, kdy během každého cyklu uzel buď obsahuje informaci, nebo ji směruje na některý ze svých spojů. V ***i***-tém cyklu pak uzel, který obsahuje příslušnou informaci, porovná ***i***-tý bit svého vlastního identifikátoru s bitem ***i*** adresy určení. Jestliže tento bit souhlasí, pak příslušný uzel zprávu uchová. Jestliže odpovídající bit je různý, pak bude zpráva směrována ve směru rozměru ***i***, kde "rozměr ***i***" je ten rozměr, která byl přidán v ***i***-tém kroku konstrukce krychle. Tento rozměr, resp. směr rozměru, je pro všechny uzly stejný.

8.1.3 Dynamické sítě

Ve složitějších sítích používáme často speciálních zařízení umístěných mezi prvky systému, která umožňují konfigurovat dočasná propojení jednotek systému. Takové sítě, v nichž se fyzická propojení přenosových cest v čase mění, nazýváme **dynamickými sítěmi**. Abychom v dynamické síti přenesli informaci z jedné jednotky do druhé, musíme tyto prvky - přepínače, nastavit tak, aby vytvořily cestu mezi zvoleným procesorem a pamětí. Informace o tom, jak se má příslušný přepínač nastavit, přichází od uzlu, který vysílá, obvykle prochází jednotlivými přepínači, kde část vysílané informace je použita k nastavení příslušného přepínače, a přichází do uzlu, který je určen jako příjemce. Velikost a počet vnitřních uzlů, které se účastní uskutečnění spoje, přispívají k režii systému a ovlivňují parametr sítě nazývaný se **nastavovací čas**. Tento čas odpovídá době potřebné k přenosu informace mezi vstupní a výstupní branou přepínače a zahrnuje dobu potřebnou k rozhodnutí o přepnutí a vlastní dobu přepnutí.

8.1.3.1 Křížové přepínače

Nejčastějším prvkem používaným ve vícestupňových sítích je **křížový přepínač** na obr. 102. Můžeme si jej představit jako spínací prvky umístěné do matice, kde na stranách jsou umístěny propojované jednotky, např. procesory a paměti. Přepínače uvnitř matice propojují odpovídající jednotky mezi sebou. Typické propojení křížovým přepínačem je na obr. 102, kde je ***n*** procesorových jednotek propojeno s ***m*** paměťovými moduly. Náklady na propojovací systém oproti nákladům na propojení typu bod-bod výrazně poklesnou za cenu latence propojovacího subsystému způsobené zpožděním v přepínacích obvodech a dobou potřebnou k nastavení cesty. Po uskutečnění spojení je však možno dosáhnout stejné šířky pásma jako u spojení bod-bod. Současně je možno sestavit **min**(*m*,*n*) propojovacích cest, neboť propojení může být realizováno vždy jen způsobem bod-bod. Jak je zřejmé z obrázku, jestliže klient(y) z procesoru bude požadovat spojení s jedním modulem paměti pak běh procesoru nebude narušen a nedojde ke kongescím. Jestliže však k jednomu modulu paměti bud požadovat přístup dva nebo více procesorů, pak všechny následující žádosti na příslušný přepínač musí bý pozastaveny do té doby, než současně propojený procesor ukončí svůj požadavek.

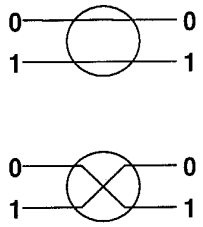


Obr. 102 Křížový přepínač

Výhodou křížového přepínače je velmi malý průměr sítě, protože informace prochází pouze jedním přepínacím elementem, síť má však velmi malou variabilitu. Existuje-li n procesorů a stejný počet pamětí, pak budeme potřebovat n2 vnitřních přepínačů. Přidání dalšího procesoru a paměti bude znamenat přidání 2n+1 vnitřních uzlů. Křížové přepínače se často používají pro propojení pamětí a procesorů multiprocesorových systémů, avšak flexibilita výstavby propojovacích subsystémů s křížovými přepínači klesá s počtem připojených jednotek. S ohledem na svoji topologii je oblíbeným prvkem ve vektorových procesorech pro propojení vektorových registrů s datovými cestami. Plně integrovaný křížový přepínač se vyskytoval např. v řad speciálních integrovaných obvodů - transputerů firmy Inmos a byl určen pro propojování transputerů v dvourozměrných mřížových sítích.

8.1.3.2 Propojovací sítě s výměnou

Cena křížového přepínače pro systémy používající tuto technologii jako základní nástroj komunikace byl limitujícím prvkem ve výstavbě. Proto konstruktéři hledali řešení, které by odstraňovalo nevýhody křížovýcl přepínačů a bylo dosažitelné za srovnatelnou nebo nižší cenu. Aby nemusela být konstrukce komunikačníh subsystému pokaždé znovu navrhována, vývoj směřoval k návrhu univerzálního elementu přepínací sítě, který b umožňoval její modulární výstavbu. Nejlépe se osvědčily dynamické sítě sestavované z elementárních přepínač typu ***m*** x ***n*** , které vhodným propojení vstupů a výstupů umožňovaly tvořit propojovací sítě s dostatečnou flexibilitou při relativně nízké ceně základního propojovacího prvku. Přepínač typu 2 x 2, jehož dva základní stavy je možno ovládat pomocí jednoho bitu, se ukázal vhodným prvkem pro konstrukce dynamických sítí vycházejících z filosofie křížových přepínačů (viz obr. 103). Z hlediska jeho funkce je možno jej rovněž považovat za elementární křížový přepínač a s jeho pomocí můžeme sestavit dvě základní sítě - propojovací síť s dokonalým promícháním (perfect shuffle) a propojovací síť s dokonalým oddělením (perfect deshuffle).



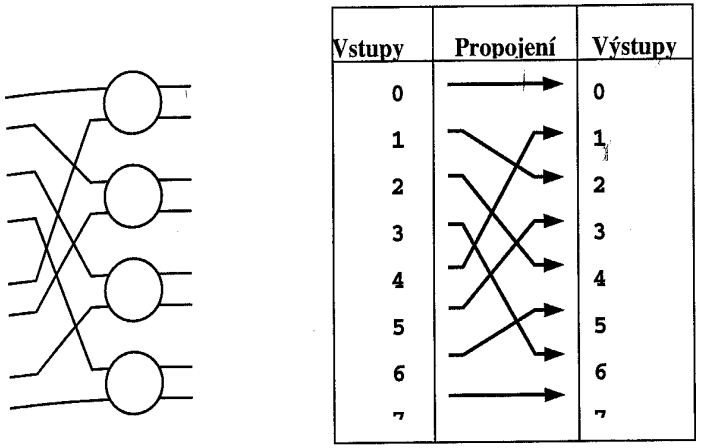
Obr. 103 Elementární přepínač 2 x 2.

Konfigurace sítě s dokonalým promícháním je na obr. 104.
Název propojovací sítě s dokonalým promícháním byl odvozen z metody míchání karet, kdy jsou karty ze dvo balíčků ve výsledném balíčku vzájemně prostřídány. Rozdělíme-li vstupní cesty přepínací sítě s dokonalýn promícháním na dvě části, pak promíchání proběhne podle schématu, v němž se na straně výstupů pravideln střídají vstupy z horní a dolní skupiny vstupů. Formálně by bylo možno popsat takové promíchání tak, ž každému vstupu a výstupu přiřadíme dvojkové číslo. Výstup, se kterým má být tento vstup spojen, bude určer rotací vlevo tohoto dvojkového čísla vstupu. Tento způsob adresace a přiřazování se snadno realizuje a proto j často používán.

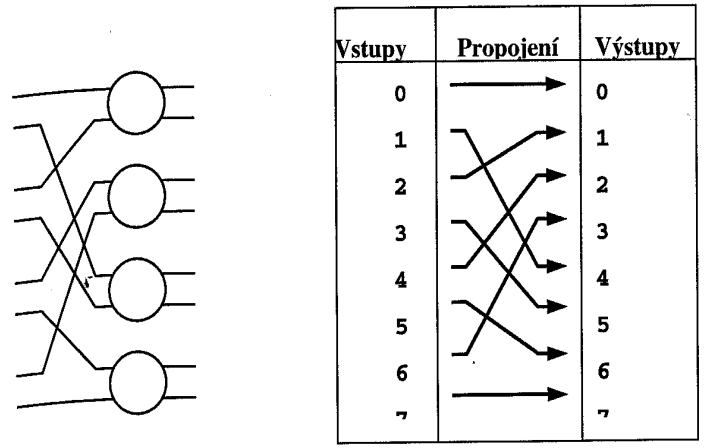
Případ přepínací sítě s dokonalým oddělením je na obr. 105, který je vlastně zrcadlovým obrazem předcho konfigurace s dokonalým promícháním. V tomto případě budou skupiny rozděleny tak, že všechny liché vstup se propojí do horní skupiny výstupů a všechny sudé do spodní skupiny výstupů. Formální metoda získání adres příslušného výstupu z adresy vstupu bude obdobná jako v předchozím případě s tím rozdílem, že rotac dvojkové adresy vstupu bude prováděna opačným směrem, tedy o jedno místo vpravo.

8.1.3.3 Víceúrovňové přepínací sítě

Použití přepínacích sítí tak, jak jsme je poznali, je již způsobem realizace omezeno tím, že signál z každého vstupu může být přiveden jenom na dva výstupy. To je s ohledem na kladené požadavky málo, a tak se v reálných konstrukcích používají sítě vícestupňové, které jsou kaskádním spojením těchto elementárních sítí. Podle toho, jaké propojení vícestupňové sítě dovolují a k jakým interakcím dochází při současných požadavcích na větší počet propojení, můžeme rozeznat tři hlavní typy sítí



Obr. 104 Konfigurace sítě s dokonalým promícháním

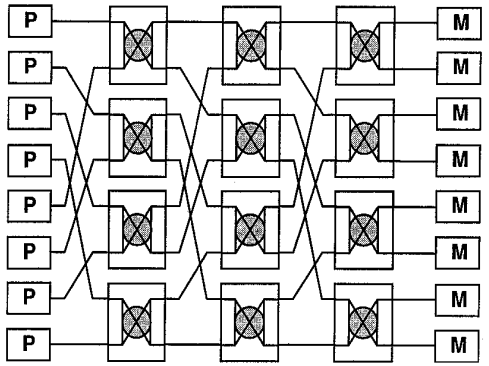


Obr. 105 Konfigurace sítě s dokonalým oddělením

- vícestupňové blokující sítě, u nichž požadované propojení jednoho vstupu s jedním výstupem, kde oba jsou dosud volné, může být blokováno existujícími spojeními ostatních vstupů a výstupů;
- vícestupňové přestavitelné sítě, které umožňují realizovat propojení všech vstupů a výstupů, ale v některých případech jenom za cenu přestavění existujících propojení;
- vícestupňové neblokující sítě jsou nejdokonalejším propojovacím prvkem, avšak také nejdražším, a umožňují vytvoření spojením mezi dvěma libovolnými body bez ohledu na to, jaké spojení již bylo realizováno.

Mezi nejznámější vícestupňové blokující propojovací sítě patří síť Omega, která je sestavena z elementárních přepínačů 2x2, a její jednotlivé úrovně jsou propojeny promícháním potřebného počtu prvků. Počet úrovní této sítě typu n x n je $\log_2 n$ a počet přepínačů v jedné úrovni je $n/2$. Příklad sítě pro n = 8, kter propojuje osm procesorů s osmi jednotkami pamětí, je na obr. 106. Velmi často se setkáváme s vícestupňovým blokujícími sítěmi, které mají strukturu spojů podobnou pravidelně propleteným větvím, jež v prvním stupn propojují dvojice, ve druhém čtveřice atd. a jsou, označovány souhrnným názvem **banyanové[1] sítě**. Zajímavo vlastností blokujících sítí je, že jsou topologicky ekvivalentní, tedy lze vždy nalézt vhodnou transformaci, kterou převedeme jednu blokující síť na druhou.

Charakteristickou vlastností přestavitelných sítí je existence několika možných spojení mezi dvěma body sítě, z nichž si při zcela neobsazené síti můžeme vybrat. Pokud ovšem požadavek na sestavení spojení přijde později a původně vybrané spojení v jeho realizaci překáží, musíme síť přestavit. Nejznámějším typem přestavitelných sítí je **Benešova síť**, ve které vedou na každý výstup čtyři možné cesty a síť je sestavena ze základní sítě a jejího zrcadlového obrazu. Pokud při novém požadavku na sestavení cesty v Benešově síti nelze spojení zajistit, je nutno přestavit některé z existujících spojení, a tak uvolnit žádanou cestu.



Obr. 106 Příklad sítě typu Omega pro n=8

Poslední typ sítí - **neblokující sítě** vykazují takové množství variant propojení, že je možno vždy najít volnou cestu k propojení vstupního uzlu s výstupním. Jejich realizace se provádí většinou pomocí přepínačů **n** x **m** a mnohdy jsou interpretovány jako víceúrovňové křížové přepínače. Neblokující sítě jsou složité a drahé, což vede k jejich velmi omezenému používání. Jejich charakter však umožňuje vytvářet specializované jednotky pro vzestupné nebo sestupné řazení položek přiváděných na vstupy pomocí informací přicházejících do přepínačů.

Speciálními typy dynamických sítí jsou válcové posouvače (barrel shifter), založené na přepínačích typu 3 x 3. Tento typ dynamické sítě je dnes téměř samozřejmou součástí nejen většiny výkonných procesorů, ale i aritmetických jednotek mikroprocesorů, neboť umožňuje rychle provádět některé permutace vstupů a předávat je na výstup. Funkce jednoúrovňového válcového posouvače může být charakterizována operací posunu (rotace) vstupů o n míst vpravo nebo vlevo, kde n je typická charakteristika pro jeden stupeň válcového posouvače. Pokud chceme docílit většího posunu, pak musíme používat vícestupňových válcových posouvačů nebo několika průchodů.

8.1.3.4 Strategie řízení dynamické sítě

U propojovacích obvodů dynamických sítí jsou důležité tři základní strategické atributy:

- řídící strategie,
- strategie časování,
- strategie přepínání.

Pro **řízení přepínací sítě** existují dvě základní alternativy - buď jeden centrální řadič, nebo distribuované řízení, kde strategie směrování zprávy je implementována v každém uzlu. Příkladem distribuovaného řízení je směrování zprávy založené na identifikátoru uzlu u sítí typu hyperkrychle nebo motýlových sítí, protože každý uzel rozhoduje sám, zda použije nebo přesměruje přicházející informaci. Naopak příkladem centrální strategie je síť s topologií hvězdy. Zprávy ve hvězdicové topologii přicházejí z vnějších uzlů, musí projít centrálním uzlem a ten je jediným, jenž rozhoduje o směrování této zprávy.

Časování nebo též synchronizace řízení přepínací sítě je u synchronních řídících technik zajišťována použitím globálních hodin, které jsou přivedeny ke všem zařízením v systému. Celý systém pak pracuje v

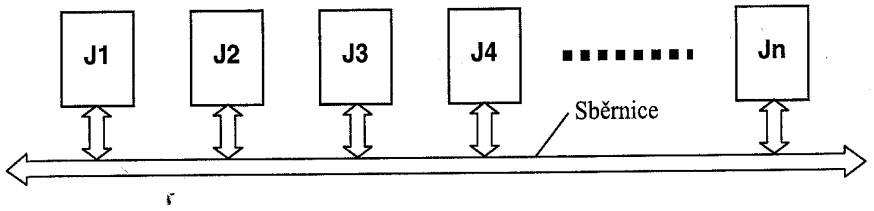
[1] Název této sítě je odvozen od jména indického fíkovníku, protože struktura sítě připomíná jeho větve a kořeny.

jednotném rytmu. Na rozdíl od tohoto přístupu asynchronní techniky distribuují řídicí funkce uvnitř systému a jednotlivá zařízení pro svoji vlastní práci používají své vlastní hodiny. Řízení a koordinace jednotlivých částí systému se dosahuje prostřednictvím výměny synchronizačních zpráv nebo pomocí zvláštních synchronizačních signálů. Výhoda jednotných hodin v případě synchronního systému spočívá ve zjednodušení jak hardware, tak i software, výhoda distribuovaného řízení spočívá ve větší možnosti expanze a vyšší flexibilitě. Synchronní nebo asynchronní časování patří mezi základní charakteristiky každého výpočetního systému.

Přepínání určuje postup, který bude sítí zvolen při přenosu zprávy. Jedna z možností je, že dynamická síť je nejdříve nastavena od příslušného vstupu k příslušnému výstupu a následovně se uskutečňuje komunikace mezi dvěma propojenými uzly. Sestavení spoje trvá do té doby, než se jeden z uzlů rozhodne relaci ukončit. Důvodem k ukončení může být např. přenesení celé zprávy. Tento způsob nazýváme přepínáním okruhů a setkáváme se s ním zejména v oblasti počítačových sítí. Stejně tak druhý způsob, kdy je zpráva rozdělena do menších částí, nazývaných **pakety**, které jsou přenášeny tak, že mezilehlý uzel si paket zapamatuje a po analýze jej předá příslušnému následujícímu uzlu, patří dnes k technikám běžně používaným v počítačových sítích. Důvodem k tomu, že uzel se rozhodne paket uložit ve své lokální paměti předtím, než jej odešle na další místo určení, může být, že jsou zde jiné pakety, čekající na odeslání po stejném spoji. Pakety se rovněž mohou šířit po různých spojích na své cestě od zdroje informace k místu určení a vykazovat různá zpoždění způsobená každým přepínacím bodem, závislá zejména na celkovém provozním zatížení sítě.

8.2 Sběrnicové organizace

Sběrnicové organizace mají vzhledem ke směrovým organizacím celou řadu výhod, jak z hlediska realizace, tak i funkce, které přispěly k jejich obecnému rozšíření. Jsou používány k přenosu informace mezi několika různými jednotkami systému. V nejjednodušším případě jsou všechny jednotky propojeny jednou společnou sběrnicí podle obr. 107 a šířka datových cest, tedy šířka sběrnice, bývá poměrně velká. U výkonných systémů je používána šířka sběrnice 64 bitů i více, avšak jsou k dispozici i jednobitové, tzv. sériové sběrnice. Jednoduchou sběrnici, která propojuje všechny hlavní komponenty, používají většinou malé a střední počítačové systémy. Avšak i superpočítače a ostatní výkonné stroje, které mají mnohem složitější systém propojení, velmi často používají vnitřní sběrnice k propojení některých součástí systému.

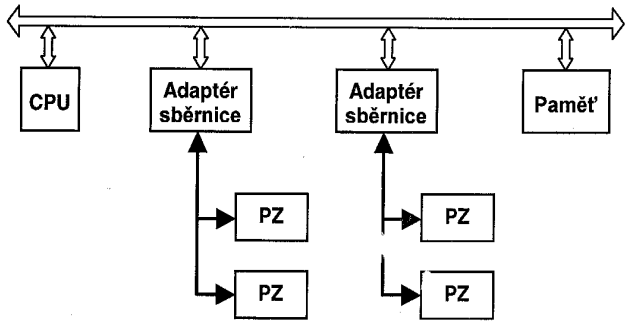


Obr. 107 Jednoduchý sběrnicový systém propojující jednotky J1 až Jn

Komunikace na sběrnici je rozdělena do jednotlivých diskrétních transakcí. Každá transakce je realizována mezi vysílačem a přijímačem. Aby taková transakce mohla započít, některý z výše uvedených účastníků transakce musí získat kontrolu nad sběrnicí a stát se řídící jednotkou, tzv. **masterem**. Velice často existuje více zařízení, která mají možnost řídit práci sběrnice. Např. procesor může řídit přenos instrukcí a dat mezi pamětí a CPU, avšak řadič disku převezme řízení sběrnice v okamžiku, kdy bude nutno přenést blok dat mezi diskem a pamětí. Pokud vznikne požadavek od dvou nebo více zařízení přenášet informaci ve stejnou dobu, musí existovat **arbitrážní protokol,** jenž pomůže rozhodnout, kterému z těchto zařízení bude předáno řízení nejdříve. Tento protokol se sestává z celé řady signálů vyměňovaných mezi jednotlivými moduly připojenými na sběrnici, kde výsledkem této výměny je rozhodnutí, které zařízení se v následujícím okamžiku stane řídící jednotkou sběrnice.

Ve chvíli, kdy zvolené zařízení převezme řízení sběrnice, používá k přenosu informace **komunikační protokol** sběrnice. V zásadě je možno mluvit o **asynchronních protokolech**, kdy přenos jednotlivého elementu informace může začínat kdykoliv. V tomto případě je nutno počítat s přídavným časem na přenos informace inicializační informace, ve kterém potenciální příjemce informace detekuje skutečnost, že v následujícím okamžiku k tomuto přenosu dojde. Na rozdíl od tohoto typu přenosu **synchronní protokoly** používají diskrétních okamžiků, ve kterých může začít přenos informace. Jak asynchronní, tak zejména synchronní protokoly jsou řízeny základním taktovacím kmitočtem sběrnice.

Výkon sběrnicového systému je definován dvěma parametry - přenosovým časem a celkovou šířkou pásma (často nazýván prostupností). **Přenosový čas** je podobný parametr jako latence v paměťových systémech. Je to doba, potřebná k dodání dat od zdroje dat k přijímači dat během jedné transakce. Například přenosový čas bude definovat, jak dlouho procesor bude čekat, než bude vyzvednuta instrukce z paměti. **Šířka pásma** se obvykle udává v jednotkách bitů za sekundu a odpovídá kapacitě příslušné sběrnice. Je definován jako součin počtu bitů, které mohou být přeneseny paralelně v kterémkoliv směru během jedné transakce a počtu transakcí za sekundu. Například sběrnice, která má 32 datových vodičů a může přenášet po jednom vodiči 1 milion bitů za sekundu, má šířku pásma 32 Mb/s.



Obr. 108 Dvouúrovňový hierarchický systém sběrnic

Na první pohled to vypadá, jako kdyby tyto dva parametry vyjadřovaly tutéž míru, ale je zde malá diference. Doba přenosu vyjadřuje zpoždění v dodání příslušné části dat přijímači. Jakmile jsou data dodána, mohou být příslušným modulem použita, zatímco ostatní signály na sběrnici dokončují komunikační protokol. Dokončení tohoto protokolu zase zpozdí následující transakci a právě šířka pásma bere v úvahu tato dodatečná zdržení. Jiný faktor, který může být použit pro rozlišení těchto dvou parametrů, může být pozorován u systému s vysokou výkonností, kde během jedné transakce může být přenesen celý blok informací. Jinými slovy, komunikační protokol může realizovat operaci "pošli n položek z místa x". Tato operace bude požadovat jistou dobu na inicializaci celé transakce, takže zde bude delší zdržení před přijetím první části dat, ale následovně budou data předávána daleko rychleji. Šířka pásma je důležitý parametr, který může být použit k popisu výkonnosti procesoru, když budeme počítat počet instrukcí, které mohou být vykonány v jednotce času, stejně tak jako k popisu výkonnosti sítě.

8.2.1 Sběrnicové hierarchie

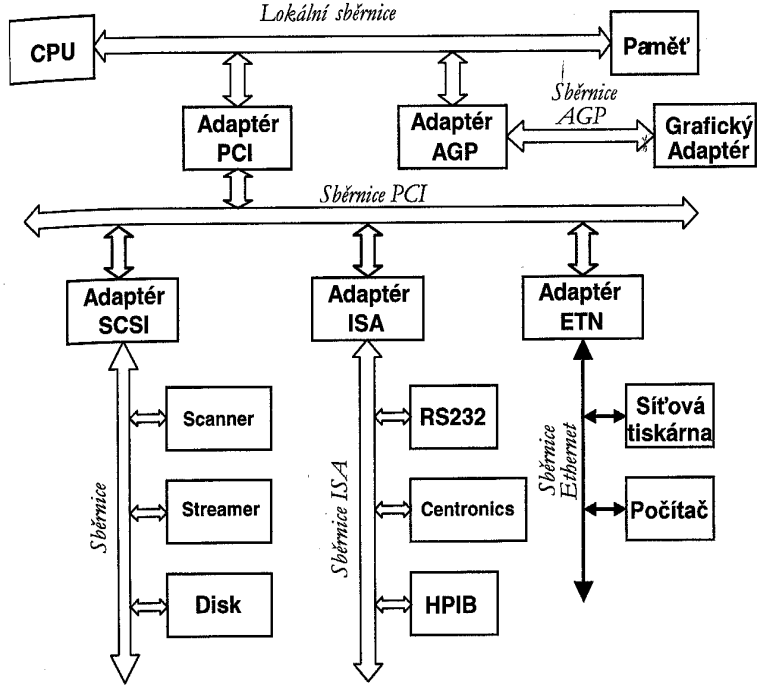
Jednoduché sběrnice jako na obr. 107 jsou použitelné pouze pro jednoduché systémy s malým výkonem. Se zvyšujícími nároky na výkon systému vzrůstají i požadavky na přenosový výkon sběrnice neboli na její šířku pásma. Protože ta je v mnoha případech omezena fyzikálními možnostmi použité technologie, používají se sběrnicové systémy sestávající se z paralelních sběrnic nebo sběrnic hierarchicky uspořádaných. Paralelní sběrnicový systém používá nezávisle řízené sběrnice tak, že je-li požadovaná sběrnice obsazena, pak pro spojení se použije jiná dostupná sběrnice. Přidělování jednotlivých sběrnic řídí zvláštní algoritmus tak, aby jejich zatížení bylo rovnoměrné. Stejně tak mohou být sběrnice uspořádány hierarchicky, kdy sběrnice v hierarchii nižší (a obvykle i pomalejší) je propojena ke sběrnici v hierarchii vyšší pomocí speciální jednotky, tzv. mostu.

Příklady realizace jednotlivých typů sběrnic jsou na obr. 107 až obr. 109. V prvním případě jde o klasické řešení pomocí jedné sběrnice, kde připojené jednotky tuto sběrnici sdílejí. Klasickým případem jednosběrnicového systému jsou jednoduché řídicí aplikace s mikroprocesory nebo sběrnice osobního počítače v základním provedení AT. Přidáním jednotky propojení můžeme sběrnici rozšířit a připojit další sběrnicový systém nižší hierarchie - např. sběrnice SCSI (viz dále) pro připojování periferních zařízení k procesorové

[1] U asynchronních protokolů je možné taktovací kmitočet nahradit výměnou řídicích signálů.

sběrnici. Takovým způsobem je možné [illegible] vlastnosti.

Příkladem většího sběrnicového systému je organizace propojení jednotek v moderním personálním počítači, který používá lokální sběrnici pro propojení procesoru a pamětí, k ní pomocí mostů jsou připojeny sběrnice PCI a AGP a na ně pomocí adaptérů další paralelní sběrnice např. SCSI nebo ISA nebo sériové sběrnice např. Ethernet LAN nebo USB.



Obr. 109 Hierarchický systém sběrnic personálního počítače

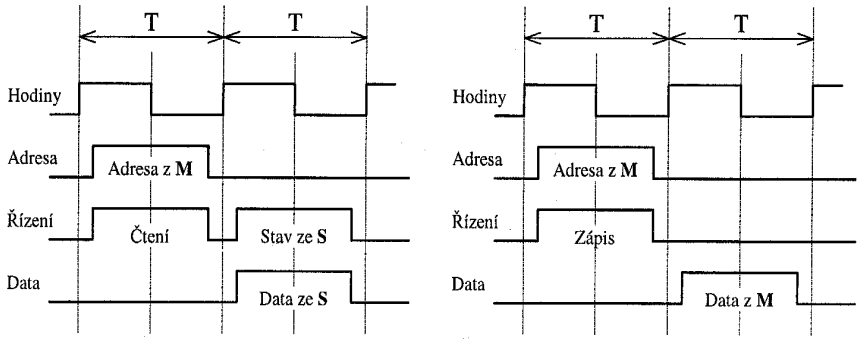
8.2.2 Synchronní a asynchronní přenos

Děje při přenosu dat po sběrnici mohou mít společný zdroj synchronizačního kmitočtu nebo mohou být synchronizovány pomocí zvláštních zpráv, které si mezi sebou dotčené jednotky vyměňují. V definicích sběrnic obvykle používáme značení řídícího uzlu jako **master (M)** a řízeného uzlu jako **slave (S)**, nebo se orientujeme podle směru přenosu dat a dotčené jednotky účastnící se přenosu označujeme jako **zdroj** informace (source - **S**) a **příjemce** informace (acceptor - **A**).

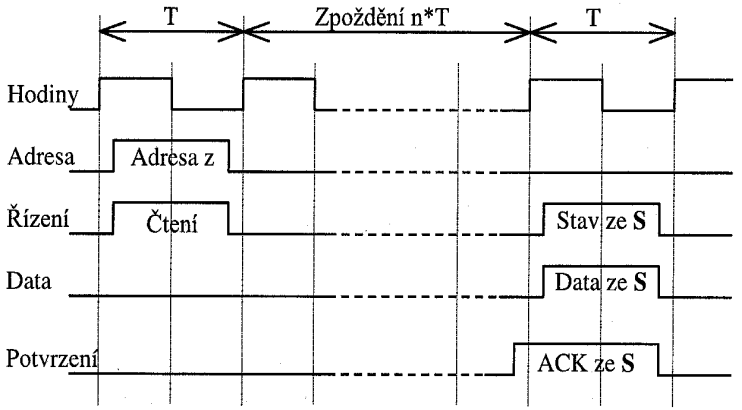
Základem práce synchronní sběrnice je jeden cyklus, který je definován jako celistvý násobek hodinových impulsů. Časové diagramy práce zařízení na sběrnici potom vypadají tak, jako na obr. 110. Jak je vidět, jednotky, které používají synchronní sběrnici, musí komunikovat stejnou rychlostí. Synchronizace zařízení komunikujících různými rychlostmi je možno dosáhnou pomocí vkládání tzv. čekacích cyklů (wait state). Tím se dosáhne zpomalení práce rychlejší jednotky a přizpůsobení rychlostí. Protože doba strávená v čekacích cyklech je neproduktivní, je snaha tyto čekací cykly omezit (např. na maximálně 10). Pokud pomalejší zařízení neodpoví ani po uplynutí čekací doby, dojde ke stavu, kdy vyprší čekací lhůta a celý přístupový cyklus je řídící jednotkou zrušen. Potvrzení připravenosti ze strany pomalejší jednotky přitom musí přijít před započetím vlastního cyklu čtení nebo zápisu.

Použití sběrnice pro komunikaci jednotek, jejichž rytmus práce se bude výrazně lišit, nebude zřejmě vhodné. Počet čekacích cyklů by se neúměrně zvyšoval zejména s ohledem na tendenci použití co nejvyššího nutné, aby byla co nejkratší a tím vzniknou i konstrukční problémy[1]. Proto se zejména pro připojová periferních zařízení používají asynchronní sběrnice, které pro řízení využívají speciálních signálů.

Asynchronní přenos může být řízen buď jednostranně, tedy buď zdrojem informace nebo jejím příjemce nebo i oboustranně, kdy se na řízení přenosu podílejí oba uzly. Případ asynchronního jednostranně řízené přenosu je v časovém diagramu na obr. 112. V obou případech, jak při řízení zdrojem (b), tak při říze příjemcem (a) dochází k převzetí dat na základě potvrzení jejich platnosti[2] při řízení zdrojem nebo poskytnutí c po vydání žádosti o nová data. Tento druh přenosu však není schopen rozlišit mezi správným a špatný převzetím dat, neboť nemá žádnou zpětnou signalizaci (pokud tato signalizace není zajištěna novým čtení např. stavového slova). Ke špatnému převzetí může dojít, pokud se příjemce opozdí, má poruchu či z jiný příčin.



Obr. 110 Časový diagram práce synchronní sběrnice při čtení a zápisu



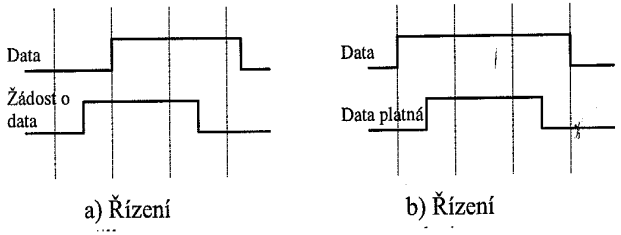
Obr. 111 Čtení na synchronní sběrnici s vkládáním čekacích cyklů

Pro odstranění této nevýhody se doplňuje do řídících signálů asynchronní sběrnice další signál - potvrzer správného převzetí - ACK (acknowledge). Tím dostáváme oboustranně vyvážený korespondenční protoko který se také nazývá **handshake**. Opět je možno rozpoznat dva případy řízení - přenos vyvolaný zdrojer informace a přenos vyvolaný příjemcem informace. Oba tyto případy jsou pomocí časových diagramů ukázán

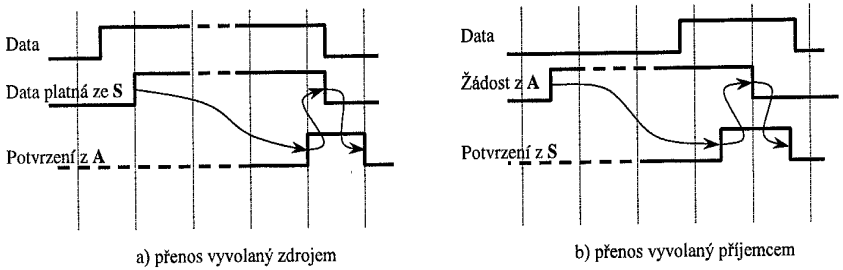
[1] Pro větší délky sběrnice se projeví rozptyl náběžných a sestupných hran signálů jednotlivých paralelních cest (tzv. skew). Pro eliminaci tohoto jevu je nutno vkládat bezpečnostní doby mezi jednotlivé hrany a to vede opět ke zpomalení sběrnice.

[2] Signál potvrzení platnosti dat se někdy označuje jako výběrový signál - "strobe".

na obr. 113. Za okamžik komunikace mezi jednotkami je vždy považována hrana impulsu tak, jak je to naznačeno v diagramech šipkami.. V případě, kdy výměnu informací inicializuje zdroj informace, nastaví na sběrnici platná data a označí je pro příjemce jako platná. Po potvrzení o přijetí dat příjemcem je platnost dat zrušena a data ze sběrnice jsou odstraněna. Pokud bude inicializován přenos informace příjemcem, musí příjemce o data požádat. Na základě této žádosti nastaví zdroj dat data na sběrnici a označí je řídícím signálem sběrnice jako platná. Jakmile jsou platná data převzata příjemcem (signál žádost o data je shozen), zdroj dat zruší signál platnosti dat na sběrnici a odstraní z ní data.



Obr. 112 Řízení asynchronní sběrnice - jednostranné

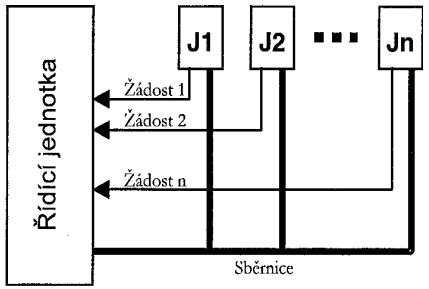


Obr. 113 Řízení asynchronní sběrnice - oboustranné

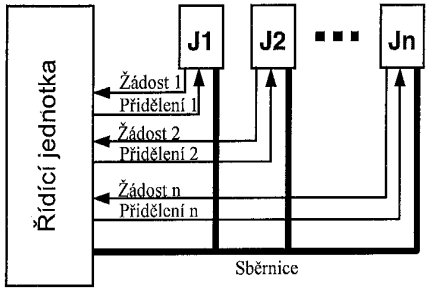
8.2.3 Arbitráž požadavků o sběrnici

Sběrnice je sdílené přenosové médium, a proto je nutno předpokládat, že může dojít k situaci, kdy bude o použití této přenosové cesty soupeřit najednou několik jednotek, které jsou ke sběrnici připojeny. V tom případě bude nutno mít k dispozici rozhodovací mechanismy, které určí jednotku, jíž bude sběrnice prioritně přidělena. Základní úlohou přitom bude rozhodování - arbitráž, vycházející z několika základních algoritmů. Principy rozhodování při přidělování sběrnice můžeme shrnout následovně:

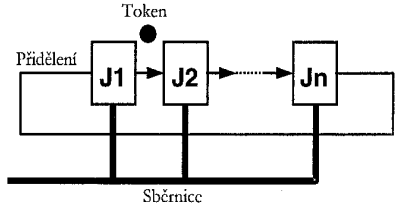
- centrální arbitráž, kdy je použita zvláštní jednotka pro vyhodnocení přicházejících žádostí a ta může požadavky vyhodnocovat podle
  - priority, kdy každému přicházejícímu požadavku je přiřazena priorita, většinou podle toho od jakého typu jednotky žádost přichází. Výsledkem je přidělení sběrnice jednotce s nejvyšší prioritou
  - pořadí příchodu - FIFO, kde jsou přicházející požadavky uspořádány do fronty a sběrnice je přidělována podle pořadí příchodu požadavků,
  - náhodné, při kterém jsou přicházející požadavky opět ukládány do paměti řídící jednotky a z ní náhodně vybírány. Sběrnice je potom přidělena jednotce, jejíž požadavek byl náhodně vybrán.
- decentralizovaná arbitráž, která staví na decentralizaci rozhodovacích prvků a vlastní rozhodování se děje na základě algoritmů vestavěných v každé připojené jednotce. V tomto případě je nutno předpokládat jak vyšší inteligenci připojovaných zařízení, tak i nezbytný čas na provedení arbitrace a výměnu zprávy o jejím výsledku mezi připojenými jednotkami. Obvykle se používá
  - prioritní linka (daisy chain) - priorita zařízení je dána jejich pořadím v řetězci vzájemně sériově propojených jednotek,
  - cyklické, používá zvláštní zprávy, nazývané token[1], která obíhá mezi připojenými jednotkami a je okamžitá pozice určuje přidělení sběrnice jednotce, která právě token vlastní
  - kolizní, spočívající v detekci kolize přístupu na sběrnici a zabudovaných algoritmech jejího řešení.



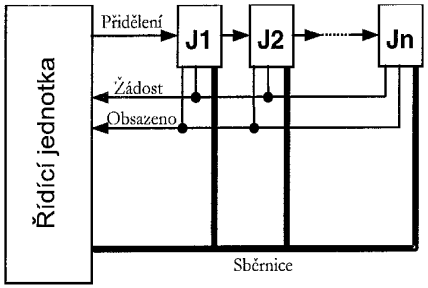
Obr. 114 Centrální řízení s výzvami



Obr. 115 Centrální vyzývání s nezávislými vodiči



a) decentralizované řízení linky



b) centralizované řízení linky

Obr. 116 Přidělování prioritního přístupu na sběrnici - metoda daisy chain

Některé příklady centrálního i decentralizovaného řízení jsou na obr. 114 až obr. 116. V těchto případech j pro řízení sběrnice rozhodující arbitrážní algoritmus vestavěný ve speciální řídící jednotce sběrnice. Na obr. 11 je centrální vyzývání s vyzývací adresou, které realizuje prioritní mechanizmus tabulkou pořadí výzev. Řídí jednotka postupně čte adresy jednotlivých zařízení připojených na sběrnici a dotazuje se jich, zdali nepožadu přístup na sběrnici. Ve chvíli, kdy narazí na jednotku, která má aktivní žádost o sběrnici, aktivuje se příslušn signál žádosti a řídící jednotka přidělí sběrnici právě adresované jednotce. Adresa jednotky nebo její vyzývací čísl může být předáváno po datové, adresové nebo speciální sběrnici.

Druhou možností tohoto uspořádání je, že řídící jednotka kontroluje pouze signály žádostí o sběrnici tak, ž se dotazuje na jejich hodnotu ve stanoveném pořadí. Pokud nalezne platnou žádost o přidělení sběrnice, přidělí té jednotce, jejíž žádost akceptovala. Priority jsou opět stanoveny pořadím výzev.

Metoda centrálního vyzývání s nezávislými vodiči opět používá speciálních vodičů pro žádost o přidělen sběrnice, ale její přidělení je realizováno signálem, který je zvlášť přidružen každé jednotce. Sběrnice v tomt případě je použita pouze pro přenos platné adresové a datové informace.

Řízení pomocí prioritní linky využívá pořadí propojení jednotek, které jsou zvláštním vodičem propojen do řetězce. Priorita je pak dána pořadím zapojení jednotek v řetězci a centrální řídící jednotka pouze řídí časov sledy přidělování sběrnice. Vlastní rozhodovací proces je decentralizovaný. Pro rychlou identifikaci zdroje žádos a případný výběrový režim obsluhy je možno předávat žádosti po více vodičích se zakódovanou adreso

[1] Token - anglický termín pro hrací žeton, používá se k označení zvláštní zprávy - značky.

s nejvyšší prioritou. Centrální člen je možno vypustit, pokud zajistíme vhodný způsob řízení celé soustavy. To může být zajištěno např. tím, že jednotky si mezi sebou předávají token, který určuje jednotku, jež má v daný okamžik právo přidělení sběrnice. V mnoha případech splývá datová sběrnice se sériovou linkou pro předávání tokenu a nabývá podoby kruhové sítě.

Každý sběrnicový systém má své typické parametry, které určují jeho použití s ohledem na očekávaný výkon celého počítačového systému. Základním parametrem je doba čekání na přidělení sběrnice, která závisí na

- době obsazení sběrnice, která se odvozuje z počtu přenášených datových jednotek v bloku dat,
- době čekání na ukončení arbitráže,
- době šíření signálů po sběrnici.

Zvětšení přenosových rychlostí sběrnice, a tím i výkonu celého propojovacího systému lze dosáhnout např.

- oddělením datové a adresové části sběrnice, neboť multiplexování adres a dat vede ke zpoždění předávání dat. Používá se pouze u levnějších systémů.
- zvětšením datové šířky sběrnice, např. na 64 bitů,
- zvýšením hodinového kmitočtu se současným zmenšením fyzických rozměrů sběrnice a zlepšení jejich elektrických vlastností.

Právě omezení, daná fyzikálními vlastnostmi materiálů vedou k nutnosti paralelizovat práci sběrnic, vytvářet široké paralelní sběrnice, které umožňují předat množství informace v jednom kroku.

8.2.4 Příklady sběrnic v komerčních systémech

Pro snazší představu nakolik je složitá definice sběrnice i její práce uveďme příklady dvou sběrnic používaných v personálních počítačích. První je případ rychlé paralelní sběrnice určené pro zvýšení propustnosti systému při grafických operacích - sběrnice AGP - Accelerated Graphic Port, zatímco druhá je sériová sběrnice určená pro periferní zařízení USB - Universal Serial Bus.

8.2.4.1 Paralelní sběrnice AGP

Paralelní sběrnice AGP[1], navržená firmou Intel, vznikla jako odezva na stále se zvyšující nároky na šířku pásma sběrnic připojujících grafické displeje, jenž se zejména projevovaly při 3D grafice. Zároveň byl sledován cíl snížit nároky na lokální paměť přesunutím některých datových struktur do hlavní paměti.. Standard AGP vychází ze standardu PCI[2] a rozšiřuje jej o

- proudové zpracování paměťových operací, což vede ke snížení latence paměti,
- demultiplexaci adresové a datové sběrnice, což dovoluje až 100% zvýšení účinnosti přenosů a
- nové elektrické rozhraní spolu s možností realizovat až čtyři datové přenosy během cyklu sběrnice, což dovoluje maximální šířku pásma sběrnice AGP až 1 GB/s.

Tato rozšíření standardu PCI používají speciálních signálů ke zvýšení výkonu sběrnice a jsou zcela na stávajícím připojení PCI nezávislá., což je zřejmé i z obr. 109. Nezvyklé pro sběrnici AGP je, že je použita pouze pro propojení bod-bod (grafický řadič - hlavní paměť), a tedy její sběrnicový charakter se poněkud vytrácí.

Standard AGP předpokládá dva základní modely použité pro 3D grafiku, které popisují způsob, jak jsou data přenášena mezi hlavní pamětí a lokální pamětí a jak jsou data mezi tyto dvě paměti rozdělena. V modelu DMA (viz dále kap. 8.3.3) je primární grafickou pamětí lokální paměť grafického adaptéru, která je spojena s akcelerátorem. Bloky dat jsou přenášeny mezi hlavní pamětí a lokální pamětí desky pomocí mechanizmu

[1] Zkratka pro **A**ccelerated **G**raphics **P**ort

[2] Zkratka pro **P**eripheral **C**omponent **I**nterconnect bus, standardní lokální sběrnici zavedenou firmou Intel a používanou zejména v personálních počítačích. I když PCI je 64-bitová sběrnice, obyčejně je implementována jenom pro šířku 32 bitů.

DMA. Takto je provoz na sběrnici AGP tvořen dlouhými sekvenčními přístupy při přenosu dat z hlavní pamě do lokální grafické paměti a obecně není citlivý k nespojitému paměťovému prostoru.

Ve druhém případě je akcelerátor používán pro obě paměti, lokální i hlavní, a obě paměti jsou chápány jak logicky ekvivalentní datové struktury, které však nejsou z hlavní paměti kopírovány do lokální paměti, a zpracovávány na místě, kde jsou uloženy. Tento druhý přístup naopak vede ke krátkým přístupům na více mér náhodné adresy a je velmi citlivý na nespojitosti v hlavní paměti.

8.2.4.2 Sériová sběrnice USB

Nedostatek vhodné sběrnice pro jednoduché připojování periferních zařízení byl asi hlavní motivací návrh sběrnice USB[1]. Široká možnost jejího použití a orientace na zlepšené komunikační vlastnosti jsou asi hlavním rysy, které tuto sběrnici uvedly do širokého použití. Základními charakteristikami sběrnice USB jsou

- jednoduché rozšiřování periferií připojitelných k personálnímu počítači,
- levné řešení zajišťující přenosy až do rychlosti 12 Mbit/s,
- plná podpora pro zpracování dat v reálném čase pro hlas, audio a komprimované video,
- flexibilita protokolu a asynchronní charakter přenosu.

Sběrnice USB, tak jak je fyzicky stavěna, by mohla být omylem považována za stromovou směrovo organizaci, ale vzhledem k elektrickému propojení všech zařízení prostřednictvím hubů do jednoho řídícího bodu - USB hostu, je logická struktura připojení sběrnicová.

Celá sběrnice je řízena z USB hostu výzvami a USB host je vždy tou jednotkou, která přenos dat po sběrnic inicializuje. Veškeré transakce na sběrnici probíhají jako výměna tří paketů. USB host odešle podle svéh rozpisu výzev na sběrnici zvláštní paket - token, který obsahuje informace o druhu a směru přenosu, adres zařízení USB a další informace. Zařízení, které je adresováno se aktivuje a v následující transakci přenese dat daným směrem, podle toho jak bylo specifikováno v tokenu. Tento paket může buď obsahovat data, neb informaci o tom, že zařízení nemá co přenášet. Příjemce informace, který byl specifikován v tokenu, odpoví n tuto transakci potvrzovacím paketem.

Tímto mechanismem jsou jednotlivá zařízení po připojení na sběrnici konfigurována USB hostem protokol umožňuje rozlišit, zda se jedná o zařízení splňující nějakou funkci nebo o hub.

8.3 Způsoby programové obsluhy propojení

Propojení jednotek počítače, ať už pomocí směrového nebo sběrnicového systému, bude vyžadova odpovídající programovou obsluhu. Tato obsluha v případě srovnatelných rychlostí obou jednotek - řídící řízené - může mít podobu synchronní komunikace, a potom bude pro programátora transparentní. Jinak tom však bude v případě výrazného rozdílu rychlostí mezi oběma jednotkami. Pak je nutno vyřešit vzájemno spolupráci tak, aby rychlejší řídící jednotka byla minimálně zatížena operacemi s pomalejší jednotkou, přizpůsobit programovou obsluhu i odpovídající mechanismy spolupráce. Můžeme rozlišit mezi čtyřm základními způsoby práce:

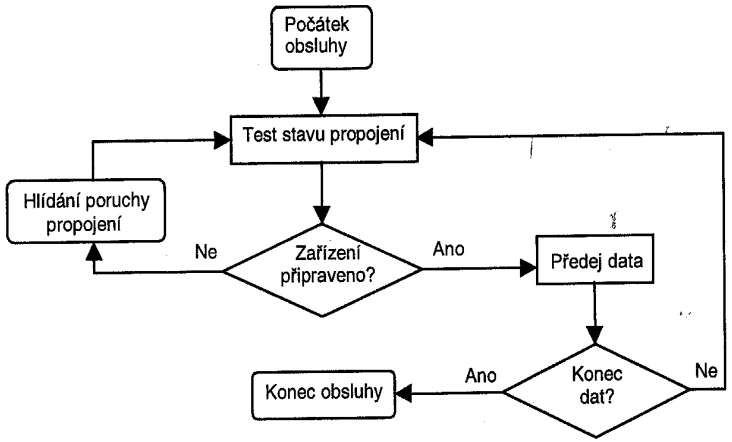
- programová obsluha,
- obsluha s využitím přerušení,
- použití přístupu DMA,
- samostatné procesory (kanály).

Jednotkou, která bude ve velké většině případů čekat, bude procesor a s ohledem na tuto skutečnost se v dalším výkladu omezíme na spolupráci procesoru a připojené jednotky. Ta však může být rozšířena i na spoluprác obecně chápaných jednotek.

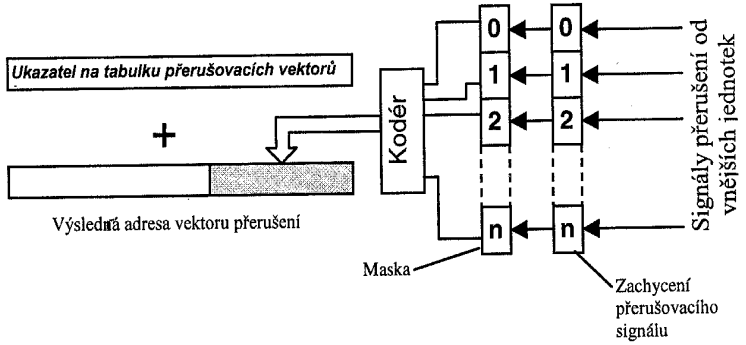
[1] Zkratka pro **U**niversal **S**erial **B**us

8.3.1 Programová obsluha

Typická posloupnost instrukcí prováděných při programové obsluze propojení bude sestávat z testování stavu druhé strany, tedy bude se řídit algoritmem



ve kterém je testování stavu propojení obvykle zajištěno zvláštním signálem od komunikujícího zařízení. V některých případech je tento zvláštní signál nahrazen obsahem slova, které udává stav komunikujícího zařízení - **stavového slova**. Stav připravenosti k přenosu dat se přitom mění podle průběhu operace - při přebírání dat přejde do stavu, kdy další data odmítne a po převzetí dat se navrátí do stavu připravenosti. Stavové slovo může obsahovat více příznaků, kterými lze odlišit stav zpracovávání převzatých dat od poruchového stavu nebo od stavu, kdy se komunikující jednotka dožaduje pozornosti obsluhy.

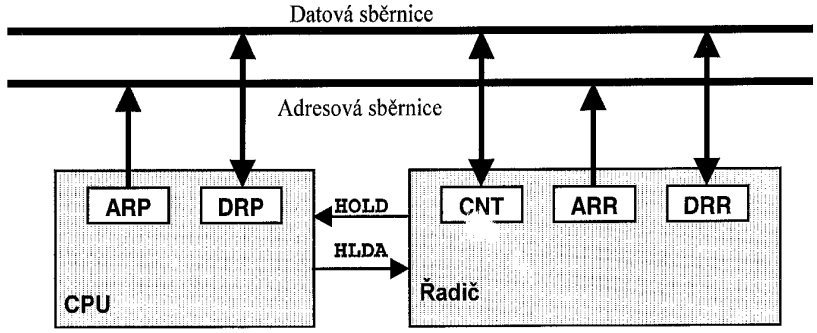


Obr. 117 Princip získání adresy vektoru přerušení

8.3.2 Obsluha s využitím přerušení

Koncepčním zdokonalením programové obsluhy je tzv. **přerušení**, kdy procesor přeruší svoji práci na vnější popud a věnuje se obsluze komunikace s propojenou jednotkou. Vnitřní mechanismy přerušení jsou obvodově řešeny nejrůznějšími způsoby, avšak jejich efekt je vždy stejný - poskytnutí mechanismu pro odložení právě probíhajícího programu a spuštění speciálního programu, který provede nezbytné kroky pro obsluhu propojení. Obvyklým způsobem vyvolání těchto speciálních programů je nucený skok na adresu, kde je tento program uložen, pomocí tabulky vektorů, ukazujících na specifické obsluhy přerušení. Každému vnějšímu přerušení je při jeho analýze hardwarem přiřazena váha, která odpovídá umístění vektoru v tabulce vektorů přerušení, a tento vektor se pak chápe jako ukazatel na příslušnou obsluhu přerušení. Je důležité, aby programový úsek obsluhující přerušení neměl žádné vedlejší efekty na přerušený program a aby by reentrantní, což znamená, aby mohl být sám přerušen novým přerušením s větší váhou. Jakým způsobem bude tento programový modul přerušení zpracovávat záleží na konkrétní propojené jednotce, která přerušení způsobila, a n filosofii příslušného operačního systému.

Princip získání vektoru přerušení je znázorněn na obr. 117. Vnější signál je zachycen na příslušném vstupním obvodu přerušovacího systému (podle své váhy), který informaci o příchodu přerušení uchová[1]. Poku registr obsahující masku přerušení, který umožňuje, že některá přerušení jsou postoupena dále a jiná nejso předána ke zpracování, je nastaven tak, že příslušné přerušení projde až ke kodéru, váha tohoto přerušení j zakódována do podoby binárního čísla. Toto číslo se pak používá jako ukazatel umístění vektoru v tabulc vektorů přerušení při přípravě ukazatele na dotčený vektor. Poznamenejme, že kodér musí obsahovat prioritn obvody, které umožní, že např. zakóduje pouze signál s nejvyšší vahou, přestože v daný okamžik mohou bý aktivní i jiné signály.



Obr. 118 Zapojení řadiče sběrnice a procesoru pro přímý přístup do paměti

8.3.3 Blokové přenosy - DMA[2]

Ne vždy je žádoucí, aby se procesor aktivně účastnil na přenosu dat mezi dvěma propojenými jednotkami Pokud je propojení uskutečněno sběrnicí tak, jak je ukázáno na obr. 118, může docházet ke třem základním variantám přístupu jednotek na sběrnici

- procesor je řadičem DMA zastaven a tato externí jednotka převezme po dobu přenosu dat řízen sběrnice. Po ukončení přenosu je odevzdáno řízení sběrnice zpět procesoru - DMA.
- řadič zpomalí procesor za účelem přímého přístupu do paměti tak, že po dobu přenosu dat zastaví jeh hodiny – kradení cyklů
- procesor se podělí o sběrnici v reálném čase tak, že část cyklu sběrnice připadne jedné jednotce a čás druhé, aniž by se významně snížila rychlost provádění instrukcí – multiplexování.

Metoda blokového přenosu DMA je často používána pro výměnu dat s řadiči rychlých periferních zařízení, jak jsou disky nebo adaptéry do počítačových sítí. Počet řadičů používajících DMA na jedné sběrnici není omezen avšak pokud je jich více, je nutno aby existoval opět arbitrační algoritmus, který rozhodne, kterému řadiči má bý přiděleno právo řízení sběrnice.

8.3.3.1 Přímý přístup s odpojením procesoru

V případě přímého přístupu s odpojením procesoru požádá v úvodní sekvenci jednotka, která hodlá použí sběrnici pro přístup do paměti, o sběrnici, počká na potvrzení uvolnění sběrnice a pod vlastním řízením přenes

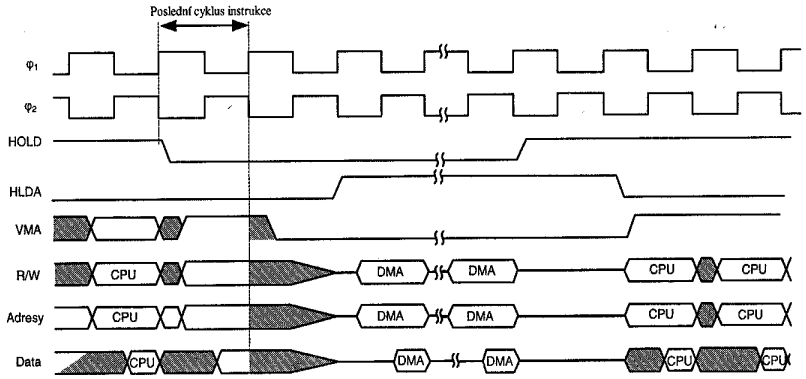
---

[1] Jedná se o tzv. přerušení hranou, přerušovací signál může být navrácen do klidové polohy i před tím, než bude zpracován. Používá se i přerušení na stav přerušovacího signálu, který se navrací do klidové hodnoty až po potvrzení přijetí přerušení příjemcem tohoto přerušení.

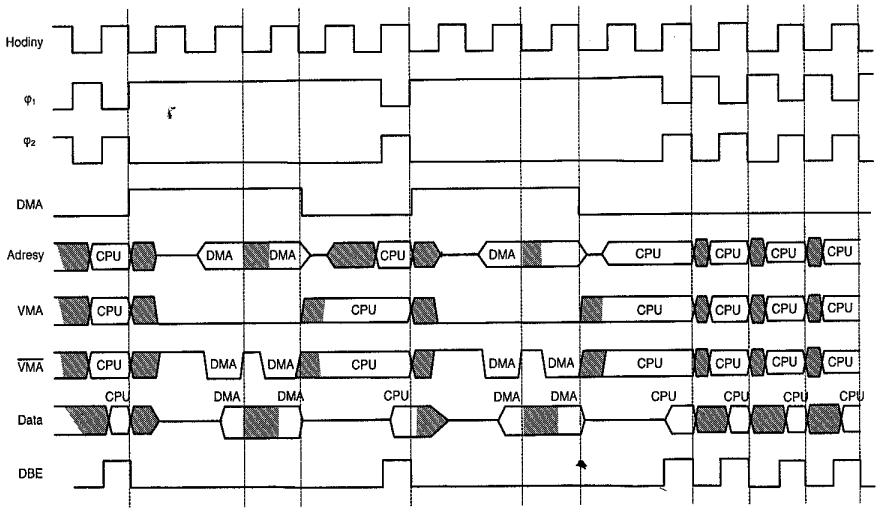
[2] Zkratka pro přímý přístup do paměti - Direct Memory Access

lata. Po ukončení přenosu předá sběrnici procesoru. Příklad časového diagramu přímého přístupu do paměti ro mikroprocesor Motorola MC6800 je na obr. 119. Přímý přístup do paměti je iniciován požadující jednotkou, terá vydá signál **HOLD**, jenž je požadavkem na procesor, aby předal sběrnici požadující jednotce. Pokud to rocesor může učinit, dokončí právě probíhající instrukci, odpojí se od sběrnice a potvrdí odpojení signálem **HLDA**.. Od tohoto okamžiku je přístup do paměti řízen jednotkou, která si přístup vyžádala, a procesor je astaven. Tato jednotka je zodpovědná za správnou adresaci, časování a povely ke čtení nebo zápisu – tedy za šechny adresové, datové a řídící signály pro paměťový blok. Po ukončení přenosu jednotka, která přístup ožadovala, uvolní signál **HOLD**, procesor potvrdí, že přijal informaci o možnosti opětného připojení na sběrnici ignálem **HLDA**, a na sběrnici se připojí.

Nevýhodou tohoto principu, přestože umožňuje nejrychlejší přenosy mezi periférií a pamětí, je zastavení rocesoru, a tím i exekuce instrukcí. Pokud však bude mezi sběrnici a procesor připojena vyrovnávací paměť, rocesor může ve vykonávání instrukcí pokračovat do té doby, než dojde k požadavku přístupu na sběrnici. Celá kce předání a převzetí sběrnice probíhá asynchronně.

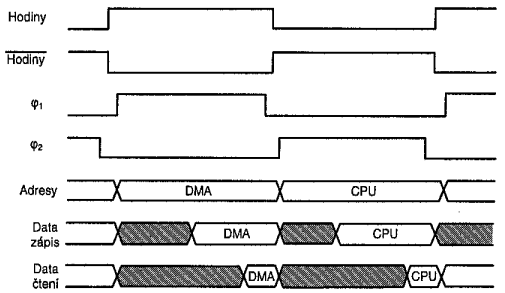


Obr. 119 Přímý přístup do paměti s odpojením procesoru – mikroprocesor MC6800



Obr. 120 Přímý přístup do paměti – metoda kradení cyklů

Předcházející případ vedl v krajním případě k zastavení procesoru a tím i k zastavení provádění programu Aby k tomu nedošlo, používá se často metoda nazývaná **"cycle stealing"** - kradení cyklů, která provádění instrukcí zcela nezastaví, pouze zpomalí. Pro vysvětlení použijme opět časový diagram práce mikroprocesoru MC6800 se sběrnicí při metodě kradení cyklů. na obr. 120.



Obr. 121 Přímý přístup do paměti - multiplexace

Aby došlo k iniciaci přímého přístupu do paměti, jednotka – řadič, požadující přístup musí nastavit signál DMA. Tento signál ovládá jednak obvody hodin procesoru (φ_1, φ_2) a jednak obvody, pomocí kterých je procesor připojen na sběrnici. Po jeho příchodu jsou hodiny podrženy v jednom krajním stavu a procesor je odpojen od sběrnice. Řadič přímého přístupu do paměti provede potřebný přenos a uvolní signál DMA. Tím se obnoví připojení procesoru ke sběrnici a hodinové pulsy pro procesor. V tomto případě je procesor skutečně zastaven a vzhledem k tomu, že většina obvodů uvnitř procesoru staví na předpokladu dynamického režimu, je čas, po který může být přerušena dodávka hodinových pulsů do procesoru, omezen[1]. Tento čas je také limitujícím faktorem doby přenosu dat mezi řadičem přímého přístupu do paměti a pamětí. Časový diagram ukazuje dva přenosy DMA při každém zastavení hodin, avšak podle parametrů procesoru je jich možno provést i více.

8.3.3.3 Multiplexovaný přístup

Tato metoda, ve srovnání s ostatními, umožňuje nejrychlejší přístup do paměti při zachování maximální rychlosti procesoru. Jejím nedostatkem je nutnost použití pamětí s krátkou dobou přístupu, což vede ke zdražení paměťového subsystému. Pro objasnění použijeme opět časový diagram mikroprocesoru Motorola MC6800 na obr. 121.

Z tohoto obrázku je zřejmé, že paměť je vybrána tak, aby její přístupový čas vyhovoval rychlosti hodin procesoru. Jedna polovina hodinového pulsu bude odpovídat práci procesoru zatímco druhá polovina bude určena pro pracovní cyklus paměti. Jak již bylo uvedeno výše, paměti s krátkou dobou přístupu jsou drahé, a proto se často volí při této metodě kompromis mezi rychlostí, výkonem systému a cenou a procesor pracuje se zpomalenými hodinami, což vede k jeho menšímu výkonu. Konkrétní časování a typ paměti pak bude záviset na požadavku na výkon subsystému DMA, požadavku na rychlost procesoru a rychlosti použitých pamětí.

8.3.4 Samostatné procesory

Tendence zvyšování nezávislosti komunikace s připojenou jednotkou na procesoru vedla ke koncepci nezávislých komunikačních procesorů. Tyto komunikační procesory pak jsou propojeny s výkonnými aritmetickými procesory systému rychlými spoji a vykonávají speciální operace podle požadavků aritmetických procesorů s dalšími připojenými jednotkami, jejichž rychlost je o několik řádů nižší. Tím se docílí, že aritmetický procesor není zatížen operacemi s připojenou jednotkou, předá pouze po rychlém spoji příkaz k provedení přenosu bloku dat, podobně jako v případě DMA, s požadavkem na informování o ukončení této operace.

[1] Např. registry mikroprocesorů jsou většinou dynamické a při delším zastavení hodin by došlo ke ztrátě jejich obsahu.

Tyto samostatné procesory jsou speciálně navrženy a obsahují tedy i speciální sadu instrukcí orientovanou na operace přenosu dat a nezbytné kroky s tím spojené. Programům běžícím na těchto procesorech obvykle říkáme **kanálové programy** a spoji mezi aritmetickým procesorem a komunikačním procesorem **kanál**. Spolupráce mezi komunikačním procesorem a aritmetickým procesorem se obvykle odehrává na úrovni kombinovaných sekvencí přerušení a přímého přístupu do paměti aritmetického procesoru s celou řadou dodatečných stavových informací.

9 Taxonomie výpočetních systémů

následováno složitým řešením
A. Bloch: "Murphy's Law

Procesory, paměti a propojovací systémy jsou základními prvky výstavby počítačových systémů. Uživatelů běžně známé, mikroprocesorem vybavené personální počítače se standardní jednoduchou organizací, přesto jsou replikovány v miliónech exemplářů, jsou pouze nepatrným úsekem pestré mozaiky počítačových organizac které během vývoje tohoto oboru spatřily světlo světa. Nejrůznější sestavy, využívající kombinací procesor pamětí a propojovacích systémů spolu s architektonickými prvky výstavby stroje, směřovaly k dosažení c nejvyššího výkonu stroje. Rozmanitost sestav se však vždy vyznačovala některými společnými rysy, které ved k zavedení základních taxonomií v architektuře a organizaci digitálních počítacích strojů.

9.1 Flynnova taxonomie

Pokud budeme důslední a budeme dbát na detaily, pak nutně dojdeme k závěru, že úplná a vyhovují charakterizace různých typů počítačových systémů neexistuje. Ze všech pokusů, které byly učiněny, nejznámější a zároveň nejjednodušší taxonomie definovaná Flynnem v roce 1966 [A3] založená na analýze **tok informace v počítači**. Z tohoto hlediska sledujeme dva typy informačních toků pohybujících se v počítači **instrukce a data**. Koncepčně mohou tyto dva druhy informací být rozděleny do dvou nezávislých toků pod toho, zda vstupují do počítače stejnou nebo různou cestou.

Flynnova taxonomie používá rozdělení počítačů podle toho, zda obsahují pouze jeden nebo více tok informací výše uvedeného typu a rozlišuje čtyři kombinace:

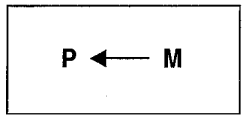
- **SISD** - **S**ingle **I**nstruction **S**ingle **D**ata (jeden tok instrukcí a jeden tok dat)
- **SIMD** - **S**ingle **I**nstruction **M**ultiple **D**ata (jeden tok instrukcí a vícenásobný tok dat)
- **MISD** - **M**ultiple **I**nstruction **S**ingle **D**ata (vícenásobný tok instrukcí a jeden tok dat)
- **MIMD** - **M**ultiple **I**nstruction **M**ultiple **D**ata (vícenásobný tok instrukcí a vícenásobný tok dat)

Přesto, že pro mnohé moderní stroje je přesné zařazení podle Flynnovy taxonomie obtížné, pro svo jednoduchost a přímost je velmi často používána.

9.2 Procesory typu SISD

Běžné jednoprocesorové stroje von Neumannovského typu jsou klasifikovány jako systémy typu **SISI** PMS diagram na obr. 122 znázorňuje jeden procesor, který z paměti vybírá instrukce i data a provádí všechn operace. Každá aritmetická instrukce provádí operaci s daty, která jsou přebírána z jednoduchého proud datových elementů. Do této kategorie lze začlenit jak historické superpočítače, jako např. Control Dat Corporation 6600 a 7600, tak i některé jednodušší současné mikroprocesory.

Vektorové procesory, jako např. stroj Cray-1 a jeho následovníci jsou často klasifikovány jako stroje typu SIMD, ačkoliv by spíše patřily do kategorie SISD. Vektorové procesory, jak bylo popsáno v 6.5.2 dosahují totiž svého vysokého výkonu tím, že využívají nezávislosti fází jednotlivých operací a pro každou tuto operaci využívají speciální část hardwaru.



Obr. 122 PMS diagram stroje typu SISD

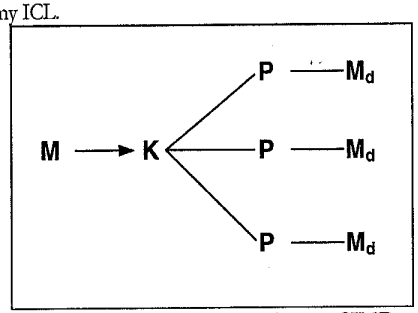
Příklad

Očekávejme, že vektorový stroj má provést součet dvou čísel s různým exponentem jako $3{,}4.2^3$ a $1{,}6.2^2$. K tomu aby mohla operace proběhnout, musí tato čísla mít stejný exponent. To znamená, že procesor musí zajistit posun mantisy a snížení hodnoty exponentu u jednoho ze sčítanců tak, aby docílil stejné hodnoty exponentů u obou sčítanců. V našem příkladě získáme posunem mantisy vpravo z hodnoty $3{,}4.2^3$ hodnotu $6{,}8.2^2$, takže může být přičtena k hodnotě $1{,}6.2^2$ a výsledkem sčítání je hodnota $8{,}4.2^2$. Vektorový procesor je speciálně navržen tak, aby předával data procesoru s co nejvyšší rychlostí, takže zatímco jedna část procesoru sčítá mantisy dvojice (a_i,b_i), druhá část procesoru nastavuje hodnotu exponentů v $(a_i + 1, b_i + 1)$.

ápání toku dat. Statický vzorek stavu procesoru při zpracování vektoru v předcházejícím příkladu ukáže kolik částí dat, která jsou zpracovávána v témž čase, a tedy z tohoto hlediska můžeme říci, že jedna instrukce ıučet vektorů) inicializovala několik operací s daty (nastavení exponentů, součet mantis apod.). Tedy stroj by ohl být klasifikován jako SIMD. Pokud však zvolíme dynamický pohled, pak je zcela jednoznačný jeden tok t a elementy tohoto toku jsou sekvenčně zpracovávány proudovým způsobem na lince proudového racování, která tento součet implementuje.

.3 Procesory typu SIMD

Procesorovou jednotku stroje typu SIMD chápeme často jako řadič pro zpracování jedné instrukce a v PMS znamu je označována písmenem **K** (viz. obr. 123). Řadič řídí práci několika jednotek pro zpracování dat, které nazývají **procesorové elementy**. Je zajímavé, že tento způsob návrhu byl použit již v raném stadiu rozvoje početní techniky pro první stroj pracující ve třídě SIMD - **Illiac-IV**. Jiným příkladem tohoto procesoru může t tzv. **D**istributed **A**rray **P**rocessor (**DAP**) od britské firmy ICL.



Obr. 123 PMS diagram stroje typu SIMD

Řídící jednotka zodpovídá za vyzvednutí strukce z paměti a její interpretaci. Pokud instrukce, erou přečte řídící jednotka z paměti, je instrukce tmetická nebo vyžaduje další zpracování, je předána em procesorovým elementům, které následovně ovedou jednu a tutéž operaci. Např. bude-li tato strukce `add r3,r0`, pak každý procesorový ement sečte obsah svého vlastního interního registru 3 a `r0`. Aby byla umožněna větší flexibilita při ıplementaci jednotlivých algoritmů, každá ocesorová jednotka může být řadičem aktivována. Procesorový element je buď nečinný, dy tento element nevykonává na základě instrukce žádnou činnost, nebo je aktivní a provádí stejnou operaci ko ostatní aktivní procesorové elementy.

Každý procesorový element má svoji vlastní paměť pro uložení dat. Instrukce, jejímž operandem je ference na paměťové místo, např. `load r0,100` způsobí, že každá procesorová jednotka zavede do svého ıitřního registru `r0` obsah paměťového místa na adrese `100`, kde touto adresou se míní adresa ve vlastní kální paměti.

Významnou výhodou této organizace stroje je úspora potřebné logiky při implementaci. Obvykle při ıplementaci stroje je třeba 20 - 50 % logických obvodů v typickém procesoru pro kontrolu a řízení jeho práce, dy k vyzvednutí instrukcí, jejich dekódování atd. Ostatní logika je určena pro lokální paměti - registry a rovnávací paměti, a implementaci algoritmů zpracování dat - sčítačky, násobičky apod. Ve stroji typu SIMD je ıužita pouze jedna řídící jednotka, která vyzvedne a zpracuje instrukci, takže ostatní logika může být plně ıužita aritmetickými obvody a registry. Jak samo uspořádání stroje napovídá, vektorové zpracování bude hodně prováděno pomocí distribuce jednotlivých elementů vektoru do lokálních datových pamětí.

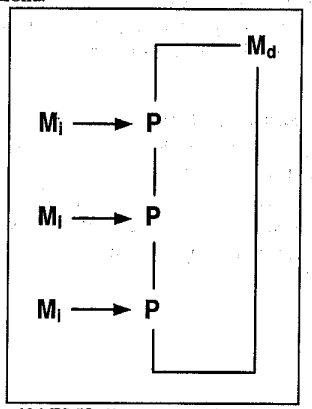
Příklad

Nechť existují dva vektory **A** a **B** a stroj, který má 1024 procesorových elementů. Prvek A_i je uložen v místě **0** paměti **i** a prvek B_i na adrese **1** paměti **i**. Abychom sečetli vektory **A** a **B**, stroj by musel instruovat každý procesorový element, aby zavedl obsah adresy 0 do jednoho registru a obsah adresy 1 do druhého registru, sečetl tyto dva registry a uložil výsledek. Tato operace bude trvat stejně dlouho pro každý vektor, jehož počet členů bude menší, než je počet procesorových elementů ve stroji typu SIMD. Tedy do té doby nebude záviset na délce vektoru. Tatáž vektorová operace na procesoru typu SISD, který používá proudové zpracování instrukcí, však bude lineární funkcí délky zpracovávaného vektoru.

.4 Procesory MISD

V této kategorii se nalézá pouze několik strojů, o kterých se nedá říci, že by byly výrazně komerčně úspěšné, ebo měly v minulosti nějaký významný efekt. PMS diagram stroje Typu MISD je na obr. 124, kde tok dat paměti instrukcí, označené M_i. Jedním ze systémů, který by spadal do popisu stroje typu MISD, je tzv **systolické pole**, což je síť malých procesorových elementů spojených v pravidelné uspořádané síti. Všechny elementy jsou řízeny centrálními hodinami. V každém cyklu každý element čte úsek dat předaných od jednoho svého souseda, provede jednoduchou operaci, např. přičte přicházející data k datům uloženým, a připrav výslednou hodnotu k zapsání na vstup svého souseda v následujícím kroku.

Zdálo by se, že případ proudového zpracování instrukcí bude patřit do této kategorie, protože každý krok linky proudového zpracování odpovídá různým operacím prováděným s daty tak, jak procházejí různými stupni zpracování. Existovaly dokonce proudové procesory s programovatelnými stupni, to znamená, že funkce, kterou každý stupeň vykonával, se mohla měnit. Jednotlivé stupně proudového zpracování však nevyzvedávají své operace z lokální řídící paměti, tudíž by bylo velmi obtížné je klasifikovat jako procesorový stupeň.

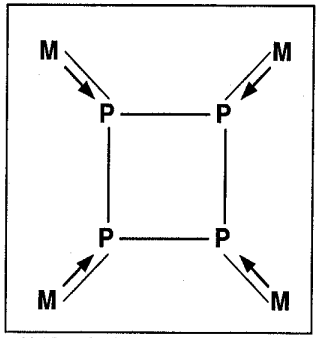


Obr. 124 PMS diagram stroje typu MISD

9.5 Procesory MIMD

Stroje typu MIMD jsou nejrozmanitější kategorií ze všech čtyř klasifikací Flynnovy taxonomie. Zahrnují stroje s procesory a paměťovými jednotkami, které jsou speciálně navrženy jako komponenty paralelní architektury, rozměrné paralelní stroje vybudované z mikroprocesorů dostupných a běžně vyráběných typů, malé multiprocesorové stroje vytvořené spojením čtyř vektorových procesorů a celou řadu dalších návrhů. Příklad PMS diagramu stroje typu MIMD je na obr. 125, kde procesory nezávisle vyzvedávají instrukce a zpracovávají data, přičemž mohou komunikovat přímo nebo prostřednictvím sdílené paměti (není uvedeno v diagramu). Se zvyšující kvalitou komunikace v síti procesorů a vývojem softwarové podpory, který umožňuje programu běžícímu na jednom stroji komunikovat s programy na ostatních strojích, se i sítě lokálních pracovních stanic stávají systémy typu MIMD.

Na druhou stranu počítačové systémy s dvěma či více nezávislými procesory byly komerčně dostupné již delší dobu. Např. firma Burroughs Corporation prodala verze svého systému B6700 ve variantě s dvěma procesory již v roce 1970. Avšak tyto dva procesory byly velmi zřídka, jestli vůbec kdy, použity ke zpracování téže úlohy. Komerčně dostupné paralelní procesory, které využívaly paralelizace úloh, se staly široce dostupné ke konci osmdesátých let. Začátkem let devadesátých s rozvojem vektorových procesorů a dalších architektur je zřejmé, že budoucnost zvýšení výkonů leží v organizacích využívajících paralelní zpracování.



Obr. 125 PMS diagram stroje typu MIMD

9.6 Další používané taxonomie

Jak jsme se již seznámili, jedna z největších slabin klasifikace kategorií procesorů podle Flynna je zařazení vektorových procesorů. Rovněž kategorie MIMD tak, jak je široce pojata, zahrnuje pole všech procesorů nezávisle na tom, jak jsou vzájemně propojeny a jak jsou připojeny jejich paměti. Protože právě připojení paměti je jeden z parametrů, který může mít výrazný dopad na výkon stroje, bylo by žádoucí, aby taxonomie odrážela tyto rozdíly. Objevují se snahy nabídnout velmi podobnou taxonomii, která rozdělila čtyři hlavní kategorie do dalších subkategorií. Např. Hwang [A4] rozdělil kategorii MIMD do subkategorií podle toho, zda paměť byla sdílená nebo distribuovaná a zda systém byl rekonfigurovatelný. Bohužel, tento pokus o další rozdělení taxonomie vzájemně smísil organizaci paměti s organizací komunikace, a i když popsané odlišení je velmi významné, není uspokojující jako základ pro přesnější taxonomii. Jednodušší rozdělení použité Bellem [A5] pro kategorii MIMD se spokojilo pouze s rozdělením na systémy využívající sdílenou paměť a systémy bez sdílené paměti.

říklad velmi populárního typu **SPMD**, což je zkratka pro Single Program/Multiple Data, i když při rigorózním hápání přístupu k taxonomii počítačů vybočuje tato charakteristika z původního pojetí, neboť se opírá spíše o ɛprezentaci stylu zpracování výpočtu než o architekturu a organizaci stroje. Obecně totiž zapadá do kategorie /IMD, protože může mít několik nezávislých procesorů, každý se svým vlastním souborem dat a pamětí rogramu. Avšak každým procesorem je vykonáván tentýž program a tyto procesory jsou periodicky ynchronizovány. Takové, zjednodušující chápání přístupu k systému MIMD, umožňuje i daleko větší flexibilitu ež systém SIMD, protože různé procesory mohou v kterémkoliv čase být v různých částech zpracovávaného rogramu.

Jedna z nejsložitějších až nejasných taxonomií byla popsána Pockneyem a Jeshopem [A6]. Motivací autorů ylo pravděpodobně popsat vektorový procesor s proudovým zpracováním dat jako odlišnou architekturu, a tak odlišit od mnoha možností konfigurace multiprocesorového stroje. Používané záznamy organizace strojů ɪřipomínají chemické vzorce organických sloučenin a bližší popis může čtenář najít v již uvedené literatuře. Na lruhou stranu je pravdou, že navržená klasifikace poskytuje unikátní taxonomii pro všechny systémy, které až losud byly navrženy a vyrobeny.

se jen obtížně odliší od magie
A. Bloch: "Murphy's Law"

10 Počítačové systémy s vysokým výkonem

Stále se zvyšující požadavky na výkon počítačových systémů vedou ke stále vyššímu a kvalitnějšímu využívání technologií a technologických vylepšení. Avšak i tato technologická vylepšení mají svá fyzikální omezení. To je důvodem, proč se vedle všech těchto technologických přístupů objevují i techniky založené na paralelismu, tedy organizaci stroje, využívající opakování základních komponentů v systému. Při tomto přístupu můžeme např. v organizaci systému navrhnout použití čtyř sčítacích a násobících jednotek místo jedné uvnitř centrální procesorové jednotky, nebo připojit jednu či více pamětí k procesorové jednotce za účelem zvýšení šířky pásma. Přitom paralelní zpracování není žádnou novinkou a jeho první představa se datuje již od dob Charlese Babbage na začátku 19. století. V zásadě můžeme rozlišit čtyři výrazné úrovně paralelismu

- **na úrovni úlohy**, což je nejvyšší stupeň paralelismu a je spíše předmětem zájmu administrátorů systému než individuálního uživatele. Za nejdůležitější prvek z tohoto pohledu lze pokládat to, že výpočetní systém může zpracovávat množství úloh v daném časovém intervalu. Této charakteristiky může být jednoduše docíleno například nakupením většího množství počítačových systémů, na kterých může v daném čase běžet více úloh, ačkoliv jednotlivá úloha bude zpracovávána stále stejně rychle.
- **na úrovni programů** je aplikován tak, že jednotlivý program je rozdělen do menších složek, tedy např. součin matic může být vypočítán tak, že program, který jej realizuje, bude rozdělen na čtyři části a každá z těchto částí bude zpracována zvláštním procesorem, kterému budou k dispozici odpovídající data. V tomto případě bude celý součin vypočítán zhruba čtyřikrát rychleji než na jednom procesoru.
- **na instrukční úrovni** je pro uživatele neviditelný. Leží pod úrovní architektury v doméně organizace počítače. Nejčastěji se jedná o proudové zpracování, které je nejběžnějším způsobem implementace tohoto typu paralelismu.
- **na úrovni aritmetiky** je nejnižší úrovní a je v ohnisku pozornosti návrhářů aritmeticko-logických jednotek uvnitř centrálního procesoru. Příkladem může být součet 64 bitových čísel, který může být spočítán buď najednou paralelní sčítačkou, nebo z nějakých důvodů může návrhář systému rozdělit tuto operaci do tzv. **bitových řezů** např. se šířkou čtyři bity, a vypočítat celou sumu v šestnácti cyklech.

Paralelismus na úrovni úloh může být rovněž realizován uvnitř jednoho počítače tak, že úlohu nebo několik úloh budeme považovat za soubor nezávislých úkolů. Např. několik úloh může být ve stejnou dobu v paměti, avšak pouze jedna se bude v daném okamžiku provádět. Ve chvíli, kdy tato úloha požaduje např. vstupně-výstupní operaci, je tato operace je inicializována a úloha, která ji požadovala, je suspendována, zatímco jiná úloha zabere její místo a začne se provádět. Za nějakou dobu, když je vstupně-výstupní operace ukončena a data jsou k dispozici, řízení přejde zpět k původní úloze a její provádění může pokračovat. V tomto konkrétním případě centrální procesorová jednotka a vstupně-výstupní systém pracují paralelně.

Paralelismus na úrovni programů může být obecně vnímán dvěma způsoby - jako **nezávislost sekcí zpracovávaného programu** nebo jako **individuální iterace uvnitř cyklu**, ve kterém existuje nezávislost mezi jednotlivými iteracemi. Tento typ paralelismu může být široce využit ve víceprocesorových systémech nebo v systémech, které obsahují větší počet funkčních jednotek.

Příklad

Příkladem individuálních iterací uvnitř cyklu může být realizace segmentu programu pro výpočet n součtů

```
      DO 10 I = 1, N
      A(I) = B(I) + C(I)
   10 CONTINUE
```

Protože jednotlivé součty jsou nezávislé, kalkulace `B(I)+C(I)` nezávisí na `B(J)+C(J)` pro kterékoliv `J≠I` a tyto operace mohou být prováděny v libovolném pořadí. Pokud stroj bude obsahovat n procesorů, je možno celý tento cyklus provést v jednom časovém okamžiku. Je zřejmé, že i složitější segmenty programu mohou být ošetřeny podobným způsobem, závisí to pouze na "rafinovanosti" návrhu organizace a architektury.

Hlubší stupeň paralelismu pracující na úrovni [illegible] ...racování vzájemně překrývat, nebo aby daná instrukce byla rozložena na jednotlivé operace, které jsou ...isledovně provedeny paralelně, nebo jejich zpracování se opět může překrývat. V prvním případě můžeme najít ...např. takové operace, které současně kopírují hodnotu z paměti do vnitřního registru procesoru spolu s ...ováděním některé jiné aritmetické instrukce. Druhý případ může reprezentovat proudové zpracování instrukcí, ...le linka proudového zpracování se stává hlavním místem pro provádění aritmetických instrukcí. V obecném ...případě se programátoři nemusí zabývat touto úrovní paralelismu, protože kompilátory již jsou navrženy tak, aby ... přizpůsobily tomuto druhu zpracování instrukcí a reorganizovaly překládané programy s maximálním ...razem na využití realizovaného typu paralelismu. Nicméně je vždy nutno mít na paměti, že kvalita kompilátorů ... systém od systému výrazně mění a struktura překládaného kódu nemusí vždy vyhovovat způsobům, které ...ou používány kompilátorem pro optimalizaci.

Příklad

Počítače Cray jsou nejefektivnější, když délka zpracovávaného vektoru je 64 nebo menší, protože pak je možno celý vektor uložit do vektorových registrů a zpracovat jej najednou. V případě delších vektorů překladač uspořádá výpočet tak, aby procesor pracoval se segmenty těchto dlouhých vektorů, a tak se zvýšil výkon. Interní struktura a způsob, jakým překladač bude předložený program reorganizovat a zpracovávat, bude vždy centrem pozornosti při měření výkonu počítače na konkrétním programu.

Koncepce paralelismu rozeznává rovněž různé stupně paralelního zpracování úloh, které jsou závislé na tzv. **...ranularitě** úlohy. V systému pracujícím s velkou granularitou budou části, které probíhají paralelně, relativně ...elké, např. na úrovni celého programu. Systémy pracující s malou granularitou rozdělí programy do velmi ...alých částí, v mnoha případech čítajících jenom několik instrukcí. Např. procesor, který bude kalkulovat n ...očtů paralelně v n sčítacích jednotkách, bude příkladem velmi jemné granularity.

Při návrhu nového stroje stojíme zpravidla před rozhodnutím, zda k tomu, abychom dosáhli požadovaný ...ýkon, použít relativně malé množství výkonných procesorů nebo velké množství jednoduchých procesorů. ...říkladem první varianty použití relativně malého počtu výkonných procesorů jsou systémy firmy Cray Research, ...nc., které obsahují od 2 do 16 velmi výkonných vektorových procesorů. Ve prospěch tohoto řešení mluví ...ednodušší způsob jejich propojení, snadnější implementace navržené organizace paměti a současně s tím ...snadnější programovatelnosti. Na druhé straně, tyto procesory jsou velmi náročné na výstavbu, spotřebu a ...případně i potřebné chlazení[1]. Příklad druhého řešení, použití většího množství jednoduchých procesorů, může ...loužit stroj označovaný CM-1 od firmy Thinking Machines Corporation, který obsahoval až 65 536 ...ednobitových procesorů. Takové stroje, které používají velkého počtu jednoduchých procesorů se obvykle ...azývají **masivně paralelními procesory**.

Při výstavbě paralelních počítačů je v současnosti snaha využít běžně dostupných mikroprocesorů, které ...možňují na relativně levné platformě navrhnout výkonný systém s velmi komplexním propojením. Se ...zrůstajícím výkonem mikroprocesorů lze dosáhnout s polem několika set mikroprocesorů teoretického výkonu ...ejrychlejšího stroje, který nabízí firma Cray Research, Inc. Tomuto trendu se podřídila např. i firma Intel a ... roce 1996 začala s výstavbou stroje ASCI pro americké Ministerstvo energetiky, který se skládá z více než 9000 ...procesorů Intel Pentium Pro 200 MHz doplněných 580 GB operační paměti.

10.1 Přehled organizací s vysokým výkonem

Mezi nejvíce používané organizace strojů s vysokým výkonem patří

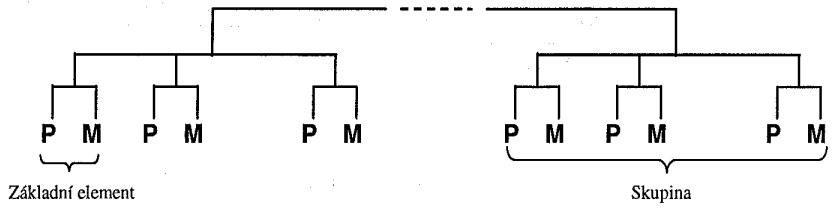
- multiprocesory MIMD se sdílenou pamětí,
- multiprocesorové vektorové stroje,
- systémy MIMD s distribuovanými pamětmi,
- stroje SIMD.

[1] Počítače firmy Cray byly chlazeny kapalným heliem.

[illegible] ...distribuované paměti jsou zvláštní kategorií, která vlastně reprezentuje výkonnou počítačovou síť; avšak používaný přístup k řešení úloh na takto konstruovaných systémech se bude lišit od typického provozu počítačové sítě, podobně jako práce paralelních systémů.

10.1.1 Multiprocesory MIMD se sdílenou pamětí

Historie multiprocesorů se sdílenou pamětí začíná v letech 1970, kdy ve výzkumném projektu na Carnegie Mellon University byl sestaven první MIMD stroj pojmenovaný **c.mmp**[1]. Tento multiprocesorový stroj obsahoval centrální křížový přepínač, který propojoval 16 minipočítačů PDP-11 a 16 jednotek paměti. Následovník tohoto stroje užíval rovněž procesory PDP-11, avšak tentokrát byly propojeny prostřednictvím stromové sítě tak, jak je ukázáno na obr. 126. Základní stavební blok tvořila skupina procesorů, která obsahovala čtyři procesory s jejich vlastní lokální pamětí. Globální paměťový prostor byl rozdělen na stejné paměťové bloky, a tak když procesor generoval požadavek na přístup k adrese i, logika na jeho sběrnici musela zjistit, zda-li adresa i je v intervalu adres jeho lokální paměti. Pokud tomu tak nebylo, byl požadavek přenesen na řadič skupiny, který opět zjišťoval, zda adresa patří paměti uvnitř skupiny. Jestliže ani tady nedošlo k uspokojení žádosti, byl požadavek přesměrován ve stromové struktuře na jinou úroveň skupinových řadičů. Tímto způsobem bylo propojeno 50 procesorů v třírúrovňové sběrnicové hierarchii. Původně se tento návrh označoval jako neuniformní přístup k paměti[2] a v závislosti na tom, zda-li požadovaný úsek dat byl v lokální paměti procesoru, uvnitř téže skupiny nebo v jiné skupině, se doba přístupu k těmto datům zvětšovala (od 3 μs do 27 μs)[3].



Obr. 126 PMS diagram systému c.mmp

Jeden z prvních komerčně dostupných systémů MIMD se sdílenou pamětí byl počítač **BBN Butterfly**. Jak již jméno samo naznačuje, sestával se z motýlového přepínače, který propojoval 256 procesorů a paměti. Byly použity mikroprocesory Motorola 68000 a organizace BBN Butterfly přidala ještě zvláštní propojení vždy mezi jedním procesorem a jednou pamětí, takže párováním procesorů s paměťovým modulem získal každý procesor přednostní jednotku paměti. Tuto jednotku paměti mohl procesor používat přímo, bez toho, aby informace procházela přepínačem. Výsledkem byla opět architektura NUMA s poměrem přístupových časů zhruba 15:1 podle toho, zda-li procesor používal motýlový přepínač nebo přímé spojení.

Jiným příkladem komerčně používaného systému v této kategorii je počítač **KSR-1** od firmy Kendall Square Research (viz obr. 127). Procesory jsou v tomto počítači spojeny do kruhů s 8 až 32 procesory v jednom kruhu. Větší systémy obsahují druhou úroveň kruhu, která může propojovat až 34 kruhů první úrovně. Maximální rozměr tohoto stroje je tedy 1088 procesorů. Každý kruh je jednosměrný, to znamená, že informace se může pohybovat pouze jedním směrem a rychlost přenosu v tomto kruhu je 1 Gbyte/s.

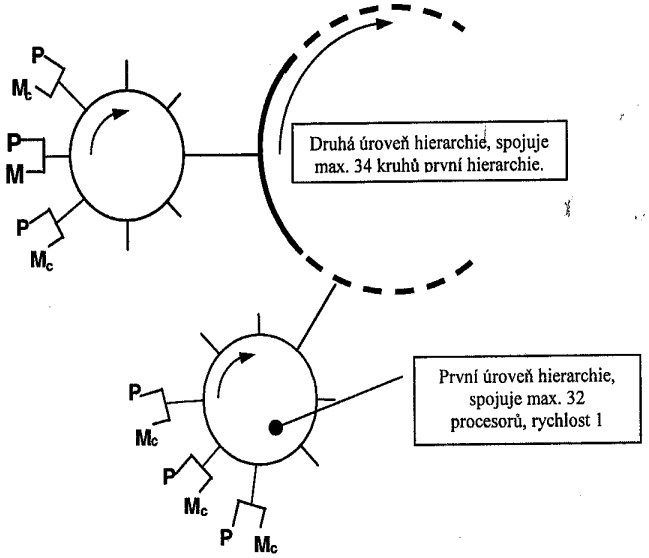
Každý procesor má svoji vlastní vyrovnávací paměť (cache) o velikosti 32 MB, ale nemá žádnou primární paměť. Tato nezvyklá organizace používá vyrovnávací paměť a v ní uložený adresář ke zpřístupnění všech potřebných informací. Když procesor požaduje přístup k datům na adrese i, řádek vyrovnávací paměti, který obsahuje tuto položku, se pohybuje podél kruhu do té doby, než dosáhne procesoru, který tuto informaci požadoval. Speciální hardware zajišťuje nezbytnou koherenci, aby společné části dat byly neustále udržovány ve správném stavu. Pro tento typ systému se někdy používá název systém se **sdílenou virtuální pamětí**. Procesory

[1] Název pochází z PMS diagramu pro počítač s několika miniprocesory.

[2] NUMA - Non Uniform Memory Access.

[3] Pro porovnání: počítač PDP 11 v té době mohl obdržet informaci ze své paměti během zhruba 2 μs.

použite ve stroji KSR-1 jsou 64bitové superskalární procesory s hodinovým cyklem 20 MHz. Podle interní pomocí Linpack Bench Mark jeden procesor KSR-1 dosahuje výkonu 31 MFLOPS a teoretické špičky 40 MFLOPS při výpočtu gausovské eliminace matice 100x100. Systém obsahující 32 procesorů pak dosahuje ustáleného výkonu 513 MFLOPS.



Obr. 127 Organizace systému KSR-1

10.1.2 Multiprocesorové vektorové stroje

Většina výrobců vektorových superpočítačů vyrábí multiprocesorové systémy založené na svých vektorových procesorech. Protože jednotlivý procesor tohoto multiprocesorového systému je velmi drahý a je velmi přesně nastaven s ohledem na šířku pásma paměti a dalších parametrů architektury, multiprocesorová konfigurace obsahuje pouze několik procesorů. Tak je tomu i v případě nejznámějšího výrobce superpočítačů firmy Cray. Jeden z jejich největších multiprocesorových vektorových strojů, počítač Cray C90, obsahuje pouze 16 procesorů.

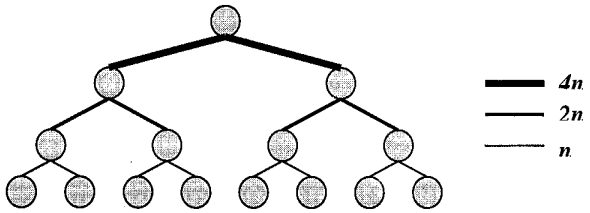
Osmiprocesorový systém Cray Y-MP je systém typu MIMD se sdílenou pamětí podobný BBN Buterfly, s procesory propojenými se sadou pamětí prostřednictvím vícestupňové spínací sítě, kterou tvoří tříúrovňový křížový přepínač. Hlavní rozdíl mezi spínací sítí u počítače BBN Buterfly a Cray Y-MP je, že počítač Cray Y-MP obsahuje daleko více modulů pamětí než procesorů, protože procesory byly původně navrženy pro přístup do paměti s prokládanými cykly. To dává dostatečnou kapacitu paměťových modulů - 32 modulů na 1 procesor a dostatek flexibility pro přepínač tak, aby každý procesor měl možnost být připojen k několika paměťovým modulům najednou. Tak může snadno přenášet vektory z paměti a do paměti, resp. z a do vektorových registrů s rychlostí jedné položky za jeden cyklus hodin, tedy i zajistit plynulé zásobování linky proudového zpracování. Rovněž přiřazení adres paměťovým modulům se liší od BBN-Buterfly, protože je použita paměť s prokládanými cykly a tak po sobě následující adresy jsou uloženy v různých modulech.

Výjimkou z výše uvedeného pravidla, že tyto typy multiprocesorových strojů obsahují pouze několik procesorů, je vektorový multiprocesor VPP500, který vyrábí firma Fujitsu. Tento počítač obsahuje 222 procesorových jednotek vzájemně propojených prostřednictvím jednostupňového křížového přepínače. Každý procesor v systému má svoji lokální paměť s prokládanými cykly, skalární procesorovou jednotku a vektorovou procesorovou jednotku. Jednotný adresový prostor je pak složen z individuálních lokálních pamětí. Každý procesor v systému VPP 500 má špičkový výkon přibližně 1,5 GFLOPS, takže celkový výkon systému s 222 procesory může dosáhnout teoretické hodnoty přes 300 GFLOPS.

10.1.3 Systémy MIMD s distribuovanými pamětmi

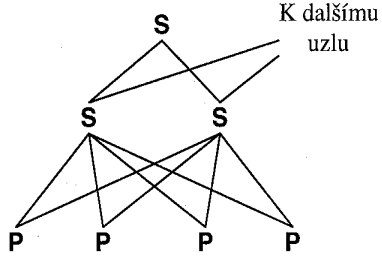
Snad nejznámější, ale také nejstarší paralelní procesor s distribuovanou pamětí byl postaven v rámc výzkumného projektu na univerzitě v Caltechu a nazýval se **"Cosmic Cube"** [A6]. Byl to jeden z prvníc systémů který zacházel s propojovací sítí jako s médiem pro výměnu zpráv.

Každý procesor v počítači Cosmics Cube byl jednoduchý počítač vybavený mikroprocesorem Intel 808(který byl doplněn koprocesorem Intel 8087 pro instrukce v pohyblivé řádové čárce a 128 kilobyty pamět Celkem 64 těchto jednoduchých počítačů bylo propojeno do šestirozměrné hyperkrychle. Rychlost komunikac v propojovací síti byla velmi pomalá, na úrovni 2 Mbit/s, a používala velmi jednoduchý protokol. Později firm Intel vybudovala vlastní komerční verzi počítače Cosmic Cube, kterou nazvala iPSC-1 a kde použila procesor 80286, 512 KB paměti na jeden uzel a spoje s rychlostí 10 Mbit/s. Konfigurace těchto komerčních verz obsahovaly od 16 do 128 procesorů, tedy vytvářely čtyřrozměrnou až sedmirozměrnou hyperkrychli. Ostatn komerční hyperkrychle z tohoto období zahrnují počítače NCUBE-1 a FPS-T.



Obr. 128 Profily komunikace ve struktuře "fat-tree"

Následovník tohoto počítače, iPSC-2 byl také založen na konfiguraci hyperkrychle, avšak místo jednoduch sítě, která pro směrování zpráv používala strategii přijetí-analýza-odeslání, byl použit systém směrování paket známý pod názvem **worm-hole**. Speciální přepínače - routery, pracující s tímto typem směrování používal modifikovanou metodu přepínání cest k tomu, aby sestavily komunikační cestu mezi dvěma procesory v systém podle předem daných pevných pravidel. Např. ve dvoudimenzionální síti toto pravidlo může být realizováno tak že zpráva se bude nejdříve pohybovat vertikálním směrem, až dosáhne řádky odpovídající umístění procesoru který je cílem této zprávy. Pak budou použity horizontální spoje k dosažení příslušného procesoru. Tímt jednoduchým pravidlem se zvýší efektivnost směrování, protože odpadne složitý proces rozhodování každéh procesoru ležícího na cestě zprávy o směru dalšího pohybu této zprávy. Avšak efektivnější způsob práce se n druhé odrazí ve ztrátě flexibility systému, protože směrování worm-hole eliminuje možnost použití alternativníc cest, které může síť poskytnout, např. kongesce na jednom spoji nemůže být odstraněna použitím alternativníc spojů.

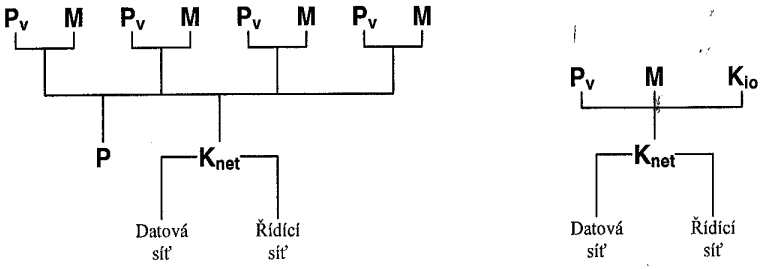


Obr. 129 PMS diagram propojovací sítě stroje CM-5

Po komerčním typu iPSC-2 byl realizován počítač iPSC860, který měl podobnou strukturu, avšak jako uzlové procesory používal mikroprocesory i860 s architekturou RISC, místo klasických mikroprocesorů 80286. V této sérii firma Intel postavila i výzkumný stroj známý jako **Touchstone Delta**, ze kterého vyšel pozdější komerční systém **Paragon XP/S**. Oba tyto stroje používají dvourozměrnou síť místo hyperkrychle a pro směrování zpráv je použito speciálně navržených integrovaných obvodů, čímž se výrazně zlepšuje komunikační šířka pásma. Každý procesorový uzel v počítači Paragon obsahoval dva mikroprocesory typu i860, z nichž jeden byl použit pro výpočty a druhý pro zpracování zpráv. Ten byl zcela dedikován zpracování zpráv a veškeré režie s tím spojené, takže hlavní proceso nebyl přerušován požadavky na manipulaci s přicházejícími zprávami.

V organizaci počítačů MIMD s distribuovanou pamětí stojí za povšimnutí stroj **CM-5** od firmy Thinkin Machines, který spojuje atributy jak SIMD, tak MIMD architektury s distribuovanou pamětí. Základní stro používá stromovou topologii, ve které každý uzel obsahuje mikroprocesor typu SPARC s možností instalac dodatečného vektorového procesoru a až 32 MB lokální paměti. Spojovací síť je založena na myšlence tzv

tlustého stromu" (fat-tree), která spočívá v tom, že stromová topologie obsahuje komunikační kanály s různým přenosovým výkonem. Čím blíže jsou tyto kanály kořenu stromu, tím vyšší mají přenosový výkon, neboť se očekává, že budou muset zpracovávat daleko větší komunikační provoz (viz obr. 128). Každý komunikační spoj e stroji CM-5 pracuje se šířkou pásma 20 Mbit/s. Z každého uzlu na větvi směřují dva spoje směrem nahoru. Tyto spoje jsou připojeny k různým přepínačům jednak pro docílení větší šířky pásma a jednak pro realizaci lternativních cest pro předcházení kongescí v síti. První úroveň vnitřních přepínačů má dva spoje směrem zhůru, avšak vyšší úroveň těchto spínačů obsahuje tyto spoje již čtyři. Tak je implementována myšlenka tlustého stromu" a docíleno vyššího přenosového výkonu směrem ke kořenům.



a) Datový uzel stroje CM-5

b) Řídící uzel stroje CM-5

Obr. 130 PMS diagram uzlů stroje CM-5

Počítač CM-5 má vlastní řídicí síť, která obsahuje sadu řídicích procesorů propojených vlastním komunikačním systémem se stromovou topologií. Řídicí procesory a jejich komunikační síť tvoří zcela separátní subsystém. Tyto řídicí procesory jsou také mikroprocesory typu SPARC, avšak protože pracují s velmi limitovaným objemem dat, obsahují daleko méně paměti a nemají vektorové koprocesory. Takto vybavený řídicí subsystém umožňuje celému systému pracovat jako architektura SIMD nebo SPMD tak, že synchronizuje množiny datových procesorů při jejich práci řízené stejným programem.

10.1.4 Stroje SIMD

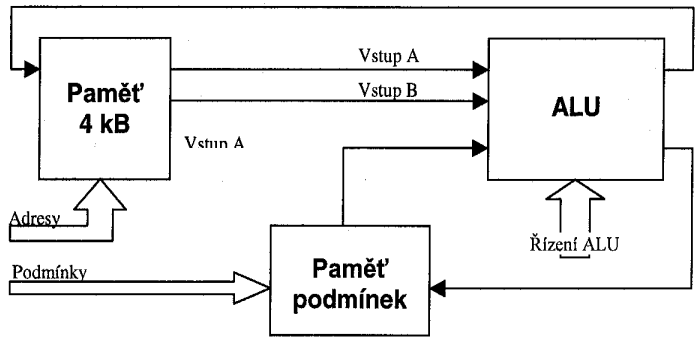
V roce 1970 bylo uvedeno na trh hned několik komerčních strojů používajících organizaci SIMD, avšak nedošly očekávané pozornosti. Její renesance o deset let později v souvislosti s uvedením počítače CM-2[1] firmou Thinking Machines Inc. souvisela s rozvojem technologie výroby integrovaných obvodů VLSI, která v té době dospěla do stavu, kdy několik malých procesorů mohlo být integrováno do jednoho čipu. Samotné tyto procesory byly velmi jednoduché na to, aby mohly soupeřit s jednočipovými procesory pro obecné použití, jako byly např. Motorola 68020 nebo Intel 80386. Avšak ve chvíli, kdy se tisíc těchto procesorů mohlo zapouzdřit do relativně malého prostoru, byl vybudován základ pro stavbu velmi efektivního systému. Například 32 procesorů bylo možno zapouzdřit do jednoho čipu a 32 těchto čipů mohlo být umístěno na jedné desce plošného spoje a tak se 1024 procesorů a jejich pamětí vtěsnalo na plochu jen o málo převyšující 36 dm^2.

Procesor počítače CM-2 obsahoval jednobitovou aritmeticko-logickou jednotku se dvěma vstupy operandů a jedním vstupem podmínky (viz obr. 131). Paměť operandů je realizována jako dvoubránová, tzn. že umožňuje současné čtení obou operandů - bitů. Dvanáctibitové adresy operandů jsou předávány řadičem podobně jako ovládání operací aritmeticko-logické jednotky. Paměť podmínek je pouze šestnáctibitová a slouží rovněž k uložení záznamu o přenosu do vyššího řádu.

V počítači CM-2 bylo použito 4096 takových procesorových modulů propojených do dvanáctirozměrné krychle, v níž se informace přenášely ve formě paketů. Všechny moduly jsou napojeny na speciální jednobitovou sběrnici, která může provádět některé operace výběru po bitech ve všech procesorech najednou (např. operace výběru minima nebo maxima). Celá soustava procesorů pamětí a propojovacích kanálů byla připojena na řídicí počítač VAX firmy DEC. Běžné výpočty zvládal procesor CM-2 rychlostí 2,5 GFlops, avšak jeho rychlost se

[1] Zkratka pro Connection Machine

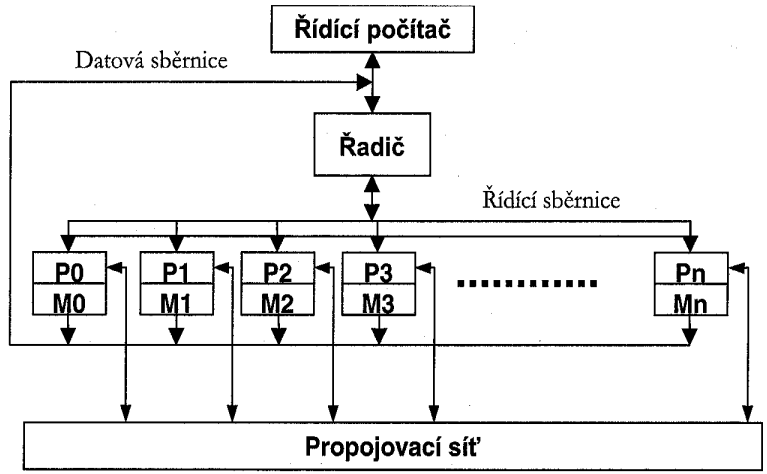
pracujícím s až pětinásobným výkonem.



Obr. 131 Procesor počítače CM-2

10.1.5 Maticové počítače

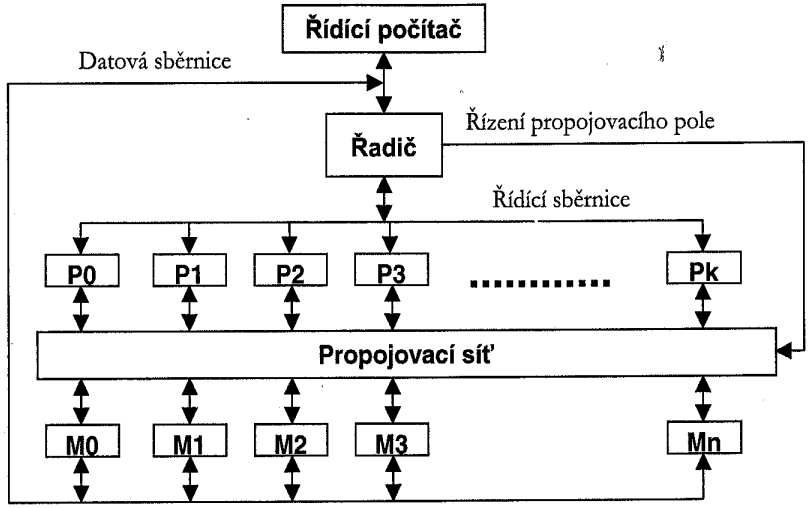
Významné místo mezi stroji pracujícími v organizaci SIMD zaujímají maticové procesory. Pro operace s maticemi je totiž charakteristické, že se zpracovává větší počet prvků současně, přičemž se všemi se provádí stejná operace. Tedy nejvhodnější organizací procesoru pro tento druh výpočtu bude zobecněný typ SIMD, kde každý procesor bude vybaven vlastní pamětí, popř. všechny procesory mohou sdílet společnou, modulárně organizovanou, hlavní paměť. Hlavní rozdíl mezi tímto typem organizace SIMD a organizací MIMD spočívá v umístění propojovací sítě.



Obr. 132 Organizace maticového počítače s lokálními pamětmi

Organizace maticového počítače s lokálními pamětmi je na obr. 132. Každý procesor P_i je vybaven vlastní lokální pamětí M_i, která je přístupná pouze tomuto procesoru. Z tohoto důvodu musí řadič synchronizovat práci systému tak, aby před tím, něž vydá povel k zahájení výpočtu zajistil naplnění všech použitých lokálních pamětí správnými daty. K tomu dojde prostřednictvím datové nebo řídicí sběrnice. Datová sběrnice umožňuje přesunutí dat z paměti řídicího počítače do lokálních pamětí procesorů, zatímco řídicí sběrnice je určena pro předávání instrukcí jednotlivým procesorům v matici. Tyto instrukce jsou pro všechny procesory stejné, jedná se o organizaci SIMD, a proto řídicí sběrnici je možno použít pro zápis dat do lokálních pamětí procesorů pouze tehdy, jsou-li zapisovaná data pro všechny procesory stejná. Řídicí počítač, a tedy i programátor, má možnost blokovat práci některých procesorů pomocí speciální masky, a to většinou pouze na dobu jedné instrukce. Pro delší úsek nečinnosti je nutno tuto masku opakovat.

Při operacích s maticemi jsou velmi často zapotřebí k pokračování výpočtu data z několika sousedních prvků matice. Za tím účelem je maticový procesor vybaven propojovací sítí, která umožňuje výměnu dat mezi jednotlivými procesory maticové sítě. Tato síť bývá obvykle statická, takže maticové procesory mají neměnnou strukturu, avšak řadič systému může být navržen tak, že řídí jednotlivá spojení, a tak jej lze využít i k řízení případné dynamické spojovací sítě. Vlastní program pro maticový počítač je uložen v nadřazeném řídícím počítači a výpočet probíhá tak, že tento nadřazený počítač přesune část programu, která se má provádět v maticovém procesoru do řadiče. Tam je provedena analýza této části programu, připravena data v lokálních pamětech procesorů maticové sítě a rozhodnuto o místě, kde budou instrukce prováděny - skalární instrukce v nadřazeném počítači, vektorové instrukce v maticové síti. Po spuštění výpočtu koordinuje řadič práci jednotlivých prvků sítě s nadřazeným počítačem.



Obr. 133 Organizace maticového procesoru se sdílenou pamětí

Poměrně obtížnou manipulaci s daty, jejich přiřazování jednotlivým lokálním pamětem a procesorům a přesouvání během výpočtu se snažili konstruktéři řešit přemístěním propojovací sítě mezi procesory a paměti jak je ukázáno na obr. 133. Procesory maticové sítě jsou odděleny od paměťových modulů přepínací sítí, což umožňuje použít větší množství paměťových modulů než procesorů a používat tyto moduly dle potřeby pro různé procesory. Toto je výhodné zejména tam, kde mají procesory pracovat s prvky matic, které jsou uloženy v různých paměťových modulech. Propojovací síť sice přináší do práce systému určité zpoždění, to je však eliminováno flexibilitou přiřazení procesorů pamětem, a tedy snížením potřebného objemu meziprocesorové komunikace.

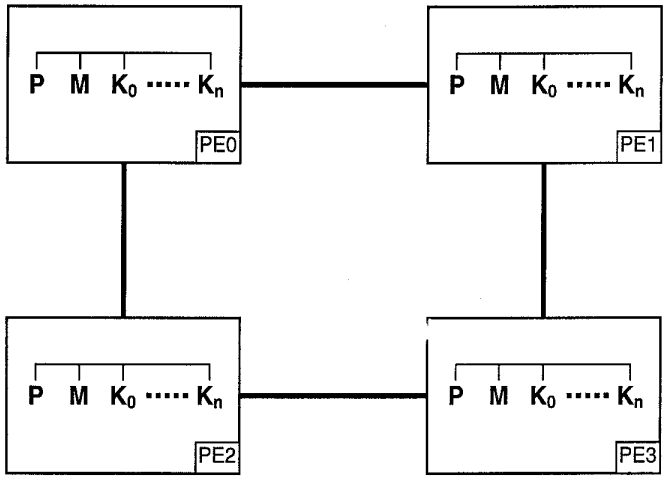
Maticové procesory SIMD se používají ve všech oblastech, kde se uplatní matematické modely založené na práci s maticemi, které zhusta používají základní maticové operace typu násobení, rozklad nebo inverze matic a výpočet vlastní hodnoty matice. Mezi aplikační oblasti bude tedy patřit zejména lineární a celočíselné programování, modelování v meteorologii, signálové procesory používané pro filtraci, Fourierovu analýzu, zpracování obrazů a analýza scény v reálném čase spolu s uplatněním při zpracování map v geodézii a kartografii.

10.1.6 Distribuované paměti

Podobně jako v maticovém počítači je lokální paměť přiřazena jednoúčelovému procesoru, v distribuovaném paměťovém systému je paměť spojena s univerzálním lokálním procesorem. Stejně tak v tomto případě procesor může adresovat pouze svoji vlastní paměť. Někteří autoři nazývají tento typ **multipočítačovým systémem**, v čemž se odráží skutečnost, že stavební bloky systému jsou v podstatě samostatnými malými počítači vybavenými jak procesorem, tak pamětí.

Tento druh organizace má několik výhod, které vyplývají ze způsobu komunikace jednotlivých elementů. Je zřejmé, že odpadá nutnost arbitráže požadavků na sběrnice nebo požadavky na přepínače a každý procesor může používat plnou šířku pásma přístupu ke své vlastní lokální paměti bez toho, aby docházelo k interferenci s jinými procesory. To, že neexistuje společná sběrnice, znamená, že rovněž neexistuje praktický limit počtu spojených procesorů. Velikost systému je omezena pouze rozměrem sítě použité k propojení procesorů.

V systémech s distribuovanými paměťmi však leží největší váha na **komunikačním systému**, který zprostředkovává výměnu informací mezi jednotlivými procesory. Jestliže procesor požaduje data z paměti jiného procesoru, musí dojít k výměně zpráv mezi těmito dvěma procesory. Tato skutečnost reprezentuje dva zdroje režie systému a je třeba počítat s dodatečným časem na konstrukci a vyslání zprávy z jednoho procesoru k druhému. Druhý procesor musí přijatou zprávu zpracovat a předat příslušnému procesu.



Obr. 134 PMS diagram paralelního procesor s distribuovanou pamětí

Programování strojů s distribuovanými paměťmi je založeno na organizaci programů jako množiny nezávislých procesů, které mezi sebou komunikují prostřednictvím zpráv. Přitom programátor musí sledovat, kde jsou konkrétní data uložena, což vyžaduje lokalita zpracování paralelního algoritmu pracujícího s využitím distribuovaných pamětí. Je zřejmé, že algoritmus, který umožňuje, aby data byla oddělena do diskrétních jednotek a potom zpracovávána s minimálními nároky na komunikaci mezi jednotkami dat, bude daleko účinnější a efektivnější než algoritmy, které budou požadovat náhodný přístup ke globálním strukturám.

Paralelní programovací techniky používající např. "semafory[1]" nebudou přímo aplikovatelné v systémech s distribuovanými paměťmi, avšak mohou být nahrazeny podobnými nebo modifikovanými technikami. Např. uživatelský program může používat "semafor", který je implementován tak, že je vyslána zpráva na uzel v síti procesorů, který má tento semafor přidělen do správy. Takový způsob implementace však není efektivní a zahrnuje vysokou režii systému spočívající v nutnosti operace s lokální pamětí dotazovaného procesoru a s přenosem zpráv mezi oběma procesory.

Která z programovacích technik je vhodnější a účinnější, zda používání sdílené paměti nebo distribuované paměti, je často otázka osobní preference. Technika předávání zpráv lépe odpovídá metodám objektově orientovaného programování, a jestliže program byl popsán objekty a odpovídajícím způsobem navržen, může být velmi jednoduché jej adaptovat pro použití v systému s distribuovanou pamětí. Jestliže jsme nuceni rozhodnout, zdali bude lepší implementovat program ve sdílené paměti nebo v distribuované paměti, rozhodování bude obvykle založeno na množství informací, které musí být sdíleny paralelními úlohami. Tam, kde bude existovat velké množství informací sdílených jednotlivými úlohami, které v systému s distribuovanými pamětmi musí být přenášeny z jednoho uzlu do druhého, bude režie systému na zpracování a předávání zpráv vysoká. V tom případě bude výhodnější použití systému se sdílenou pamětí.

Jednoduchý paralelní procesor s distribuovanou pamětí je pomocí diagramu PMS ukázán na obr. 134. Jednotlivé procesorové elementy jsou propojeny komunikační sítí, jejímž prostřednictvím jsou zprávy předávány

[1] Programovací technika využívající příznaky měněné a testované "zúčastněnými" procesy pro řízení běhu těchto procesů.

i jednoho procesorového elementu ke druhému. Z hlediska jednoho procesorového elementu jsou ostatní ocesory jeho vstupně-výstupními zařízeními. Aby došlo k odeslání informace z jednoho procesorového ementu ke druhému, musí procesor poskytnout informaci ve své lokální paměti komunikačnímu řadiči, který jistí přenos informace na externí zařízení - komunikační síť. Blok dat je přenesený prostřednictvím omunikační sítě až k řadiči přijímajícího uzlu a ten zajistí umístění přijaté zprávy do lokální paměti přijímajícího ocesoru a informuje tento procesor o tom, že byla přijata nová zpráva.

0.1.7 Porovnání systémů SIMD a MIMD

Aplikace systémů s organizací SIMD a MIMD je v mnohém velmi podobná, avšak dosahované výkony a ogramovací techniky se budou v obou případech velmi lišit. Pokud budeme předpokládat, že k sestavení ogramu pro oba typy organizací dochází na úrovni programátora a nejsou specifikovány speciální instrukce, k lze zásadní odlišnost sestaveného programu shrnout následovně:

- u procesoru typu MIMD je pro každý procesor sestaven vlastní program, který se podobá programu pro skalární, jednoprocesorový systém. Protože však procesy v systému MIMD musí spolu komunikovat, budou navržené programy obsahovat synchronizační mechanismy využívající hardware systému pro zajištění koordinované práce. Program je většinou ukládán v kopiích pro každý stroj zvlášť, avšak v případech, kdy se práce jednotlivých procesorů liší pouze v několika parametrech, pak např. transputerové[1] systémy umožňují spuštění jednoho procesu, který generuje procesy pro všechny procesory zapojené v síti.
- příprava programu pro procesor typu SIMD bude principiálně odlišná od programování skalárního jednoprocesorového stroje, protože každá instrukce spouští cyklus zpracování na mnoha procesorech současně. V takovém případě bude existovat pouze jeden program pro celý stroj, čímž se zmenší velikost paměti potřebná pro uložení programů.

Oba typy systémů používají různé způsoby řízení. Systémy SIMD jsou orientovány na centralizované řízení diným řadičem, a tedy obvody vyzdvižení a dekódování instrukcí popř. mikroinstrukcí jsou obsaženy v celém troji pouze jednou. Naproti tomu systémy MIMD jsou velmi decentralizovány, a pokud je použit společný ystémový řadič, pak je to jenom z důvodů koordinace procesů běžících na jednotlivých procesorech[2]. Procesory racující s organizací SIMD jsou již z hlediska svého návrhu synchronně pracující systémy a na rozdíl od systémů MIMD není nutno věnovat synchronizaci práce procesorů specifickou pozornost. U organizací MIMD edstavuje synchronizace dodatečnou režii programu a většinou se realizuje pomocí speciálních instrukcí.

Z obecného hlediska lze prohlásit, že systémy MIMD jsou obecnější, a tedy i aplikační segment tohoto typu očítačů je širší. Organizace SIMD jsou nezastupitelné v oblastech, kde základním typem výpočtu je manipulace maticemi, jako např. u zpracování obrazových informací[3].

10.2 Paralelní systémy

Základní myšlenkou přístupu ke zrychlení výpočtu, pokud nepředpokládáme zrychlování na bázi echnologických změn implementace procesorů, je paralelní provádění výpočtu. Možnost vytváření paralelních ystémů se pronikavě zvyšuje se stále snižujícím se rozměrem elektronických prvků, s jejich snižujícím se lektrickým příkonem.

10.2.1 Podmínky pro paralelní provádění výpočtu

Paralelní provádění výpočtů lze zajistit pouze za jistých, předem známých podmínek, které omezují harakter prováděného programu. Automatická kontrola těchto podmínek je obvykle součástí překladače, který

[1] Transputery - speciální mikroprocesory s integrovanými komunikačními kanály určené pro výstavbu multiprocesorových systémů. Jejich návrh pochází od firmy Inmos, výrobcem byl SGS Thompson.

[2] Obvykle jsou však koordinační mechanismy zabudovány do každého procesoru, takže koordinaci je možno zajistit programovými mechanismy založenými na spolupráci procesů.

[3] V této souvislosti je nutno uvést, že procesory Pentium MMX sice používají speciální instrukce pro podporu multimédií, avšak jejich zpracování je odlišné od organizace SIMD. Jedná se tedy spíše o atribut architektury než organizace.

současným prováděním jiných instrukcí. Podmínky, které omezují paralelní provádění instrukcí můžeme shrnou následovně:

- **závislost toku dat** – instrukce s_1 produkuje vstupní data pro instrukci s_2,
- **závislost výstupu** – instrukce s_1 a instrukce s_2 zapisují do téže proměnné,
- **závislost vstupu** – instrukce s_1 přepisuje vstupní data instrukce s_2,
- **vstupně/výstupní závislost** – instrukce s_1 a instrukce s_2 zapisují do téhož souboru.

Další, méně zřejmé závislosti, zahrnují případy, které nejsou snadno analyzovatelné a mohou obsahova skryté závislosti omezující paralelní zpracování programu. Mezi ně patří např.

- závislost indexů (např. indexované indexy),
- indexy, které neobsahují řídící proměnnou cyklu,
- proměnné, které se vyskytují pokaždé s jinými indexy,
- nelineární indexy.

Poznámka

Případy, kdy průběh řízení programu závisí na datech, které tento program zpracovává, označujeme jako závislost řízení. Příkladem může být následující cyklus

```
for (i=1; i<n; i++)
  if (a[i-1] == 0) a[i]=0;
```

10.2.2 Granularita

Možnost jemnějšího dělení úlohy na její jednotlivé části označujeme pojmem "granularita", přičemž tento parametr úlohy určuje stupeň paralelního zpracování jejích jednotlivých částí. Podle hloubky, do které paralelizace zasahuje, rozeznáváme paralelizmus na úrovni

- úlohy,
- programu,
- podprogramu,
- cyklu,
- strojové instrukce,
- mikroinstrukce.

Čím hlouběji bude paralelismus zasahovat, tím propracovanější musí být organizace procesoru. Např. paralelismus na úrovni úloh může být reprezentován síťovým propojením procesorů pod společným síťovým operačním systémem, který je zodpovědný za přidělování úloh ke zpracování jednotlivým procesorům. Tutéž strukturu můžeme použít pro paralelizaci na úrovni programů. Na druhé straně, paralelizmus na úrovni strojové instrukce nebo mikroinstrukce bude vyžadovat speciální obvody v procesoru, které realizují jak spolupráci funkčních bloků procesoru tak jednotlivé funkce. Tyto funkční bloky existují několikanásobně uvnitř procesoru, který používá paralelní zpracování na nejnižší úrovni.

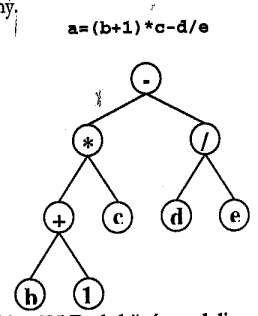
10.2.3 Způsoby paralelního zpracování

Paralelní práce procesoru může být charakterizována několika základními paradigmaty, která se promítají i do vyšších vrstev paralelních programovacích jazyků. V současnosti je nejvíce používán paralelismus na úrovni řízení, i když paralelismy řízené tokem dat nebo událostí jsou rovněž předmětem zájmu zejména v oblasti řídících počítačů.

Paralelizmus řízení je založen na paralelním zpracování procedur nebo příkazů, které jsou doplněny o paralelní konstrukty používající synchronizační primitiva. Základní model stroje, který používá tento typ paralelního zpracování si můžeme představit jako propojení několika běžných procesorů, kde každý z nich vykonává instrukce podle stavu svého vlastního programového čítače. Celkový stav stroje je možno charakterizovat stavem množiny programových čítačů.

Paralelizmus datových toků je případ, kdy datové položky poukazují na exekutibilní kód, který se začne vykonávat teprve tehdy, až všechna potřebná data budou k dispozici. Tento přístup vede k vyhodnocování fragmentů paralelního kódu směrem zespoda a příklad zpracování programového segmentu používajícího paralelizmus datových toků je na obrázku obr. 136. Postup výpočtu je zřejmý.

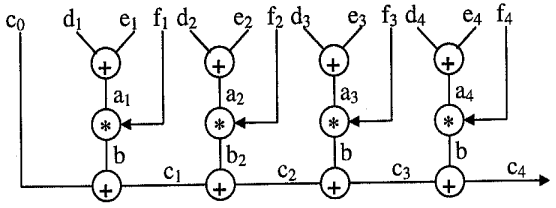
Použití dat k řízení práce procesoru klade značné nároky na provedení procesoru, zejména na mechanismy vyhodnocení přítomnosti požadovaných dat. Data, která jsou vyžadována jako vstupy pro více operací, jsou obvykle požadována v několika kopiích, každé pro jednu instrukci. Nevýhoda řízení práce procesoru tokem dat spočívá v rapacitní[1] strategii chování procesoru, kdy dochází k provádění operací, jejichž výsledky nebudou vůbec potřeba, např. konstrukce příkazu *if-then-else*, kdy obě větve příkazu jsou prováděny paralelně přesto, že vyhodnocení podmínek nutně vede k irelevantnosti jedné větve výpočtu. V konečném dopadu to vede ke snižování efektivnosti práce procesoru řízeného tokem dat.



Obr. 135 Redukční paralelismus

Redukční paralelizmus vznikl jako snaha po odstranění nízké efektivnosti stroje řízeného tokem dat a staví na strategii vyhodnocení podmínek pro odstartování fragmentu kódu na potřebě výsledku tohoto zpracování. Paralelní zpracování jednoduchého výrazu vede ke stromovému rozvoji jednotlivých operací, tak jak je uvedeno v obr. 135. V tomto případě je fragment kódu spuštěn jen tehdy, když jiná operace požaduje výsledek zpracování tohoto fragmentu, tedy hodnotu *a*.

```
input d,e,f;
c[0]=0;
for(i=1;i<=4;i++)
{
  a[i]=d[i]+e[i];
  b[i]=a[i]*f[i];
  c[i]=b[i]+c[i-1];
}
```



Obr. 136 Paralelismus datových toků, schéma provádění výpočtu

10.2.4 Výkon paralelního systému

Pro vyhodnocení výkonu paralelního systému předpokládejme, že existuje program P, který vykonává P_t paralelních instrukcí během dostatečně malého časového intervalu t. Parametr P_t nazýváme **profilem** nebo **paralelním profilem** programu. Počet paralelních operací se může měnit v závislosti na čase, a tak pro program, který poběží v čase od t_0 do t_1 a provede celkem E instrukcí lze tuto hodnotu vyjádřit integrací přes daný časový úsek

$$E = \mu \int_{t_0}^{t_1} P_t \, dt \qquad \text{R - 10-1}$$

kde μ je idealizovaný výpočetní výkon jednoho procesoru. Průměrný paralelizmus úlohy, tedy průměrný počet současně prováděných operací, může být zapsán jako

[1] Z latinského základu; označuje dravost, hrabivost, chamtivost.

$$\phi = \frac{\mu}{t_1 - t_0} \int_{t_0}^{t_1} P_t \, dt \qquad \text{R - 10-2}$$

Označíme-li t_i čas po který je současně používáno i procesorů, pak celkový čas běhu programu T_n na stroji s n procesory můžeme vyjádřit pro okrajové podmínky $n = 1$ a $n = \infty$ jako

$$T_1 = \frac{E}{\mu} \qquad \text{R - 10-3}$$

pro jeden procesor, tedy plně sekvenční zpracování úlohy, a

$$T_\infty = \sum_{i=1}^{\infty} \frac{E_i}{i.\mu} \qquad \text{R - 10-4}$$

pro nekonečný počet procesorů a tedy plně paralelní běh úlohy. Výraz ve jmenovateli přitom reprezentuje okamžitý idealizovaný výpočetní výkon soustavy při použití i procesorů. Teoretické zrychlení výpočtu pro neomezený počet procesorů S_∞, nazývané také **asymptotické zrychlení**, bude odpovídat poměru časů potřebných pro zpracování úlohy jedním procesorem a nekonečně mnoha procesory, tedy

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{E}{\sum_{i=1}^{\infty} \frac{E_i}{i}} \qquad \text{R - 10-5}$$

Výše uvedená zjednodušení mohou být zobecněna na případy celé množiny programů s různými charakteristikami pomocí statistického vyjádření jednotlivých parametrů. Jedním ze základních problémů přitom bude určení celkového výpočetního výkonu v případech, kdy

- je zpracováváno několik programů s různými charakteristikami paralelního chodu,
- v jednom paralelně zpracovávaném programu mají jeho jednotlivé fáze různé charakteristiky.

Označíme-li W_i výpočetní výkon v i-té části z celkového počtu k částí programu, pak **aritmetický průměrný výpočetní výkon** bude

$$W_a = \frac{1}{k} \sum_{i=1}^{k} W_i \qquad \text{R - 10-6}$$

Přestože výpočetní výkon je úměrný převrácené hodnotě času potřebného ke zpracování úlohy, převrácená hodnota průměrného výpočetního výkonu nebude reprezentovat celkový čas výpočtu. **Geometrický průměrný výpočetní výkon**, který se někdy používá pro srovnání různých strojů, může být vyjádřen jako

$$W_g = \sqrt[k]{\prod_{i=1}^{k} W_i} \qquad \text{R - 10-7}$$

opět nebude reprezentovat celkový čas výpočtu. Prakticky jediným použitelným parametrem je **harmonický průměr**

$$W_h = \frac{k}{\sum_{i=1}^{k} \frac{1}{W_i}} \qquad \text{R - 10-8}$$

který odpovídá celkovému počtu operací dělenému celkovým časem zpracování a tak nejlépe reprezentuje míru průměrného výpočetního výkonu. Pro takto určené průměrné výpočetní výkony bude platit Cauchyova věta

$$W_a > W_g > W_h .$$

Z praktických důvodů se rovněž používá vážený harmonický průměr, přiznávající každé hodnotě W_i nějakou váhu w_i. Harmonický průměr definovaný výše je základem ke stanovení **průměrného harmonického zrychlení výpočtu** při paralelním provádění programu na n procesorech. Jestliže program může běžet kdykoliv na kterémkoli z procesorů, pak pro čas sekvenčního běhu programu na jednom procesoru platí

$$T_1 = \frac{1}{W_1} = 1 \qquad \textbf{R - 10-9}$$

a idealizovaný výkon stroje plně využívajícího i procesorů se promítne do celkového času běhu programu

$$T_i = \frac{1}{W_i} = \frac{1}{i} \qquad \textbf{R - 10-10}$$

Předpokládejme nyní, že program je vykonáván v n různých modech zahrnujících $1, 2, 3, ..., n$ procesorů a jsou přiděleny váhy w_i pro i= 1, 2, 3, ... , n. Pak průměrné harmonické zrychlení stroje bude definováno jako

$$S = \frac{1}{\sum_i n \cdot \frac{w_i}{W_i}} \qquad \textbf{R - 10-11}$$

za předpokladu znalosti vah w_i. Použijeme-li předcházející vztahy, pak můžeme nalézt několik okrajových stavů paralelního systému, které se budou lišit maximálním dosažitelným zrychlením výpočtu:

1. Paralelní n-procesorový stroj může používat libovolný stupeň paralelizace (tedy libovolný počet procesorů) ke zpracování $1/n$ celé úlohy. Pak použité váhy budou stejné

$$w_1 = w_2 = \ldots = \frac{1}{n} \qquad \textbf{R - 10-12}$$

Průměrné harmonické zrychlení je

$$S_n = \frac{1}{\sum_{i=1}^{n} \frac{1}{i \cdot n}} = \frac{n}{\Psi(n) + C} \qquad \textbf{R - 10-13}$$

kde $\Psi(n)$ je Gaussova funkce celočíselného argumentu a C je Eulerova konstanta[1]. Po vyčíslení pro některé typické hodnoty počtu procesorů dostáváme např. S_2=1,33, S_{32}=7,88 nebo S_{1024}=136,36. Tyto hodnoty nejsou nijak vysoké a se zvyšujícím počtem procesorů se parametr zrychlení zhoršuje.

2. Paralelní stroj s n procesory dělí zpracování úlohy tak, že každý počet i procesorů (i= 1,2,3, ... ,n) je použit při paralelním zpracování části úlohy tak, že zpracovávaná část je úměrná i. V tomto případě bude podíl času, kdy stroj bude zpracovávat části úloh paralelně, evidentně větší než v předchozím případě. Pro jednotlivé váhy bude platit

$$w_1 = \frac{1}{q}, \quad w_2 = \frac{2}{q}, \quad w_3 = \frac{3}{q}, \cdots, w_n = \frac{n}{q}, \qquad q = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \textbf{R - 10-14}$$

a zrychlení výpočtu bude

$$S_n = \frac{1}{\sum_{i=1}^{n} \frac{i}{i \cdot q}} = \frac{q}{n} = \frac{n+1}{2} \qquad \textbf{R - 10-15}$$

Tento způsob dělení výkonu vede k asymptotické hodnotě zrychlení 50%.

3. Pro pesimistický případ, kdy část práce, kterou bude stroj zpracovávat paralelně, bude úměrná převrácené hodnotě počtu procesorů, tedy

$$w_1 = \frac{n}{q}, \quad w_2 = \frac{n-1}{q}, \quad w_3 = \frac{n-2}{q}, \cdots, w_n = \frac{1}{q} \qquad \textbf{R - 10-16}$$

dojdeme ke vztahu

$$S_n = \frac{1}{\sum_{i=1}^{n} \frac{n+1-i}{i \cdot q}} = \frac{1}{2} \cdot \frac{n(n+1)}{2 - n + \Psi(n) \cdot (n+1) + C(n+1)} \qquad \textbf{R - 10-17}$$

Podobně jako v prvním případě, $\Psi(n)$ je Gaussova funkce celočíselného argumentu a C je Eulerov konstanta. Dosažená hodnota zrychlení je podstatně horší než v prvním případě, což můžeme opě porovnat pro typické hodnoty n, S_2=1,2, S_{32}=5,18 nebo S_{1024}=78,64.

4. Posledním sledovaným případem je triviální, avšak běžný případ, kdy část s úlohy je zpracovávána pouz sekvenčně, zatímco zbytek $1-s$ je plně paralelizován n procesory. Přidělení vah jednotlivým modům odpovídá existenci pouze dvou způsobů zpracování – sekvenčního a paralelního a bude se řídit předpisem

$$w_1 = s, \quad w_2 = w_3 = \cdots w_{n-1} = 0, \quad w_n = 1 - s \qquad \textbf{R - 10-18}$$

Výsledné zrychlení bude dáno vztahem

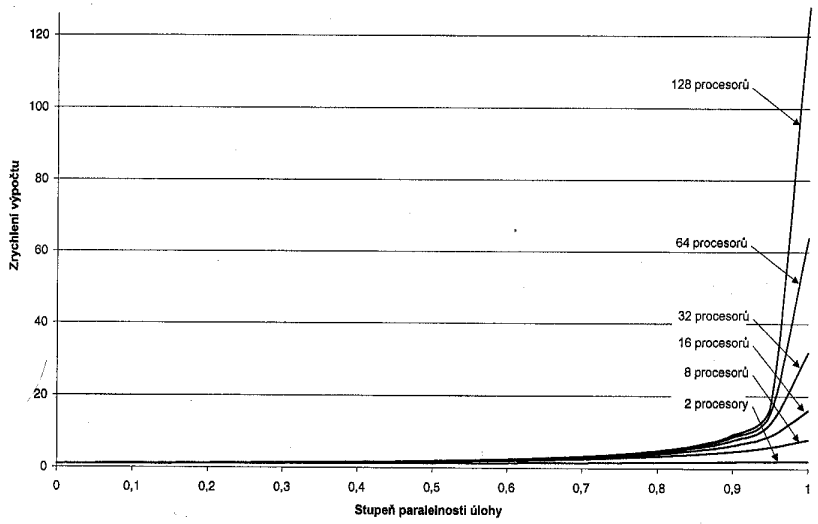
$$S_n = \frac{1}{s + \frac{1-s}{n}} = \frac{n}{n \cdot s + 1 - s} = \frac{n}{s(n-1)+1} \qquad \textbf{R - 10-19}$$

a v limitním případě nekonečného počtu procesorů bude zrychlení

$$\lim_{n \to \infty} S_n = \frac{1}{s} \qquad \textbf{R - 10-20}$$

Tento vztah se nazývá **Amdahlův[1] zákon**.

Grafické znázornění Amdahlova zákona vystihuje závislost zrychlení zpracování úlohy použitím paralelních procesů, které je jednoznačně závislé na stupni paralelizace úlohy, hodnotě p, vynesené na osu x v grafu na obr. 137. Jak z grafu vyplývá, praktické zrychlení výpočtu je možno očekávat až pro hodnoty paralelizace blízké jedné. Pro hodnoty menší než asi 0,9 je použití většího počtu procesorů problematické, tím spíše, že praktické rozdělení úlohy na příslušný počet částí nebo zajištění plného využití procesorů není vždy řešeno optimálně.



Obr. 137 Grafické vyjádření Amdahlova zákona

[1] Úprava vyplývá ze součtu řady - viz např. Rektorys: Přehled užité matematiky

[1] Gene Amdahl, konstruktér řady System/360 firmy IBM.

0.2.5 Účinnost paralelního zpracování

Při paralelním zpracování úlohy patří mezi rozhodující ukazatele práce stroje **účinnost zpracování úlohy, upeň využití stroje** a **kvalita paralelizace úlohy.** Pro další označme O_n počet operací na n-procesorovém stému a T_n poměrnou dobu zpracování programu v n-procesorovém systému. S ohledem na předcházející ýsledky budeme chápat zrychlení výpočtu jako poměr mezi dobami zpracování téže úlohy na jednom a na n ocesorech. Přitom zřejmě bude platit číselná nerovnost $T_n \le O_n$.

Jestliže si pro tento případ definujeme zrychlení jako

$$S_n = \frac{T_1}{T_n} \quad ; \quad 1 \le S_n \le n \qquad \text{R - 10-21}$$

ak **účinnost paralelního systému** může být definována jako podíl zrychlení na jeden procesor, tedy

$$\eta_n = \frac{S_n}{n} = \frac{T_1}{n \cdot T_n} \qquad \text{R - 10-22}$$

de obvykle platí $\frac{1}{n} \le \eta \le 1$.

'ři realizaci paralelního výpočtu je nutno očekávat existenci nadbytečných mechanismů, např. synchronizaci rocesů, které se u sekvenčního zpracování neprojeví, neboť tam jsou dány implicitně použitou metodou. Toto oplnění paralelního programu dalšími operacemi, které z hlediska jeho sekvenčního protipólu zdají zbytečné, ede k nezbytné **redundanci operací**, kterou lze vyjádřit jako

$$R_n = \frac{O_n}{O_1} \qquad \text{R - 10-23}$$

'ři návrhu stroje je důležitým parametrem poměr cena/výkon, který je většinou určující pro obchodní úspěšnost rojektu. Pro paralelní stroje bude důležité maximální využití výpočetních prostředků, které jsou k dispozici, rotože jinak by výsledný poměr cena/výkon prudce narůstal díky nevyužitým, avšak již investovaným ákladům. **Koeficient využití prostředků** můžeme definovat jako součin účinnosti a redundance paralelního ystému, tedy

$$U_n = R_n \cdot \eta_n = \frac{O_n}{n \cdot T_n} \qquad \text{R - 10-24}$$

Mezi výše definovanými parametry paralelního systému budou platit nerovnosti

$$\frac{1}{n} \le \eta_n \le U_n \le 1 \quad ; \quad 1 \le R_n \le \frac{1}{\eta_n} \le n \qquad \text{R - 10-25}$$

Nakonec lze definovat i kvalitu paralelizace jako podíl positivně přispívajících atributů, tedy zrychlení a účinnosti, k negativním atributů reprezentovaným redundancí

$$Q_n = \frac{S_n \eta_n}{R_n} = \frac{T_1^2}{n \cdot T_n^2 \cdot O_n} \quad ; \quad Q_n < S_n \qquad \text{R - 10-26}$$

V praktických a výjimečných případech může zrychlení být i větší než je celkový počet procesorů, zejména když

- sekvenční program je prováděn v malé paměti a je neustále zatěžován režijními časy výměny stránek ve virtuální paměti,
- sekvenční program je navržen bez ohledu na práci vyrovnávací paměti, takže dochází k opakovaným výpadkům vyrovnávací paměti a celý běh programu je neúměrně zatížen režií systému.

Oba výše uvedené případy **superlineárního** nárůstu rychlosti zpracování programu spočívají v odstranění vysoké režie úlohy, která odpadne po jejím rozdělení na paralelní části, jež mají obvykle daleko menší nároky na paměť.

Poznámka

Jiným příkladem superlineárního nárůstu zrychlení výpočtu může být vyhodnocení vztahu pro logický součet $A_1 \vee A_2$ kde oba operandy mohou nabývat pravdivé hodnoty s pravděpodobností 0,5, avšak průměrná doba pro výpočet hodnoty A_1 se výrazně liší podle toho, zda výsledek je pravdivý či nepravdivý. Předpokládejme, že

$$T(A_i) \begin{cases} \texttt{1s} & \texttt{(průměr) pro } A_i = 1 \\ \texttt{100s} & \texttt{(průměr) pro } A_i = 0 \end{cases}$$

a vyčísleme tabulku pro čtyři možné případy, které mohou nastat se stejnou pravděpodobností

A_1	A_2	sekvenční výpočet (s)	paralelní výpočet (s)
T	T	1 + 0	1
T	F	1 + 0	1
F	T	100 + 1	1
F	F	100 + 100	100
Průměr		75,75	25,75

Paralelní provádění programu na stroji se dvěma procesory povede k průměrnému zrychlení

$$S_2 = \frac{T_1}{T_2} = \frac{75,75}{25,75} = 2,94 \ ,$$

které je větší než počet procesorů. Příčinou tohoto efektu je nepružnost sekvenčního zpracování v daném případě. Pokud bychom chtěli dosáhnout podobného zrychlení na sekvenčním stroji, bylo by nutno realizovat pseudoparalelní zpracování, kdy by průběh operace byl po jedné sekundě zastaven (zatím bez výsledku) a byla by ověřena hodnota A_2, s cílem zjistit, zda je možno již provést příslušné rozhodnutí.

10.2.6 Účinnost paralelních algoritmů

Termínem účinnost paralelních algoritmů označujeme vliv faktorů paralelního výpočtu na výkon cílového stroje. Mezi faktory ovlivňující výkon cílového stroje patří

- granularita (viz strana 178),
- profil paralelního zpracování (viz strana 188),
- způsob komunikace jednotlivých částí stroje,
- zvolený způsob synchronizace procesů,
- společné prvky návrhu,
- požadavky na paměť a její organizace.

Amdahlův zákon očekává velmi pesimistické výsledky při paralelizaci a pro pevné algoritmy s omezenou paralelizovatelností je dosažené zrychlení prakticky minimálně závislé na počtu procesorů. Protože však paralelní stroje jsou používány pro řešení rozsáhlejších úloh, než pouhé vykonávání jednoduchých algoritmů, může být tento závěr zavádějící. Pro fixní algoritmy bude jejich sekvenční část, tedy část kterou není možné paralelizovat, limitovat dosažitelné zrychlení nezávisle na počtu procesorů použitých v paralelních segmentech výpočtu. Jak se tedy projeví vliv složitosti algoritmu na jeho možnosti paralelizace, a tedy i na účinnosti práce paralelního systému?

Pro následující úvahy zaveďme abstraktní parametr s, který nějakým způsobem reprezentuje velikost úlohy, např. pro úlohy s maticemi to může být rozměr matice. Budeme-li tuto úlohu zpracovávat sekvenčním algoritmem, pak bude nutno provést celkový počet operací $w(s)$, který bude odpovídat rozsahu výpočtu - u našeho příkladu při násobení matic bude $w(s) = s^2$. Paralelní procesor však musí počítat s jistou redundancí.

algoritmu v paralelním prostředí včetně nezbytné režie systému zahrnující komunikaci mezi paralelními částmi programu i dobu potřebnou na přidělení paralelizovatelných částí algoritmu odpovídajícím procesorům.

S ohledem na výše uvedené předpoklady můžeme definovat účinnost paralelizace úlohy velikosti s na systému obsahujícím n procesorů jako

$$E = \frac{w(s)}{w(s) + h(s,n)} \qquad \textbf{R - 10-27}$$

která se bude blížit jedné pokud nárůst režie systému vlivem paralelního zpracování se bude blížit k nule. Snahou konstruktérů a programátorů bude udržet tuto hodnotu konstantní bez ohledu na velikost úlohy. Jak však ukazuje vztah odvozený výše, bude pro úlohu pevné velikosti klesat účinnost paralelizace se zvyšujícím se počtem procesorů. To znamená, že pro udržení konstantní účinnosti algoritmu bude vyžadovat paradoxně zvýšení velikosti úlohy se zvýšením počtu procesorů, jak ukazuje vztah

$$w(s) = \frac{E}{1-E} \cdot h(s,n) \qquad \textbf{R - 10-28}$$

Funkci w(s) pro E konstantní nazýváme funkcí **rovnoměrné účinnosti**. Pro daný algoritmus tato funkce popisuje, jak rychle se musí zvětšovat velikost úlohy s nárůstem počtu procesorů, aby bylo zachována konstantní účinnost paralelizace algoritmu.

Příklad účinnosti paralelizace

Předpokládejme, že máme realizovat algoritmus rychlé Fourierovy transformace[1] (FFT) buď na paralelním stroji, jehož propojovací schéma je hyperkrychle nebo na stroji, kde procesory jsou propojeny ve dvourozměrné mříži. Naším úkolem bude vybrat nejefektivnější organizaci stroje při předpokladu, že velikost úlohy se bude měnit. Počet operací nutných k provedení FFT je

$$w(s) = s \cdot \log s$$

kde velikost úlohy s je reprezentována počtem vzorků signálu v časové oblasti. Paralelní implementace na stroji se strukturou hyperkrychle bude vyžadovat

$$h(s,n) = O(n \log n + s \log n)$$

operací, kde $n \log n$ je závislé pouze na počtu procesorů a $s \log n$ je komunikační zátěž v hyperkrychli. Funkce rovnoměrné zátěže bude tedy závislá na druhém členu argumentu, a tak můžeme pro zjištění závislosti na velikosti úlohy použít rovnost

$$w(s) = s \log n \qquad \Rightarrow \qquad s \log s = s \log n,$$

ze které je zřejmé $s = n$. Z tohoto zjištění plyne pro funkci rovnoměrné zátěže ve stroji používajícím propojení v hyperkrychli vztah

$$f_E(n) = O(n \log n).$$

Stejná implementace ve stroji používajícím dvourozměrné planární propojení procesorů ve mříži bude vyžadovat nárůst způsobený paralelizací

$$h_2(s,n) = O(n \log n + s\sqrt{n}),$$

kde komunikační zátěž je opět reprezentována druhým členem. Podobně jako v předchozím případě

$$s \log s = s\sqrt{n} \qquad \Rightarrow \qquad \log s = \sqrt{n} \qquad \Rightarrow \qquad s = 2^{\sqrt{n}}$$

a z toho funkce rovnoměrné účinnosti bude

$$f_E(n) = O(2^{\sqrt{n}} \cdot \sqrt{n}).$$

Porovnáním obou funkcí rovnoměrné účinnosti zjistíme, že pro úlohu se bude velmi dobře přizpůsobovat síť hyperkrychle, zatímco přizpůsobení pro planární síť je horší. Přesto, právě ve většině úloh řešících FFT se paradoxně používají planární sítě, zejména pro svoji jednoduchou konstrukci.

[1] Rychlá Fourierova transformace se používá zejména pro frekvenční analýzu signálů v nejrůznějších oborech.

kvalitní a opačná revize

A. Bloch: "Murphy's Law

11 Netradiční stroje

Pod pojmem netradiční stroje můžeme chápat architektury a organizace počítačů, které se vymyka struktuře popsané Johnem von Neumannem ve čtyřicátých letech. I když by tato kapitola mohla zahrnovat celou řadu procesorů a počítačových struktur popsaných dříve, jako třeba procesory třídy post-risc neb struktury procesorů SIMD, budeme se v následujícím krátkém přehledu zabývat spíše organizacemi, jejichž řízen se vymyká standardním postupům v širším smyslu tohoto slova.

11.1 Asociativní procesory

Procesory typu SIMD, které jsou navrženy pro práci s maticemi byly popsány v 9.3. Tyto tzv. maticov procesory používají sdílenou paměť, ze které vybírají data podle adresy. Pokud však výběr dat bude činěn podl obsahu paměťové buňky, tedy asociativně, pak takové procesory budeme označovat jako asociativní procesor Výběr dat z paměti podle určitého znaku bude výhodný zejména při aplikaci algoritmů, které by jinak tento úko musely provádět pomocí posloupnosti instrukcí prováděných aritmeticko-logickou jednotkou. Ta bývá většino úzkým místem počítače, a tak realizace asociativních operací mimo ALU výrazně zvýší výkon procesor v aplikacích, které takové operace často vyžadují. Podle způsobu, jakým se asociativní operace provádějí můžem rozdělit **asociativní procesory** na **paralelní** a **sériové**, přičemž dalším příznakem pro třídu sériovýc asociativních procesorů je šířka dat použitých při asociativní operaci - bit nebo slovo.

Paralelní asociativní procesor používá plně asociativní paměť, se kterou jsme se již seznámili, a tak může vyhledat odpovídající slova v paměti v jednom kroku. Složitější logické podmínky se většinou vyhodnocu v několika krocích programu asociativního počítače. Sériové asociativní procesory vyhodnocují podmínk pomocí speciálních logických členů seskupených po slovech nebo po bitech. Pokud procesor pracuje sériově p bitech a paralelně po slovech, pak všechna slova jednoho modulu asociativní paměti se zpracovávají současně to tak, že v každém kroku se vyhodnotí stejný bit ve všech slovech pomocí vyhodnocovací jednotky přidělen každému slovu. V případě sériové práce po slovech je vyhodnocovací jednotka pouze jedna a ta postupně přečt a zpracuje všechna slova z paměti. Stupeň paralelizmu použitý v asociativním procesoru při vyhledávání dat bud určovat jeho rychlost, a tím i výkon.

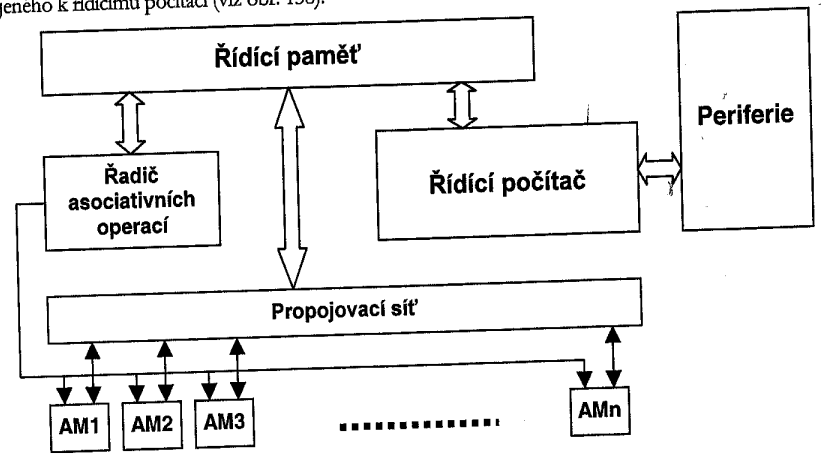
Pomocí asociativních procesorů lze provádět všechny běžné aritmetické a logické operace, avšak hlavn oblasti jeho použití jsou v úlohách, které staví na vyhledání shod nebo podobností v rozsáhlých souborech dat tedy zejména v databázových procesorech. Předpokládáme-li, že existuje několik klíčů a maska, která vybír z každého řádku pole, jenž je určeno k porovnání s klíčem, pak můžeme uvést tyto základní operace vykonávan asociativním procesorem:

- hledání ekvivalence (shoda, neshoda, podobnost ve stanovené toleranci),
- operace porovnání s klíčem (menší než, větší než, menší nebo rovný),
- hledání extrémů (největší a nejmenší hodnota řádku, medián - tedy řádek ležící mezi dvěma řádky v daném uspořádání),
- hledání sousedů (největší hodnota řádku menší než klíč nebo naopak nejmenší hodnota řádku větší než klíč),
- vyhledávání intervalů (řádek, jehož hodnota je menší než horní mez a větší než horní mez, popř. řádek, jehož hodnota vyhovuje více stanoveným intervalům),
- operace uspořádání (sestupné, vzestupné).

Tyto elementární operace nad souborem dat vedou ke snadnému provádění rozsáhlejších rozhodovacích operací a implementaci dotazovacích jazyků. Kromě toho jsou doplněny aritmeticko-logickou jednotkou, která umožňuje realizaci běžných aritmetických operací.

Oblasti aplikace asociativních procesorů se významně shodují s aplikacemi paralelních procesorů, tedy zejména rychlá Fourierova transformace, zpracování signálů a operace nad bázemi dat. Mezi nejznámější asociativní procesory patří počítač **STARAN**, který od roku 1972 vyráběla firma Goodyear Aerospace

orporation[1], jehož jeden maticový modul používal asociativní paměť 256 x 256 bitů s vícenásobným řístupem. Její obsah se vyhodnocoval sériově po bitech a paralelně po slovech tak, že ke každému modulu aměti bylo prostřednictvím propojovací sítě připojeno pole 256 jednobitových procesorů[2]. Pole maticových ıodulů bylo propojeno sítí, umožňující komunikaci mezi jednotlivými moduly a přebírání a instrukcí od řadiče řipojeného k řídícímu počítači (viz obr. 138).



Obr. 138 Struktura počítače STARAN

Tento procesor byl použit například k řízení letecké dopravy v okolí letiště, v oblasti kartografie a pro zpracování obrazů ve strereofotogrametrii. Pro poslední aplikaci byla sestavena konfigurace s 8 216 procesory řízená počítačem CDC 6400. V současnosti je možno nalézt aplikace inovované verze této architektury ve vojenských radarových systémech.

11.2 Počítače řízené událostmi

Přechod od počítačů programovaných propojováním funkčních jednotek ke strojům řízeným posloupností instrukcí byl revoluční změnou, umožňující snadnou změnu zadání a tím i větší univerzálnost stroje. Konstruktéři však hledali nadále mechanismy, které by umožňovaly spouštět výpočet s ohledem na události během výpočtu. Tento způsob řízení se obecně nazývá řízením tokem událostí a podle toho, jaké události jsou příčinou spuštění výpočtu nebo jeho části, jsou dále členěny na počítače řízené tokem dat, tokem požadavků či vnějšími událostmi.

11.2.1 Počítač řízený tokem dat

Základní myšlenkou počítače řízeného tokem dat je spuštění každé instrukce v okamžiku, kdy jsou k dispozici všechny zdroje k jejímu provedení. To znamená, pokud bude dostatek operačních jednotek, bude tímto spouštěcím impulsem přítomnost všech operandů instrukce - tedy dat. Pro okamžik provedení instrukce tedy není rozhodující, kde se v posloupnosti zápisu instrukcí právě vykonávaná instrukce vyskytuje, zda na začátku či na konci. Počítač řízený tokem dat tedy nepotřebuje ani čítač instrukcí, neboť instrukce jsou vybírány z paměti podle zcela jiného principu.

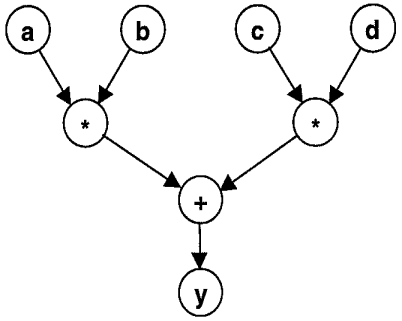
Základní odlišností počítače řízeného tokem dat je způsob jeho programování. Pro objasnění použijeme příklad zpracování součtu součinů, který se v literatuře běžně vyskytuje (viz např. [H1]). Nechť výraz

$$y = a \,.\, b + c \,.\, d$$

[1] Pozdější verze STARAN-A (1974) a STARAN-B.

[2] Tento procesor byl velmi jednoduchý, obsahoval několik klopných obvodů propojených kombinační logikou a umožňoval provést kteroukoli z 16 logických operací.

lna být zpracován pomocí počítače řízeného tokem dat. Pokud budeme tentýž program zpracovávat na klasickém procesoru řízeném tokem instrukcí, pak budeme operace provádět postupně a přitom nebude záležet na tom, zdali jako první provedeme operaci $a \,.\, b$ nebo $c \,.\, d$. Avšak budeme-li mít k dispozici jenom jeden procesor, nemůžeme tyto součiny provést současně. Operaci sčítání můžeme uskutečnit teprve tehdy, až budeme mít k dispozici výsledky obou předchozích operací. Vyznačíme-li si nyní pomocí grafu postup operací vedoucí k výsledku tak, že do stejné úrovně uvedeme operace, které se mohou provádět paralelně, pak obdržíme graf



který reprezentuje postup výpočtu. Naznačené operace budou vykonány vždy, když budou k dispozici zdroje pro danou operaci, což se při dostatečném počtu procesorových jednotek zúží na přítomnost operandů. Sestavený graf přitom jasně znázorňuje, které výpočty mohou být provedeny současně, které jsou závislé na předchozích výpočtech a které na vstupních hodnotách získaných z prostředí (zadaných programátorem nebo předaných vnějším zařízením).

Výše uvedený výpočet, jehož postup lze popsat orientovaným stromem, představuje elementární programový krok počítače řízeného daty. V případě použití pro obecné výpočty musíme počítat s tím, že aplikace bude požadovat použití podmíněných cyklů nebo volání procedur. Pro zápis takového programu se používá grafický formální aparát podobný záznamu obecných vývojových diagramů. Takový grafický zápis se však obtížně předkládá počítači, proto byly navrženy jazyky sloužící pro zápis programů pro tento typ procesorů. Tyto jazyky se vyznačují několika speciálními vlastnostmi, které jsou vyžadovány architekturou počítače[1]. Mezi hlavní rysy patří skutečnosti, že

- operace s daty probíhají lokálně, neexistuje tedy reference, jejíž hodnota by závisela na jiném modulu,
- řazení operací je omezeno pouze na datové závislosti mezi nimi,
- instrukce nemají žádné vedlejší efekty, neboť tyto efekty by se mohly projevit v závislosti na pořadí zpracování instrukcí,
- splňují podmínku jediného přiřazení hodnoty každé proměnné, což je nutné z hlediska vyloučení záměn proměnných a lokalitou účinku operací,
- iterační výpočty jsou prováděny paralelně, podobně jako v případě vektorových procesorů, zavedením indexových značek,
- volání procedur je reentrantní, a tedy nezávislé na historii.

Praktické použití těchto počítačů je široké, avšak většinou aplikace zůstávají u jednoúčelových procesorových struktur, které se vzhledem k možným omezením návrhu lépe realizují. Univerzální počítače řízené daty musí být schopny řešit libovolnou předloženou úlohu a obvykle je dále dělíme na statické počítače řízené daty a na dynamické počítače řízené daty.

11.2.1.1 Jednoúčelové počítače řízené daty

Digitalizace řídících a komunikačních technologií vede k rozšířenému používání jednoúčelových procesorů, které řeší elementární úlohy, neustále se opakující při příchodu nových dat. Událostí, která spustí cyklus takového

[1] Tyto jazyky jsou často navrhovány pro řízení systémů, které zpracovávají řadu vnějších podnětů. Příkladem může být jazyk Erlang, který používá firma Ericsson pro programování svých ústředen.

všechny následující proměnné chápany ve výše uvedeném smyslu jako proměnné související s právě probíhajícím výpočtem. Pokud výpočet má pouze jednoduché operandy, může operace proběhnout okamžitě při příchodu nového operandu. V případě složitějších výpočtů s celou řadou vstupních operandů bude v místech provádění operací s více operandy docházet k automatické synchronizaci výpočtu, neboť příslušná instrukce musí počkat až bude mít k dispozici všechny potřebné zdroje. Příkladem může být jednoduchý procesor řízený daty, který při příchodu dat na svůj vstup vykoná naprogramovanou operaci - interpolační procesor v řadičích mechanický výstupů (zapisovače, obráběcí stroje), směrovací procesory v telefonních ústřednách. Datové procesory (DSP), speciálně orientované na jednu funkci, např. modem, se velmi často vybavují logikou s charakterem daty řízeného počítače (např. modemové čipy firmy Motorola).

11.2.1.2 Statické a dynamické univerzální počítače

Základním znakem statického počítače řízeného tokem dat je příslušnost operandů pouze k jedné úloze a nemožnost rozlišení mezi příslušností operandů v případě výskytu více úloh. To vede k nutnému požadavku, aby se na vstupu operační jednotky statického počítače řízeného daty vyskytovaly pouze operandy příslušející jedné úloze - operaci. Příkladem statického počítače řízeného tokem dat může být struktura dle obr. 139, navržená na MIT. Základem počítače jsou dvě pole - pole paměti instrukcí a pole procesorových elementů oddělená řadičem operací a řadičem výsledků. Instrukce, která je připravena ke zpracování obvody řízení procesoru, je uložena do paměti instrukce tak, že její jednotlivé části obsahují veškeré informace potřebné k provedení instrukce, včetně informace o uložení výsledku. V této paměti instrukce čeká na doplnění operandů. Ty mohou být do příslušné části instrukce zapsány buď obvody řízení procesoru, pokud se jedná o operand z vnějšího prostředí, nebo řadičem výsledků, pokud operand je výsledkem nějaké jiné operace. Jakmile instrukce je doplněna veškerými potřebnými operandy a jediný zdroj, který zbývá k tomu aby mohla být vykonána je procesorový element, začne přidělení procesorového elementu požadovat na řadiči operací.

Řadič operací je autonomní jednotka, která na jedné straně sleduje požadavky přicházející z paměti instrukcí a na druhé straně eviduje volné procesorové elementy. Na základě požadavku je tedy instrukci přidělen procesorový element a proběhne její zpracování. Výsledek je předán řadiči výsledků a ten, podle informace uložené v těle instrukce, rozhodne, jak má být s výsledkem naloženo - např. jej zapíše do dosud volného pole jiné instrukce, pro kterou tento výsledek reprezentuje vstupní data. Podle výše uvedeného algoritmu jsou zpracovávána všechna předložená data.

Podmínkou u tohoto statického počítače je, aby správná data příslušela odpovídající instrukci. Pokud bude více možností, kterým instrukcím budou data příslušet, bude nutno mezi nimi rozlišit a struktura počítače bude muset takovou jednotku zahrnout do své konfigurace. Tato "přidělovací jednotka" potom rozliší podle příznaků v zvláštním poli instrukce a přidělí data se stejnými příznaky stejné instrukci. Podobně na výstupu z pole procesorových elementů musí být zajištěno odebrání výsledku ze správného procesorového elementu.

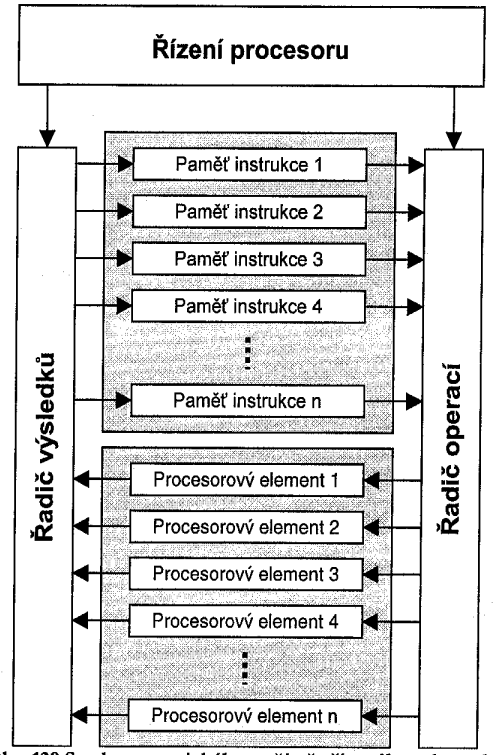
11.2.2 Aplikační charakteristiky počítačů řízených tokem dat

Snaha odstranit prostoje procesorových jednotek v případě, kdy jsou zbržděny čekáním na data a jiné instrukce by mohly být nezávisle zpracovány, byla právě tím hnacím motorem konstruktérů při návrhu počítače řízeného daty. Jak ukazují zkušenosti, při realizaci tohoto typu procesoru na "malém prostoru", tedy s celou řadou omezení, je možno docílit výrazné zvýšení efektivity práce. Proto poměrně snadno pronikly do aplikací jednoúčelové nebo jednoduché procesory navržené v této architektuře.

Univerzální počítače tohoto typu však stojí před celou řadou obtížně řešitelných problémů. Jedním z největších je způsob programování, který není ani tak obtížný svoji formou, ale zcela odlišným přístupem. Rozvrhování procesorových elementů k vykonání instrukcí souvisí úzce právě s programováním stroje a může dojít k tomu, že některé elementy budou přetíženy, zatímco jiné nebudou využity. Nevýhodou je i složitá realizace některých prvků a poměrně velký nárok na paměť programu. Paradoxní je neschopnost efektivně zpracovávat velká pole dat, která vyplývá z apriorní neznalosti metody přístupu k elementům pole během překladu programu - viz obecný požadavek na odstranění vedlejších účinků instrukcí. Jinými slovy, to znamená nemožnost aktualizace proměnné, a tedy každá hodnota proměnné se musí objevit jako nová proměnná. Z tohoto hlediska znamená např. aktualizace pole dat vytvoření nového pole s aktualizovanými prvky.

Velkým nedostatkem procesorů řízených tokem dat je nízká efektivita výpočtu, protože procesor spočít vše co je mu předloženo, tedy provede i výpočty, jejichž výsledky nebudou požadovány. Tato nepříjemn vlastnost je odstraněna u procesorů řízených tokem požadavků, kde se každý mezivýsledek spočítá teprve tehdy když je jeho hodnota požadována. Vzhledem k tomu, že výpočet probíhá vlastně jako redukce programu nahrazování výrazů jejich hodnotami, nazývá se tento typ počítačů řízených tokem požadavků redukčním počítačem. Při tomto způsobu zpracování, kdy předávání dat se šíří proti směru vzniku požadavků, odpadaj operace rozhodování, neboť jsou vzniklému procesu vlastní - pokud data nejsou potřeba, nevyvolá se jejich výpočet.

Většina procesorů navržených v této oblasti vychází z formalizace v jazyce FFP (Formal Functional Programming), která vychází z mapování objektů do jiných objektů. V tomto případě je operátor uplatňován na tzv. sekvenci, která je reprezentována množinou objektů (čísel, symbolů, slov nebo dalších sekvencí) - např. operátor násobení * uplatněný na sekvenci *<4,2>* převede tuto sekvenci na *<8>*. Postupnou redukcí redukovatelných sekvencí v souladu se sémantikou operátorů vznikne výsledek - dále neredukovatelná sekvence. Každá redukce je přitom plně lokální a nemá žádný vztah ke zbytku programu, což vede k automatickému paralelismu výpočtu. K tomu, abychom práci redukčního algoritmu zajistili, musíme použít konfiguraci stroje, která bude mít pravidelnou strukturu složenou z elementárních buněk ve kterých bude možno umístit jeden symbol jazyka nebo jednu numerickou hodnotu. Propojením těchto buněk ve stromové struktuře dosáhneme optimální topologie pro práci redukčního algoritmu. Práce programu potom probíhá v cyklech tak, že vždy se redukují všechny možné sekvence zapsané do buněk interakcí mezi obsahem buněk a příslušným uzlem stromové struktury. Po skončení cyklu vznikne nová sekvence a postup se opakuje. Problémem při konstrukci redukčních počítačů je počet elementárních buněk a jejich přizpůsobení charakteru programu.



Obr. 139 Struktura statického počítače řízeného tokem dat

11.3 Systolická pole

Systolické systémy nebo též systolická pole je pevně propojená síť funkčních jednotek, která je určena pro řešení jedné třídy úloh. Algoritmus výpočtu se zde vlastně realizuje obvodovým řešením a tak systolické pole není možno řídit, nebo jen velmi omezeně, vnějším programem. Funkční jednotky provádějí obvykle stejnou jednoduchou operaci, která vychází z dekompozice řešeného algoritmu. Data se po propojovací síti, která spojuje vždy jen sousední jednotky, rytmicky přesouvají a vnitřní buňky nemají žádný další vstup.

Pojem "systolické pole" vychází ze způsobu práce systému, který připomíná pravidelně pulsující orgán - lidské srdce. Tento rytmický postup práce mezi stejně rychlými procesorovými jednotkami eliminuje nároky kladené v paralelních systémech na výkon paměťového systému, neboť s pamětí jsou propojeny pouze krajní jednotky pole a tedy sladění paměťového systému s pouze jedním procesorem bude snadnější.

Typickou úlohou pro systolické sítě je řešení konvoluce - úlohy, která je definována jako výpočet posloupnosti hodnot $\mathbf{Y} \in \{y_1, y_2, \cdots, y_{n+1-k}\}$ daných vztahem

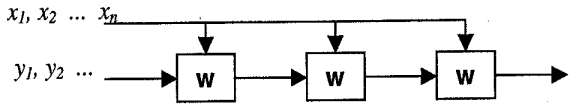
$$y_j = w_1 x_j + w_2 x_{j+1} + \ldots + w_k x_{j+k-1} \qquad \text{R - 11-1}$$

de vstupní hodnoty x a váhy w jsou dány posloupnostmi

$$\mathbf{X} \in \{x_1, x_2, \cdots, x_n\} \qquad \mathbf{W} \in \{w_1, w_2, \cdots, w_k\} \qquad \text{pro } k < n \qquad \text{R - 11-2}$$

Operace tohoto typu se používají zejména v radioelektronice při zpracování signálů, kam patří filtrace, korelace nebo diskrétní Fourierova transformace. Nejjednodušším případem je lineární systolická síť (viz obr. 140), i když e systolické sítě realizují i jako dvourozměrné nebo vícerozměrné sítě. Předpokládáme-li, že při práci budou na vstupy systolické sítě dle obr. 140 periodicky dodávány vstupní hodnoty ve formě posloupností $x_1, x_2, \cdots, x_n$ a $y_1, y_2, \cdots, y_n$, pak obdržíme na výstupu sítě postupně jednotlivé členy konvoluční posloupnosti[1] $x_1 w_3, x_1 w_2 + x_2 w_3, x_1 w_1 + x_2 w_2 + x_3 w_3$.

Další rozšíření strojů, založených na struktuře systolického pole vede k úpravám kdy se např. pohybují váhy , výsledky zůstávají na místě nebo síť je vystavěna jako vícerozměrná. Tak lze realizovat algoritmy pro násobení matice vektorem nebo, ve dvourozměrné síti, násobení dvou matic. Výkon systolického pole je omezen pouze zpožděním v jednotlivých buňkách, které je dáno fyzikálními vlastnostmi realizace. Pravidelná struktura propojováním pouze mezi sousedy je ideální pro vytváření obvodů velké integrace. Většímu rozšíření systolických polí brání zejména neznalost předpisu, který by umožnil převod libovolného algoritmu do práce systolického pole.



Obr. 140 Lineární systolická síť

11.4 Neuronové sítě

Lidský mozek byl vždy výzvou pro konstruktéry počítacích strojů, a tak touha přizpůsobit práci stroje práci lidského mozku konstruktéry neopouštěla. Základní princip práce neuronu byl objeven v polovině tohoto století a v roce 1943 definoval Mc Culloch tzv. formální neuron, který se stal základem snah o výstavbu neuronových počítačů. Tento model neuronu předpokládá, že synapse skutečného neuronu jsou vstupními signály, kterým jsou přiřazeny určité váhy a na které neuron reaguje případnou změnou svého stavu. Neuronové počítače nebo též neuronové sítě jsou pak kombinací těchto elementárních buněk - neuronů.

Základní element neuronové sítě - neuron si můžeme představit jako propojení dle obr. 141, kde jednotlivým signálům $x_1, x_2, \cdots, x_n$ jsou přiřazeny váhy $w_1, w_2, \cdots, w_n$ a přivedeny na vstupy neuronu. Ten sečte jednotlivé vážené vstupy a nastaví svůj výstup podle přenosové funkce, jejíž průběh je rovněž na obr. 141. Tedy funkci neuronu lze popsat výrazem

$$y = N\left(\sum_{i=1}^{n} w_i x_i - p_i\right) \qquad \text{R - 11-3}$$

kde x_i - je vstupní hodnota na i-tém vstupu,

w_i - je váha příslušející i-tému vstupu,

p_i - je prahová hodnota určující okamžik přechodu neuronu z jednoho stavu do druhého.

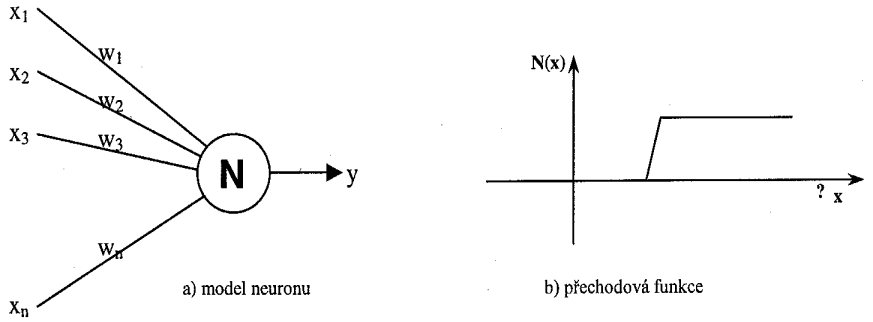
[1] Síť má jenom tři buňky, tudíž jsou uvedeny pouze první tři členy posloupnosti. Případný další rozvoj si čtenář snadno provede sám.

Skoková funkce uvedená v obrázku je však teoretickým zjednodušením a pro praktické účely se nejvíce používá průběh přechodové funkce ve tvaru

$$y = \frac{1}{1 + \exp\left(-\sum_{i=1}^{n} w_i x_i - p_i\right)} \qquad \text{R - 11-4}$$

která je monotónní, hladká, nekonečně derivovatelná a mapuje x-ovou osu do intervalu (0,1) na y-ové ose. Tato funkce má však i další vlastnosti, které ji předurčují pro použití v neuronových sítích. Pokud je signál přicházející ze vstupů silný, pak se blíží k jedné a naopak, pokud je slabý pak se blíží k nule. Nikdy tedy není jednoznačně nula nebo jedna, což se již svým principem přibližuje analogovému typu návrhu. Tato neurčitost je však výhodná v systémech pro rozpoznávání a detekci signálů.

Vytvoříme-li síť vzájemně propojených neuronových buněk, může každý neuron ovlivňovat vstupy ostatních neuronů, které jsou připojeny na výstup dotčeného neuronu. Současně stav celé sítě bude ovlivňován změnami vstupních signálů do okrajových prvků sítě. Výsledná síť se bude chovat jako paralelní dynamický systém, jehož stav bude určen stavy na vstupech do sítě a jejími vnitřními stavy uloženými během práce sítě. V takto vrstvené síti pak můžeme rozeznat **okrajové neurony** (vstupní a výstupní) a **vnitřní - skryté neurony**. Jejich činnost i přechodové funkce se mohou lišit, popř. může být celá síť sestavena z neuronů stejných typů. Podle způsobu šíření jednotlivých signálů v síti lze odlišit jednosměrné sítě, kde signály se šíří v jednom směru od vstupních neuronů k výstupním a obousměrné sítě, kde existují zpětnovazební propojení. Jsou-li jednotlivé neurony mezi sebou zcela propojeny, pak mluvíme o úplné síti. V opačném případě síť označujeme jako neúplnou.



a) model neuronu b) přechodová funkce

Obr. 141 Model formálního neuronu a jeho přechodová funkce

Činnost neuronové sítě lze rozdělit do dvou fází - učící (adaptivní) a aktivní. V učící se fázi se pomocí vzorů vstupních signálů a vzorů požadovaných odezev nastavují váhy jednotlivých spojů a prahy přechodových funkcí neuronů - síť podstupuje trénink. Na místo tréninku mohou být váhy a prahy nastaveny podle předchozího výpočtu. Po této fázi je možno síť použít v aktivním stavu, kdy zpracovává vstupní stimuly. Pokud se v této fázi změní vstupní signály, začne se odpovídajícím způsobem podle předchozího tréninku měnit stav sítě a směřuje ke stabilnímu rovnovážnému stavu, který je požadovanou výstupní informací a může být předán k dalšímu zpracování.

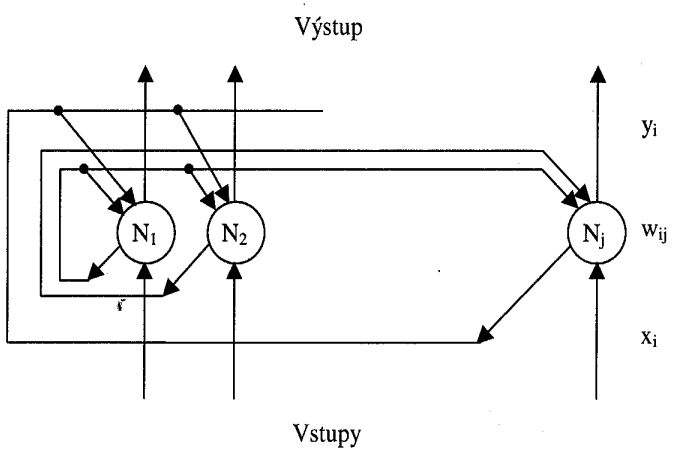
Mezi nejznámější neuronové sítě patří tzv. Hopfieldova síť, viz obr. 142, jejíž typickou vlastností je úplné propojení všech neuronů v síti. Velikost vzájemného ovlivňování neuronů se určuje vahou příslušného spoje. Tyto váhy jsou na počátku práce sítě nastaveny na nulu. Během učení se pro každý neuron nastavují hodnoty vah, které jsou vypočteny při předkládání vzorů. Postupným předkládáním vzorů se nastaví hodnoty prahových funkcí neuronů a vah spojů. Zvláštní vlastností Hopfieldovy sítě je schopnost učit se současně i inverzím vzorů, které jsou jí předkládány. Mezi její omezení patří poměrně malá kapacita vzorů, které lze v síti uchovat, a jejichž počet, při minimalizaci chyby, nesmí převýšit 15% počtu neuronů v síti.

Neuronové sítě se vyznačují celou řadou výhodných vlastností, mezi které patří zejména jejich schopnost
agovat na množinu vstupních dat, která je jen podobná tréninkové množině - tedy se vyznačují jistou
ostrakční schopností. Zároveň, vzhledem k velkému množství spojů a elementů v síti současně se způsobem
racování informace, nevede porucha jednoho prvku v síti k jejímu okamžitému výpadku, ale jenom
postupné degradaci sítě. V současné době směřuje vývoj v této oblasti do dvou realizačních výstupů -
gitálního, který spatřuje cíl v možnosti simulace chování neuronu pomocí standardních prvků výstavby
slicových strojů, a obvodového (analogového), kde funkce neuronu je realizována jako obvodová přenosová
nkce elektronického nebo optoelektronického prvku.

Specifické vlastnosti neuronových sítí umožňují originálním způsobem řešit některé problémy, které by
goritmicky byly řešitelné pouze obtížně, např.

- korekce a rekonstrukce dat narušených šumem při příjmu obrazových dokumentů z kosmických prostředků,
- vyhledávání shody v datech při rozpoznávání scény, tvarů nebo jiných typických signálů při předpokladu, že přesná shoda mezi vzorem a analyzovaným signálem neexistuje (např. diagnostika signálů EKG),
- rozpoznávání závislostí v množině sejmutých dat, což se s výhodou používá v bezpečnostních systémech, kdy neuronová síť připojená na sadu čidel je po tréninku schopna rozpoznat i jiné postupy narušení objektu než ty, které byly trénovány.

Neuronové sítě jsou specifickým přístupem, který je výraznou výjimkou v digitálních počítačových
chnologiích zejména pro svoji neurčitost. Příkladem může být nejistota konvergence sítě pro dané řešení - síť
pak zastaví na okrajových hodnotách, aniž by problém byl uspokojivě vyřešen. S tím souvisí i možnost



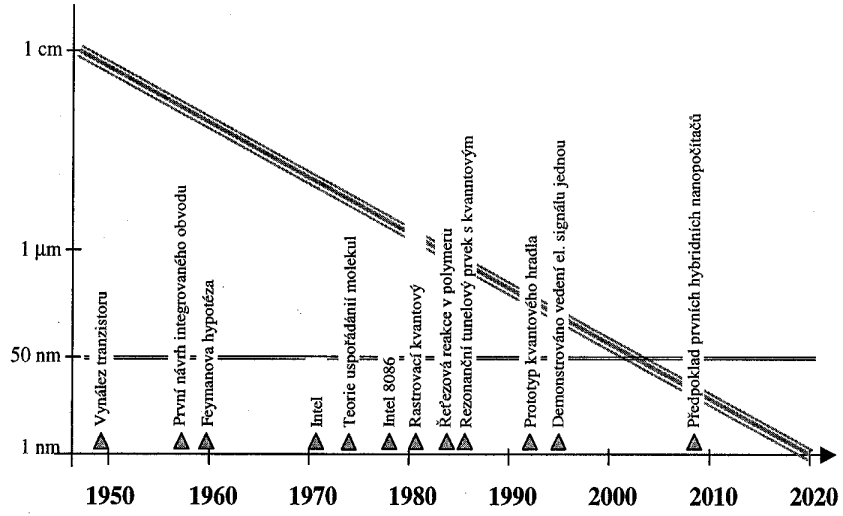
Obr. 142 Schema Hopfieldovy sítě

stavení různých obvodových parametrů různými subjekty při stejném postupu tréninku. Síť totiž může
pokojivě reagovat na tréninkové podněty při více než jednom nastavení vah a prahů a proto stejný problém
že být řešen s různým nastavením. Tomu odpovídá i nejistota okamžiku ukončení tréninku sítě. Dokonalá
alost úlohy, které se má neuronová síť naučit, je jistě velkým pomocníkem v tom, aby trénink nebyl ukončen
edčasně, při nalezení "lokálního maxima" správné činnosti sítě.

činěny omylem.
A. Bloch: "Murphy's Law"

12 Perspektivy dalšího technologického rozvoje

Během posledních čtyřiceti let zaznamenal rozvoj počítačů, založených na elektronických principech, jeden z největších skoků v oblasti nových technologií. Díky objevu transistoru a neustálému snižování jeho rozměrů se redukovaly nejen rozměry počítačů, ale i jejich příkon a spolu s tím přicházely na trh další a další aplikace. Vědcům se podařilo snížit použitelné rozměry prvků integrovaných obvodů na úroveň desetin mikrometru, a tak výrazně zvýšit hustotu prvků na jednom čtverečním milimetru. Podařilo se vyřešit i další problémy spojené s tímto stupněm mikrominiaturizace – jako snížení příkonu a odvod tepla z integrovaného obvodu nebo vytváření plošných motivů jejichž rozměr je srovnatelný s vlnovou délkou světla. Další pokračování miniaturizace však bude vyžadovat vývoj nových technologií neboť rozměry prvků se začínají dostávat na úroveň velikosti molekul a fyzikální principy práce začínají přecházet z oblasti klasické mechaniky do kvantové oblasti. V laboratořích předních světových firem a na univerzitách se ověřují celé řady nových principů, které by mohly být perspektivně použitelné pro počítačové technologie a pro většinu z nich se vžil název **nanotechnologie**.



Obr. 143 Vývoj směřující k nanotechnologiím v porovnání s Moorovým zákonem

První teoretické diskuse zabývající se vývojem a výrobou zařízení s nanometrovými rozměry probíhaly na počátku šedesátých let. Za otce nanotechnologií se považuje fyzik Richard Feynman [N1], který první vyslovil myšlenku sestavování "strojů" pomocí manipulací jednotlivých atomů. Během šedesátých a sedmdesátých let umožnil pokrok v přírodních vědách připravit praktickou realizaci některých Feynmanových myšlenek, zejména v oblasti elektroniky. Moorův[1] zákon, o kterém jsme se zmínili již v úvodu publikace, obhájil svoji platnost a odhady založené na tomto principu určily poměrně přesně překročení mikronové hranice ve výrobě integrovaných obvodů. Vývoj směřující k nanotechnologiím v porovnání s Moorovým zákonem je uveden v obr. 143.

Společně s tím postupoval rozvoj v molekulárních technologiích, který nebyl možná tak zřetelný jako v mikroelektronice, ale byl neméně důležitý. Biochemie a genetika se naučila používat přirozené biologické procesy ke stavbě a manipulaci proteinů i jiných molekul. Jedním z výrazných objevů bylo propojování krátkých

[1] Gordon Moore, jeden ze zakladatelů firmy Intel. Svoje pozorování empirických trendů v rozvoji polovodičových čipů zveřejnil v šedesátých letech. Blíže k pojmu Moorův zákon viz lit. [N2].

ů DNA a RNA[1] v mnohonásobně delší řetězce, což vedlo k rozvoji genetického inženýrství. V oblasti
ntové fyziky se při experimentech v osmdesátých letech podařilo zachytit jednotlivý elektron v potenciální
a objevily se nové přístroje např. rastrový elektronový mikroskop, využívající tunelového jevu, který
žňoval i manipulace s jednotlivými atomy.

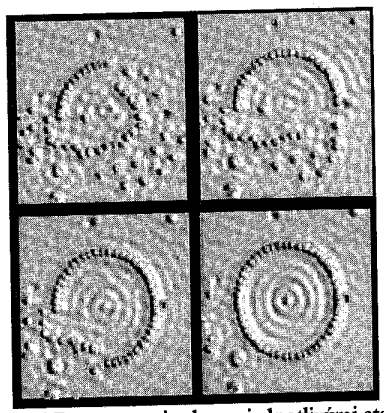
Poznámka

Práce rastrového elektronového mikroskopu, který využívá tunelový jev, spočívá v měření proudu mezi kovovým hrotem se špičkou o rozměru jednoho atomu a vodivým materiálem. Mezi tento kovový hrot a vodivý materiál je přivedeno napětí řádu milivoltů a hrot je umístěn velmi blízko vodivého materiálu (na vzdálenost menší než 10^{-9} m), avšak bez vodivého kontaktu. V těchto nepatrných rozměrech se začnou projevovat zákony kvantové mechaniky a mezi hrotem a nejbližším atomem vodivého materiálu se uplatní tunelový jev, který způsobí, že mezi těmito dvěma body začne téci proud několika pA nebo nA. Tento proud je velmi citlivý k velikosti mezery mezi materiálem a hrotem (např. měřitelná změna proudu o 2% odpovídá změně vzdálenosti o 10^{-12} m). Pohybuje-li se hrot nad materiálem v předem definovaném rastru, můžeme mapovat atomy na povrchu vodivého materiálu. Opět s ohledem na pravidla kvantové fyziky – mikroskop nezobrazí "částici" tak, jak bychom si představovali, ale spíše hustotu pole v místě měření.

Jednou z důležitých aplikací "tunelové mikroskopie" je možnost manipulovat s jednotlivými atomy ve struktuře. Hrot mikroskopu se přiblíží na nejmenší vzdálenost nad atom, který chceme přenést, a zvýší se "tunelový proud" tak, aby vzrostla přitažlivá síla. Atom je hrotem uchopen a přenesen na žádané místo. Snížením "tunelového proudu" se atom umístí do přesného místa na povrchu materiálu. Na obr. 144 je zobrazen postup manipulace s atomy železa na měděném povrchu při teplotě 4 °K. Vzniklý kruhový útvar se nazývá "kvantový korál". Rozměr korálu na obrázku je 14,3 nm.

Kombinace všech těchto nových přístupů vyústila v devadesátých letech v nové odvětví nanotechnologie[2]. zmanitost názorů, postupů a návrhu se projevila širokým přístupem k novým aplikacím a je otázka, který věřovaných principů se nejlépe osvědčí v budoucnosti. V zásadě můžeme aplikace nanotechnologií pro tavbu počítačů rozdělit do čtyř oblastí

- mechanické nanopočítače,
- chemické nanopočítače,
- kvantové nanopočítače a
- elektronické nanopočítače.



Obr. 144 Postup manipulace s jednotlivými atomy pomocí "tunelové mikroskopie".

lnotlivé principy jsou studovány ve světových oratořích a každý z výše uvedených principů má oje zastánce a odpůrce. S ohledem na vývoj posledních padesáti letech lze očekávat, že mární výstavba počítačů bude pokračovat lektronické oblasti s pokračujícím vlivem noelektroniky a nanomechaniky. Očekávaný zvoj a postupné zaplňování počítačového trhu nito produkty lze očekávat v průběhu několika sledujících desítek let. Očekávaný nárůst podílu chto technologií na trhu je znázorněn na obr. 145.

ognostika vývoje informačních technologií je nevděčná. Rychlost s jakou se objevují nové technologie a ganizace je překotná a užitečné poznatky se často ztrácí v moři informačního balastu. V březnu 1949 se, baven znalostmi o počítači ENIAC, vydal na cestu predikce dlouholetého vývoje informačních technologií litor časopisu Popular Mechanics a jeho vize vzdálené budoucnosti stavěla na počítačích, "majících jenom 1500 lektronek a vážících pouze jeden a půl tuny s výkonností jen o málo větší než ENIAC". Podobně jako u většiny edpovědí vývoje v této oblasti, chyba byla šokující. Porovnávat výkonnost ENIACU s dnešními kapesními

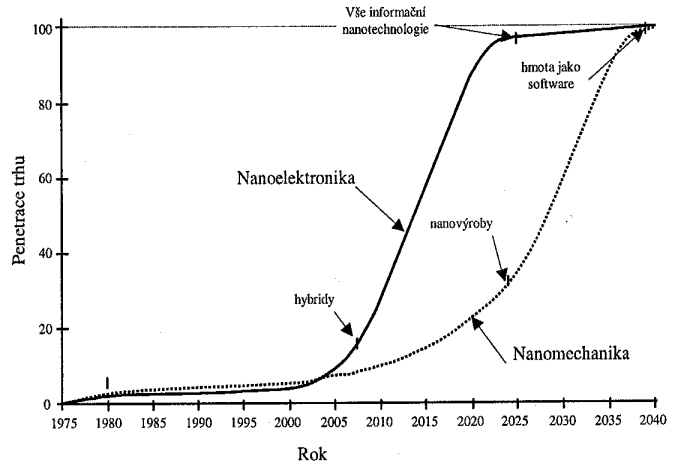
DNA – deoxyribonukleinová kyselina, RNA – ribonukleinová kyselina. Vláknité struktury těchto kyselin jsou nositeli genetické formace.

Základním inspiračním materiálem nového odvětví byla [N3].

kalkulátory nebo notebooky snad ani nelze srovnávat. Proto prosím, aby následující řádky byly chápány jako pouhá extrapolace vývoje, který se však s velkou pravděpodobností nepovede.

12.1 Mechanické nanopočítače

Drexler a jeho spolupracovníci dávali za příklad mechanického nanodesignu konstrukci Babbageova stroje v molekulárních rozměrech [N4]. Předpokládali, že takový stroj by bylo možno postavit prostřednictvím atomové manipulace z molekulárních bloků postupem nazvaným **"mechanosyntéza"**. Po sestavení by stroj pracoval obdobně, jako miniaturní programovatelná verze mechanických kalkulátorů známých z minulosti. Byly sneseny velmi fundované a silné argumenty ve prospěch mechanické konstrukce – např. principy kvantové mechaniky zajišťují, že mezi jednotlivými pohyblivými částmi nebude existovat tření, které znemožňovalo budování rozsáhlejších komplexů na mechanické bázi. Oproti tomu stál argument výroby – takový stroj by byl odkázán na manuální sestavení pomocí tunelového mikroskopu, což by bylo nejen pomalé a obtížné, ale ve větším rozsahu i nespolehlivé. Možnostmi stereospecifické chemické reakce jako výrobního postupu nebo použití pole tunelových mikroskopů pracujících současně na témž produktu byly argumenty zastánců nanovýrob. Největším praktickým problémem však zůstal způsob pohánění a programování těchto strojů.



Obr. 145 Vývoj nanomechanických a nanoelektronických technologií vzhledem k jejich uplatnění na trhu.

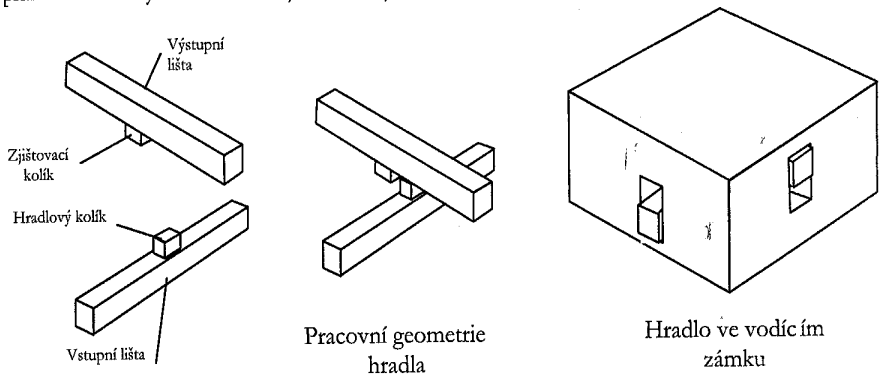
Logická hradla používaná v nanomechanických počítačích navrhl Drexler na bázi "lištové" logiky, používané u mechanických kalkulačních strojů. Zámky vytvořené kolíky nebo zářezy na lištách ovlivňují vzájemný pohyb těchto lišt, a tak vytvářejí logická hradla zpracovávající mechanické ekvivalenty logických proměnných (viz obr. 146). Propagátoři mechanických nanopočítačů šli tak daleko, že navrhli detailní implementace registrů, pamětí RAM, programovatelných logických polí a dalších elementů pomocí lištové logiky. Drexler ve svých odhadech uvádí, že nanomechanický procesor by obsahoval 10^6 vodících zámků, vešel by se do krychle se stranou 400 nm a mohl by používat "hodinový" kmitočet až 1 Ghz pro práci rychlostí asi 1000 MIPS. Příkon systému byl odhadnut na 60 nW což vede k dnes nedosažitelnému parametru 10^{16} instrukcí/s.W.

Víra v nanomechanické počítače a jejich uplatnění v souvislosti se specifickými návrhy a výrobními postupy autory do dnešních dnů neopustila. Eric Drexler, zakladatel a prezident firmy Foresight Institute, Palo Alto, obhajoval návrhy nanomechanických počítačů v americkém senátě v roce 1997. První reálné pokusy o sestavení mechanického nanopočítače proběhly ve spolupráci Foresight Institute a Xerox Rank Corp.

12.2 Chemické nanopočítače

Obecně řečeno, chemický počítač je stroj, který zpracovává informaci tak, že vytváří nebo ruší chemické vazby a výsledné logické stavy – informace uchovává v chemických (např. molekulárních) strukturách. Přitom

ometrů. Obhájci chemických metod při zpracování informace poukazují na biochemicky realizované počítače", které zpracovávají informaci v lidském nebo zvířecím nervovém systému. Umělá výroba nebo plementace takových mechanismů je však věcí ještě velmi vzdálenou.



Obr. 146 Princip mechanického logického hradla

Výrazným pokrokem v této oblasti se však jeví některé experimenty s DNA [N5], kdy složité řetězce této seliny jsou použity k řešení úloh z teorie grafů. Jak uvádí Adleman [N4], segmenty molekulárních řetězců NA mohou být použity pro reprezentaci větví sítě nebo grafu. Kombinace těchto segmentů vytvořená ochemickou reakcí probíhající paralelně na všech molekulách pak popisuje náhodnou cestu grafem. Pomocí ochemických nástrojů lze demonstrovat možnost extrakce správného řešení úlohy z teorie grafů z množiny kto vzniklých řetězců DNA. Navrhovaný stroj na bázi DNA se přitom chová podobně jako masivně paralelní roj s mnoha procesory, který je schopen sledovat množství různých řešení problému v jednom okamžiku. však řetězce DNA implementující toto řešení obsahují řádově více procesních elementů (přibližně 10^{17}) než ejmasivnější elektronický procesor současnosti a mnohem hustěji vyplňují použitý prostor bez vysokých nároků a přiváděnou energii a odvod tepla. Je dlužno připomenout, že Adlemanův počítač je zatím jediný ze všech čtyř upin nanopočítačů, který byl demonstrován na praktické úloze.

Zdá se, že výše uvedená Adlemanova metoda je použitelná pouze pro řešení kombinatorických problémů, šak pozdější práce ukazují možnost rozšíření biochemických metod kalkulace na obecnější úlohy řešené igitálními počítači., např [N6]. Avšak nalezení způsobu implementace rychlých a účinných metod stupu/výstupu, techniky detekce a oprav chyb a celá řada dalších realizačních problémů vede k tomu, že slibná chnologie zůstává stále jen v laboratořích.

Jiný případ použití chemických reakcí, tentokrát pro uložení informací, spočívá v použití proteinu acteriorhodopsinu, který je přítomen v buněčné membráně bakterie halobacterium halobium, žijící v solných ažinách. Tento protein byl vybrán proto, že jeho strukturální změny vlivem světla jsou velmi stabilní[1] a tak se ví výhodné jeho použití pro paměťové prvky. Navržená paměť používá optických charakteristik tohoto roteinu, který je rozptýlen v gelu a zapouzdřen do skleněné kyvety ve tvary anuloidu. Okolo této kyvety jsou místěny tři lasery s různou vlnovou délkou – žlutým, červeným a modrým. Žlutý laser zahajuje zapisovací nebo tecí cyklus paměti a pokud je příslušná molekula ozářena současně i červeným laserem přechází do stavu log 1. čtení je prováděno opět společnou akcí žlutého a červeného laseru, avšak energie červeného laseru je mnohem enší, čímž je docíleno nedestruktivního procesu[2]. Modrý laser je použit pro vymazání informace z paměti. Maximální paměťová kapacita jedné kyvety se pohybuje okolo 1,3 GB při rychlosti přenosu informace až 80 MB/s. Přitom rychlost přechodu jedné molekuly do požadovaného stavu je asi 500 fs[3], omezení rychlosti paměti

Podle [N7] je očekávaná latence asi pět let.

Proces je jenom částečně nedestruktivní, stránku paměti je nutno přepsat vždy po cca 5000 čteních.

fs – femtosekund, t.j. 10^{-15} s.

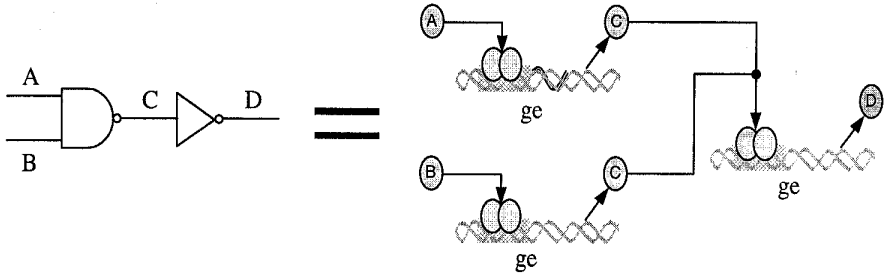
je dáno rychlostí nastavení laseru. V porovnání s moderními pevnými disky se zdá, že tento typ paměti j překonaný. Avšak je nutno si uvědomit, že

- se jedná o paměti konkurující modulům RAM,
- konstrukce paměti je daleko jednodušší, neobsahuje pohyblivé části
- výroba kyvety je mnohem levnější.

Použití biochemického typu paměti ve spojení s procesory pracujícími na biochemických principech může nastavit zcela nový směr vývoje.

12.2.1 Biologické systémy

Metody používané v biochemii se velmi rychle odrazily v myšlence využití biologických materiálů s cílem naprogramovat buňku a použít ji jako element počítacího stroje. Již v roce 1974 zaznamenal Roessler možnost výstavby univerzálního biologického automatu spojením bistabilních chemických reakcí a poukázal na formální příbuznost kinetiky chemické reakce s chováním odpovídajícího elektronického obvodu [M6]. Potenciá takových aplikací je obrovský, uvědomíme-li si, že buňka má nepatrné rozměry[1], je schopná vlastní replikace a hlavně je energeticky nenáročná. Logické operace prováděné v "živé" hmotě jsou většinou implementovány jako chemické aktivity genómů[2]. Analogicky s digitálními obvody pak můžeme přirovnat šíření digitálního signálu k postupu syntézy proteinu v molekule a logickou funkci k produkci nebo rozkladu příslušného proteinu. Tímto způsobem můžeme vystavět biologickou logiku, která odpovídá logickému schématu známému z elektronických digitálních počítačů viz obr. 147. Proteiny v tomto případě můžeme přirovnat vodičům, po kterých se šíř digitální signály a jednotlivá hradla odpovídají genům.



Obr. 147 Ekvivalence elektronické a biologické logiky

Při syntéze logického "obvodu" in vivo se používají existující biochemické mechanismy syntézy a rozkladu proteinů a RNA. Byly prokázány ekvivalentní vlastnosti a ověřeno chování biologických logických prvků jako invertor, hradlo nebo RS klopný obvod [M5]. Jsou rozpracovány metody programování a předávání informací z klasických kalkulačních obvodů do biologických počítačů stejně jako odečítání výsledků biologické kalkulace. Základní nevýhodou je však rychlost zpracování, která omezuje výkonnost konstruované biologické jednotky. Očekává se však, že je zde ještě obrovský skrytý potenciál jak ve vlastnostech buněk[3], tak i v možnosti masivně paralelního zpracování.

12.3 Kvantové nanopočítače

Za první myšlenkou kvantového počítače stál nositel Nobelovy ceny Richard Feynman, který v roce 1982 poukázal na možnost využít přirozených efektů kvantové mechaniky pro návrh a realizaci kvantového počítače. Realizace kvantového počítače na principech vytýčených Feynmanem zůstávala dlouho pouze předmětem

[1] Např. jednobuněčná Escherichia coli má rozměr cca 1x2 μm a do objemu 1 ml je možno umístit 10^9 těchto buněk.

[2] Genóm – soubor genů lokalizovaný v chromozomech buněčného jádra.

[3] Např. dvojitá šroubovice RNA se při štěpení otáčí rychlostí až 60 000 ot/min.

etických studií, avšak s uvedením algoritmů pro prvočíselný rozklad velkých čísel Peterem Shorem z Bell oratories se rázem dostala do popředí zájmu. Shorův algoritmus vhodný pro realizaci na kvantovém počítači ; ukázal na možnost rozluštění dosud odolných kryptografických kódů během několika desítek sekund. ıktivnost řešení a nebezpečí z toho vyplývající motivovala vědce a finančníky[1] z celého světa ke konstrukci ıého kvantového počítače.

V modelu klasického počítače může základní stavební prvek – bit existovat pouze ve dvou stavech logické **0** o logické **1**. V kvantovém počítači může kvantový bit, který se obvykle nazývá **qubit**, existovat nejenom v ıto stavech odpovídajících klasickému počítači, tedy 0 nebo 1, ale může též být v koherentní superpozici ıu logických stavů. Qubit v tomto stavu může být chápán jako entita existující ve dvou prostorech současně,) logická 0 v jednom a jako logická 1 v druhém. Operace prováděné s takovým qubitem budou prováděny časně s oběma hodnotami – tedy tím, že provedeme jednu operaci s jedním qubitem, provedeme ji vlastně časně – paralelně – s oběma jeho hodnotami. Rozšířením této úvahy na dva qubity dojdeme k závěru, že ıstému obsahujícím dva qubity budeme současně provádět operace se čtyřmi hodnotami, v tříqubitovém ému s osmi a pod. Lineární zvyšování počtu qubitů, které se zúčastňují operace, povede k exponenciálnímu ůstu "kvantového paralelismu". Aplikací vhodného typu algoritmu je možno tento přirozený paralelismus žít k řešení některých úloh ve zlomku času, který byl potřebný při řešení stejné úlohy na klasickém počítači.

Výkonnost kvantových počítačů, vyplývající ze zákonů kvantové mechaniky, je však také jejich slabým tem. Jsou velmi nestabilní a obtížně se řídí. Předpokládejme např. že jeden z qubitů je v koherentním stavu. mile však dojde k interakci s okolním prostředím, qubit "vypadne" z koherentního stavu a přejde do jednoho asických stavů. Tento případ se nazývá dekoherence a je kamenem úrazu realizace kvantových počítačů, ot' jejich výkon je závislý na kvantovém paralelismu existujícím jenom v koherentním stavu. A nejenom to – ıterakci s okolním prostředím, a tedy dekoherenci, dochází i při zjišťování stavu qubitu, což ztěžuje, ne-li ımožňuje, získání výsledku výpočtu z kvantového počítače.

Výsledkem práce počítače využívajícího "kvantový paralelismus" je konečná množina různých výsledků ızných prostorech, vyplývající z různých hodnot qubitů v těchto prostorech. Fakt, že výsledek se nenalézá vě v jednom prostoru znamená, že řešení bude výsledkem interference různých výsledků[2].

Příklad

Interferenci řešení je možno objasnit na příkladě průchodu světla dvěma paralelními štěrbinami. Po dopadu na stínítko bude výsledný obrazec složen ze světlých a tmavých pruhů, které jsou výsledkem konstruktivní a destruktivní koherence – tedy v pojmech klasické optiky fázového sčítání a odčítání šířícího se světelného vlnění. Podobně budou konstruktivně a destruktivně interferovat výsledky řešení z každého prostoru, a tak symbolizovat řešení úlohy.

Kvantový počítač ve své reálné podobě bude mít daleko do dnešních počítačů složených z elektronických ıků. Bude nutno vyvinout nové technologie, které umožní udržet qubity v koherentní superpozici obou ických stavů. Zatím neexistuje optimální způsob realizace, avšak mnoho metod se ověřuje a zkouší.

Příkladem implementace qubitu je "kvantový bod", který je tvořen elektronem zachyceným uvnitř speciálně budované sítě atomů. Dopadne-li na tento elektron dávka fotonů vyzářená laserem, jehož vlnová délka a doba ání pulsu jsou přesně stanoveny, elektron se dostane do vybuzeného stavu. Následující puls laseru uvede ktron zpět do klidového stavu. Vybuzený a klidový stav elektronu je možno považovat za logické stavy 1 a 0 a likaci laserového pulsu lze chápat jako řídící proměnnou řízeného hradla **NOT**. Pokud délka pulsu vyzářeného erem bude rovna polovině času požadovaného pro implementaci řízené NOT funkce, elektron se dostane do určitého stavu, který je superpozicí obou mezních stavů a je ekvivalentem koherentního stavu qubitu. Složitější ické funkce lze modelovat sestavováním kvantových bodů do dvojic. Jak z předcházejícího vyplývá, kvantové

[1] Kryptografické kódy se používají, mimo speciální služby, též ke kódování důležitých dat v bankovnictví. Dosud používaný 128 ový kód, jehož odolnost se předpokládá v řádu stovek let při použití nejvýkonnějších počítačů, je s použitím Shorova algoritmu kvantovém počítači rozluštitelný během několika minut.

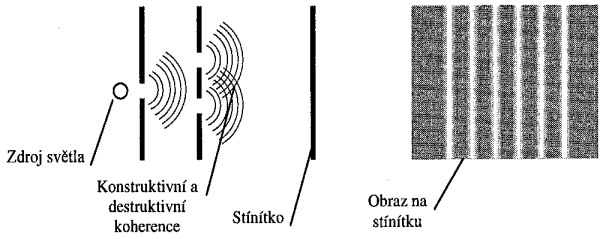
[2] Je důležité poznamenat, že zjišťování stavu výpočtu (nebo výsledku) kvantového počítače zamezuje dalším interferencím, a tedy staví výpočet.

body se jeví jako výhodné prvky pro implementaci kvantového počítače. Avšak existuje mnoho překážek problémů, které stále nejsou uspokojivě vyřešeny a stále jsou překážkou ke stavbě reálného počítače, zejména

- Elektron je ve vybuzeném stavu po relativně krátkou dobu, asi 1 μs, a pak se samovolně vrací do klidového stavu. Uvědomíme-li si, že laserový puls trvá asi 1 ns, pak počet kroků, které mohou být vykonány, než se zpracovávaná informace ztratí, je omezen.
- Vytvoření kvantového bodu je, vzhledem k jeho velikosti, velmi náročná záležitost. Typický kvantový bod dosahuje rozměru asi 10 atomů a technologie pro jeho konstrukci jsou zatím ve vývoji.
- Abychom se vyvarovali montáže tisíců laserů do nepatrného prostoru, kvantové body musí být vyrobeny tak, aby každý z nich reagoval na jinou vlnovou délku. Takový typ přesně a rychle přeladitelného laseru je rovněž ve stadiu vývoje.

Kvantové body však nejsou jedinou formou možné realizace kvantového počítače. Další techniky se pokoušely o použití jednotlivých atomů nebo polarizace světla laseru jako nositele informace. Všechny však měly společný problém – dekoherenci. Přes pokusy např. odstranit dekoherenci ochlazením vzorku na teplotu vzdálenou několik tisícin stupně od absolutní nuly, se k potřebnému výsledku zatím nedospělo.

Radikálně nový přístup ke konstrukci kvantového počítače přinesla myšlenka, že výpočetní médium nemusí nutně být ojedinělá isolovaná částice, ale je možno se stejným efektem použít shluky molekul k ukládání a zpracování informace. Tato myšlenka vycházela ze skutečnosti, že každá částice v atomových jádrech v molekule vykazuje spin v jednom ze dvou možných směrů, který může být použit pro popis stavu jádra. Technika nukleární magnetické rezonance[1] umožňuje detekovat směr spinu jader a elektromagnetický impuls určité vlnové délky může být použit ke změně spinu jádra. Kvantový počítač využívající této techniky je v tomto případě tvořen molekulou a qubity jsou tvořeny jádry atomů v molekule, přičemž je současně aktivován celý shluk molekul kapaliny. Výhodou tohoto přístupu je, že i když molekuly v kapalině do sebe při Braunově pohybu narážejí, spiny jader atomů uvnitř molekuly zůstávají nezměněny. I když dekoherence je stále ještě omezujícím faktorem, doba po kterou zůstává systém v koherentním stavu je mnohem delší, než je možno docílit jinými metodami.



obr. 148 Vznik interferencí po průchodu světla štěrbinou

Existují ještě další metody, které směřují k sestavení experimentálního kvantového počítače. Mimo kvantové body a nukleární magnetickou rezonanci jsou studovány, mimo jiné, např.

- iontové pasti byly původně navrženy jako systém, který umožňoval experimenty s řízením počtu kvantových stupňů volnosti, a přímo se nabízely k implementaci jednobitových a dvoubitových operací pro kvantové počítače. Qubity jsou přitom reprezentovány vnitřními stavy iontu a celý systém může pracovat s proměnným počtem qubitů, výrazně větším, než umožňují metody NMR.
- logická hradla využívající optické mřížky, ve které jsou zachyceny neutrální atomy. Dvojice atomů, které jsou zachyceny v potenciálových jámách pomocí změny polarizace zachycovacích laserů, interagují s elektrickým dipólem, který je vytvořen dalším laserem a tak vytvářejí základní logickou funkci "řízené OR". Tato technika rovněž umožňuje práci s vyšším počtem qubitů.

[1] Nukleární magnetická rezonance (NMR) je metoda používaná ke zkoumání hmoty využívající fenoménu, ke kterému dochází, když je jádro atomu umístěno do statického magnetického pole a vystaveno působení dalšího kmitavého magnetického pole.

- dutinové elektrodynamické rezonátory (QED) využívají interakce mezi pole elementárních částic v nehomogenních a anizotropních oblastech např. optického rezonátoru. Qubity jsou pak reprezentovány např. fotony, jejichž polarizace určuje stav, a které spolu interagují prostřednictvím silných vazeb na jednotlivé atomy v rezonátoru.

Původní idea kvantového počítače byla digitální, díky chápání reprezentace uložené informace jako rétního stavu elementu. Avšak jeden z prvních problémů, který byl teorií kvantového výpočtu sledován, byl č analogový. V roce 1982 si nositel Nobelovy ceny Richard Feynman položil otázku, zda kvantové počítače hou být rychlejší než klasické elektronické počítače při analogové simulaci jiných kvantových systémů. cová simulace je nesmírně náročná, neboť vyžaduje integraci pohybových rovnic v prostoru, jehož rozměr onenciálně roste s počtem proměnných, takže např. simulovat časový vývoj spinu čtyřiceti elementárních tic znamená umocnit matici rozměru $2^{40} \times 2^{40}$ na vektor s rozměrem 2^{40}. Tuto exponenciální explozi složitosti však obejít tak, že použijeme jiný kvantový systém, který bude studovaný kvantový systém simulovat přímo, kterém budou přechody stavy simulátoru řízeny stejnými pohybovými rovnicemi jako přechody mezi stavy tému simulovaného.

Kvantový počítač nebude vždy výrazně rychlejší než klasický počítač, např. operace násobení na kvantovém čtači nebude o mnoho rychlejší než na počítači klasickém. K tomu, abychom dali průchod výraznému chlení práce kvantového počítače, musíme implementovat algoritmus, který plně rozvine potenciál ntového paralelismu – bude vlastně simulovat chování jiného kvantového systému. Tyto algoritmy je obtížné mulovat a mezi nejznámější patří Shorův a Groverův[1] algoritmus. Použitím Shorova algoritmu, který slouží ozkladu čísel na prvočísla, je možné extrémně zkrátit čas výpočtu – rozklad čísla s tisíci číslicemi, ke kterému superpočítač vystavěný klasickou technologií potřeboval 10^{24} let se zkrátí na cca 20 minut.

2.4 Elektronické nanopočítače

Více než padesát let zkušeností a práce s elektronickými počítači, včetně rozsáhlého výzkumu a vývoje éto oblasti, dává největší naději rozvoji elektronickým nanotechnologiím. Výkonnost elektronických počítačů stále zvyšuje a jejich rozměry se v souladu s Moorovým zákonem zmenšují. Komerčně dodávané integrované vody jsou běžně vyráběny technologiemi, které umožňují vytvoření struktur s nejmenším rozměrem 350 nm. den z největších světových výrobců integrovaných obvodů, firma Intel, přechází na technologii ultrafialové ografie, která dovoluje vyrábět prvky se strukturami o rozměru 250 nm a její další stupeň umožňuje zmenšit zlišitelnost až na 180 nm. Metody ultrafialové nebo rentgenové litografie dovolují dosáhnout rozlišitelnosti až 0 nm a ještě vyššího rozlišení je možno dosáhnout přímým "obráběním" polovodiče pomocí elektronového prsku. Optimistické předpoklady hovoří o tom, že se současnými technologiemi je možno dodržet vzestup dle Moorova zákona až do roku 2015, pesimistické tuto možnost popírají.

Budeme-li však extrapolovat křivku Moorova zákona s ohledem na nejmenší dosažitelnou velikost ktronického prvku, měli bychom očekávat rozměry okolo 50 nm na přelomu století a okolo roku 2020 by jmenší rozměr měl dosáhnout 1 nm. Avšak snižování rozměru přináší sebou i jiné než výrobní problémy. kmile klesne rozměr nejmenšího elementu ve struktuře transistoru pod 100nm, začnou se projevovat kvantové vy, jako např.

- zúžení inverzní vrstvy – tato inverzní vrstva tvoří bariéru, která zabraňuje průtoku proudu mezi zdrojovou a odtokovou elektrodou, když je tranzistor pomocí řídící elektrody vypnut. Jakmile rozměr této oblasti klesne pod 100 nm, schopnost inverzní vrstvy blokovat protékající proud se ztrácí.

- tunelový jev – podle zákonů kvantové mechaniky může elektron projít potenciálovou bariérou nejenom tehdy, když je jeho energie větší než energie bariéry, ale také v případě, kdy jeho energie je menší, ale bariéra má dostatečně malý rozměr.

- problémy s odvodem tepla – při práci elektronického prvku vzniká zbytkové teplo, které je nutno odvádět. Při nepatrných rozměrech nanoelektronického prvku vznikají problémy s odvodem tepla..

Nicméně s ohledem na stéle vzrůstající nároky na rychlost procesorů a kapacitu pamětí bude nutno udržet chlost implementačních změn, a tedy je evidentní, že časem bude nutno změnit technologii výroby

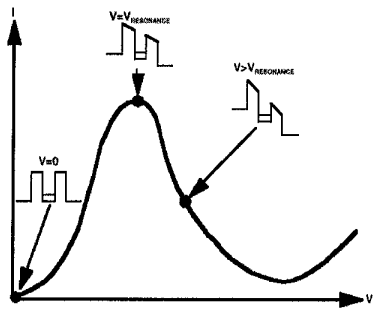
Groverův algoritmus je kvantovým obrazem algoritmů prohledávání ve velkých databázích.

elementárního prvku – tranzistoru. Za tímto účelem bylo navrženo několik způsobů, jak překonat problémy vznikající při zmenšujících se rozměrech a konstrukce využívající principů kvantové mechaniky, např.

- resonanční tunelový tranzistor [M7] je jednou z možností výroby dvoustavových nanoelektronických prvků a jeho základním znakem je vytvoření potenciálové jámy nanometrového rozměru., jejíž hloubka je ovládána řídící elektrodou. U elementární diody může elektron zachycený do této jámy nabývat podle vlnové rovnice pouze určitých energetických hodnot, a tak vznikne uvnitř této potenciálové jámy pouze konečné množství energetických hladin. Vnější elektron, který hodlá projít tímto potenciálovým valem, musí využít tunelového efektu, kde pravděpodobnost, že se to podaří odpovídá poměru energie tohoto elektronu a energetických hladin uvnitř potenciálové jámy. Pokud se energie vnějšího elektronu liší od energetických hladin uvnitř potenciálové jámy proud rezonanční tunelovou diodou (**RTD**) neprojde. Naopak, pokud se shoduje s některou s vnitřních energetických hladin, proud diodou projde a tato vlastnost se projeví na voltampérové charakteristice prvku ostrými rezonančními špičkami odpovídajícími energetickým hladinám. Zabudování takového prvku do struktury bipolárního tranzistoru vznikne nový hybridní prvek, rezonanční tunelový tranzistor, který díky rezonanční charakteristice může nabývat nejen stavu **0** a **1**, ale většího množství takových stavů podle počtu existujících energetických hladin. Vícestavový prvek pak přispívá k redukci počtu elementů nutných k realizaci logické funkce a tak pomáhá ke zvýšení hustoty logických prvků v integrovaném obvodu. Protože RTD[1] je zabudována uvnitř standardního mikroelektronického prvku, tento hybridní prvek vykazuje omezení vlastní klasickým mikroelektronickým zařízením.

- jednoelektronový tranzistor [M8] využívá efektu zachycení nebo uvolnění jediného elektronu nebo díry ze skupiny elektronů (řádu 10^7), tedy práci jednoelektronového tranzistoru nedominují kvantové efekty, jako např. interference mezi několika elektrony. Ve fyzické struktuře jednoelektronového tranzistoru je vytvořen ostrůvek tak malý, že může obsahovat jenom několik volných elektronů a jehož prostřednictvím mohou přetékat elektrony ze zdrojové do odtokové elektrody. Protože tento ostrůvek může obsahovat jenom několik elektronů, další elektrony jsou coloumbovskými silami odpuzovány, a tak nemůže téci větší proud. Počet elektronů na ostrůvku může být řízen vnější elektrodou a tím se dociluje výrazného zesilovacího efektu. Rozměry prvku pohybující se okolo 30 nm jsou ovlivňovány termální energií okolí, která může výrazně omezit tento efekt, a tak byly zatím vyrobeny pouze vzorky pracující za nízkých teplot.

- elektrostatické kvantové body – kvantové body jsou malé potenciální jámy, které elektrostaticky izolují jeden nebo několik elektronů od jejich okolí [M9]. Počet elektronů v kvantovém bodě může být ovlivněn elektrostatickým okolím bodu a jejich současná konstrukce umožňuje rozměry asi 30 nm. Na rozdíl od jednoelektronových tranzistorů je práce kvantových bodů založena na kvantových vlastnostech struktury, např. jeden kvantový bod může elektrostatickou interakcí ovlivnit jiný, i když vzájemně nebudou spojeny. Umístíme-li vedle sebe dva kvantové body a vytvoříme-li cestu, kterou by mohl volný elektron přejít z jednoho kvantového bodu do druhého, obdržíme základní dvoustavový logický prvek. Pro konstrukci takových logických elementů byly navrženy různé konfigurace kvantových bodů, např. pětice kvantových bodů, které obsahují dva elektrony, a logické funkce se tvoří sestavováním elektrostatických kvantových bodů. Protože kvantové body vzájemně komunikují prostřednictvím elektrického pole, a nikoliv proudu elektronů, označuje se tato nová forma elektronického zpracování informací jako "bezdrátová kalkulace". Obtíže spojené s konstrukcí zařízení pracujícího s elektrostatickými kvantovými body jsou podobné jako při konstrukci jednoelektronového



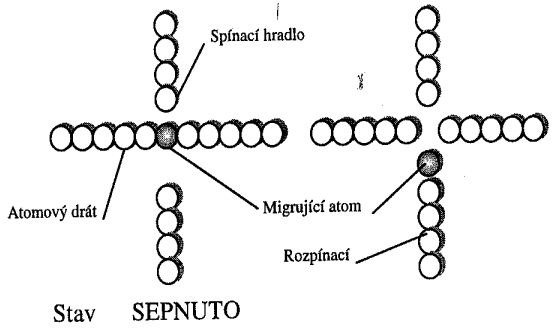
Obr. 149 Průběh jednoho maxima vodivosti rezonanční tunelové diody

[1] Zkratka RTD je používána jak pro diodu tak pro celý tranzistor – Resonant Tunneling Device

Ústřední knihovna

tranzistoru. Nutnost přesného vymezení jednotlivých bodu a možnost výpadku celého výpočtu při přítomnosti malého náboje v blízkém okolí jdou dalšími nevýhodami tohoto způsobu realizace.

- atomové relé [R1], které staví na principiální možnosti vést slabý elektrický proud řadou těsně uložených atomů – tzv. atomovým drátem. Atomové relé obsahuje dva zkřížené atomové dráty, které jsou v místě křížení propojeny migrujícím atomem viz obr. 150. Pokud je atom umístěn mezi tyto dva atomové dráty, atomové relé vede, pokud je odstraněn, spojení je přerušeno. Migrace atomu je řízena pomocí třetího atomového drátu, který probíhá poblíž křížení. Realizaci této myšlenky potvrdil experiment, při kterém byl pomocí rastrového tunelového mikroskopu sestaven bistabilní atomový přepínač.



Obr. 150 Princip atomového relé

Existují i další metody, jako epšené atomové relé, používající ɔtaci skupiny molekul k ovlivnění rotékajícího proudu, nebo mole-ılární přepínač, stavící na změně ıemické vazby uvnitř molekuly. To ɔ však stále je otevřeným polem pro ɛalizaci nanoelektronických systémů, : logický návrh a architektura, jež jsou otřebné k propojení malých a itlivých nanoelektronických prvků cílem provádět spolehlivé a efektivní perace. Protože hustota 10^9 až 10^{12} ɔgických jednotek na 1 mm² je ınohem vyšší než u současných lektronických prvků, bude nutno hledat i nové způsoby návrhu mikroelektronických prvků. Tradiční metoda ýměny informace v obvodu pomocí elektrického signálu přenášeného kovovou nebo dopovanou silikonovou rstvou se z pohledu nanoelektroniky jeví nepoužitelná. Mezi metody, které jsou předmětem studia patří

- celulární přístup[1], kde návrh je založena na dynamice přímých "bezdrátových" lokálních interakcí mezi nanoelektronickým prvkem a jeho okolím [R2]. Tento typ návrhu se nazývá celulární automat nebo v tomto případě kvantový celulární automat. Celý systém prvků sdružených v kvantový celulární automat[2] se pak snaží dosáhnout rovnovážného stavu, ve kterém bude mít nejmenší energii, a na tomto principu je založen mechanismus výpočtu. Očekávaná nevýhoda, že systém může být v rovnovážném stavu i v případě, kdy jeho energie není minimální je řešena modifikacemi této metody[3],

- kvantové celulární neuronové sítě (Q-CNN), které vytvářejí z kvantových bodů analogový počítač nanometrových rozměrů na rozdíl od kvantových celulárních automatů (QCA), které pracují s digitálním modelem. Způsob komunikace mezi jednotlivými prvky je podobný komunikaci v poli buněk QCA. Tento typ celulárních sítí představuje vlastně zcela propojenou neuronovou síť, protože primárně staví na interakci s nejbližšími sousedy, nikoliv na vzdálené nábojové interakci jako QCA.,

- nanometrové nelineární obvody, označované NNN [R3], používají vytváření nepatrných vzájemně propojených kovových ostrůvků , které provádějí kalkulace prostřednictvím nelineárních interakcí mezi jednotlivými elementy. Mezi výhody tohoto přístupu patří vliv nelineárního přenosu náboje mezi jednotlivými elementy, který přispívá k některým kalkulacím a možnost využít tohoto prvku nejenom jako procesoru ale i jako paměti.

Mnohé z uvedených metod a návrhů jsou ověřeny experimentem, jiné jsou teoretickými modely. Současně s těmito modely se připravují optimální technologie výroby, které mohou zahrnovat mechanosyntézu, chemosyntézu nebo klasickou litografii. Otázky optimalizace architektur, způsobů komunikace mezi jednotlivými nanostroji a metody korekce chyb jsou předmětem intensivního studia.

---

[1] V angl. terminologii Wireless Ground State Computing.

[2] Používá se zkratka QCA.

[3] Mezi experimentální modifikace patří tzv. adiabatické [M3] nebo rozprostřené [M4] kalkulace.

13 Měření výkonnosti

stejné.
A. Bloch: "Murphy's Law"

Snad nejrozšířenějším údajem, který reprezentuje interní organizaci stroje a který má vztah k výkonu celého počítače, je rychlost hodin, které řídí rychlost vnitřních operací centrální procesorové jednotky. Čím vyšší je hodinová frekvence, tím více instrukčních cyklů a tím více operací může být provedeno za jednotku času. To platí pro danou architekturu stroje a pro stejné architektury je možné seřadit systémy podle jejich hodinového kmitočtu. Např. procesor Pentium s hodinovým kmitočtem 90 MHz bude zajisté mít menší výkonnost než procesor Pentium stejné architektury, avšak s hodinovým kmitočtem 166 MHz.

Je však několik důvodů, proč pouhým porovnáním frekvence hodinového kmitočtu nemůžeme získat odpovídající údaje o výkonnosti systému. Jedním z těchto důvodů je, že výkon procesorů je závislý na pamětech a sběrnicích, které k němu přivádějí informaci. Tedy velikost a přístupová doba pamětí a šířka pásma sběrnice bude hrát velmi důležitou roli při hodnocení výkonnosti. Bude asi jednoduché si představit, že program, který vyžaduje velké množství paměti, poběží rychleji na počítači, který bude mít větší vyrovnávací a operační paměť, než na počítači, který takovým rozsahem paměti vybaven nebude.

Druhou příčinou je, že vlastní hodinový kmitočet může těžko charakterizovat, co se vlastně děje během instrukčního cyklu. To bude obzvláště pravdivé v okamžiku, kdy budeme porovnávat dva systémy s různými sadami instrukcí, a pak může dojít k paradoxnímu výsledku, že stroj, který bude mít nižší hodinový kmitočet, ale bude vyžadovat méně instrukčních cyklů na provedení téhož programu, dosáhne vyššího výkonu.

Vysvětleme celou situaci na příkladu dvou hypotetických strojů A a B, které se liší pouze v tom, že stroj A má instrukci násobení, kdežto stroj B nikoliv. Jednoduchý cyklus, který násobí vektor skalárem, v našem případě konstantou **3**, je ukázán v následující tabulce, kde je rovněž uveden počet cyklů potřebných k provedení instrukce.

Stroj A			Stroj B		
L:	`load X,V[j]`	2	L:	`load X,V[j]`	2
	`mult X,3`	4		`mov Y,X`	2
	`store X, V[j]`	2		`shl X,1`	2
	`brnz L,i`	2		`add X,Y`	2
				`store X,V[j]`	2
				`brnz L,i`	2
	Celkem cyklů:	10		Celkem cyklů:	12

První instrukce cyklu zavádí element vektoru do vnitřního registru procesoru. Následovně stroj A násobí tento vektorový element číslem **3** s tím, že výsledek je ponechán v registru. Stroj B, který není vybaven instrukcí násobení, provádí tutéž operaci tak, že používá instrukce posuvu a součtu, tedy $3x = 2x + x$. Stroj B kopíruje obsah registru x do jiného registru y, posune x nalevo o 1 bit (tedy vynásobí dvěma), a potom přičte y k tomuto výsledku, takže výsledek operace $3*x$ je opět uložen ve vnitřním registru x. Oba stroje následovně uloží výsledek zpět do místa uložení vektoru v paměti a navrací se na počátek cyklu, pokud index vektoru neukazuje na jeho konec. Jak je vidět, stroj A je rychlejší než stroj B, protože celkový počet cyklů ve smyčce je nižší. Pro bližší představu předpokládejme, že stroj A používá hodiny 9 MHz, zatímco stroj B používá hodiny 10 MHz. Celkový čas průchodu smyčkou u stroje A bude 1,1 us, zatímco u stroje B 1,2 us.

Poznámka

Návrháři mikroprocesorů v sedmdesátých letech používali představu co nejširší sady instrukcí podobně jako stroj A v našem příkladu. Cílem bylo vytvořit instrukční soubor s velkým sémantickým obsahem, např. násobení je relativně

vyplatí s ohledem na snížení časových nároků spojených s vyzvednutím instrukce z paměti, protože bude potřeba méně instrukcí k zajištění téže úlohy. Na začátku osmdesátých let bylo však konstatováno, že instrukční soubor podobný stroji ***B*** je daleko vhodnější pro technologie integrovaných obvodů s vysokou integrací. Přesun směrem k jednodušším instrukčním sadám je znám pod názvem RISC. Fakt, že architektura RISC má méně instrukcí ve svém repertoáru, a každá instrukce je velmi jednoduchá, vede k velmi účinné implementaci ostatních technik, jako je proudové zpracování. Jednoduché instrukce mohou být snadno implementovány a tudíž i hodinový kmitočet může být mnohem vyšší. Zároveň se otevírá prostor pro realizaci dalších jednotek přímo na čipu.

K porovnání dvou strojů s různým instrukčním souborem a třeba i s různým stylem instrukčních souborů můžeme rozdělit celkový čas potřebný pro výpočet do jeho složek. Celkový čas pro daný program bude dán součinem počtu strojových cyklů nutných k vykonání programu a délkou procesorového cyklu.

$$T = n_c \cdot t_c \qquad \textbf{R - 13-1}$$

Dosadíme-li za celkový počet vykonaných cyklů počet instrukcí s průměrnou dobou cyklu na jednu instrukci, obdržíme

$$T = n_i \cdot \frac{n_c}{n_i} \cdot t_c \qquad \textbf{R - 13-2}$$

kde

- n_i - počet instrukcí,
- n_c - počet cyklů,
- t_c - čas potřebný k vykonání cyklu.

Prostřední člen tohoto výrazu popisuje průměrný počet strojových cyklů, který procesor potřebuje pro vykonání každé instrukce a nazývá se součinitelem **CPI**[1]. Základní model pro hodnocení výkonu jednoduchého a jednoprocesorového počítačového systému z předcházejícího výrazu bude

$$T = n_i \cdot CPI \cdot t_c \qquad \textbf{R - 13-3}$$

Každý člen v předcházejícím výrazu popisuje různé atributy související s během programu.

Počet instrukcí závisí na algoritmu a kompilátoru a do jisté míry i na instrukčním souboru daného stroje. Celkový čas běhu programu může být redukován snížením počtu instrukcí, a to buď zvolením lepšího algoritmu, lepšího kompilátoru nebo změnou instrukční sady, takže tentýž algoritmus bude potřebovat ke svému provedení menší počet instrukcí. Avšak s odvoláním na předchozí úvahu s dvěma hypotetickými stroji, kompaktnější kódování nemusí nutně vést ke zrychlení programu, neboť složitější a bohatší instrukční sada obsahuje instrukce, které potřebují pro svůj běh více cyklů. Tato interakce mezi složitostí instrukce a počtem cyklů potřebných k provedení programu je výrazně závislá na aplikaci a je obtížné predikovat, zda přidání nové instrukce skutečně povede ke snížení doby běhu programu, a tedy i zvýšení výkonu.

Koeficient CPI by mohl být označen jako míra složitosti instrukční sady - jednodušší instrukce požadují méně cyklů, takže stroje typu RISC budou vykazovat nižší hodnoty CPI. Toto tvrzení je však zkreslující, protože vychází ze statického kvantitativního pohledu, zatímco výše uvedená rovnice výkonnosti popisuje průměrný počet cyklů na instrukci *měřený po dobu běhu programu.* Zásadní rozdíl si uvědomíme, když uvážíme, že techniky implementace procesorů, jako je např. proudové zpracování instrukcí, umožňují procesoru, aby se jednotlivé instrukce překrývaly, a tudíž jednotlivé jejich části byly vykonávaly paralelně. Takto se velmi sníží CPI a zvýší výkon procesoru, protože za daný časový okamžik se vykoná více instrukcí.

Poznámka

Průměrná instrukce v systému vyžaduje např. tři strojové cykly - jeden na vyzdvednutí instrukce z pamětí, druhý na vyzvednutí operandů z registrů a poslední k provedení operace a uložení výsledku zpět do registru. S ohledem na tento statický popis bychom mohli dojít k závěru, že hodnota CPI je 3,0, protože každá instrukce vyžaduje tři cykly. Avšak

[1] Cycles per Instructon

pro instrukci ***i+1*** a vykonávat se instrukce *i*. Tedy dynamická hodnota CPI, která bude vykazována při běhu programu je jen o málo vyšší než 1,0. Poznamenejme ještě, že tento příklad je možno chápat i jako ilustraci rozdílu mezi rychlostí a šířkou pásma. Celkový výkon systému může být zvýšen zvětšením šířky pásma, tedy zvýšením počtu instrukcí, které zpracovává procesor za jednotku času, bez změny vlastního času provádění jedné instrukce.

Třetí faktor v tomto modelu výkonnosti je doba cyklu procesoru označená t_c. Doba cyklu procesoru závisí zejména na implementace daného hardwaru, rozložení komponentů na povrchu čipu, délky vodičů n typu polovodivého materiálu, který určuje spínací časy logiky. Ale na druhou stranu i architektura a organiz počítače může mít vliv na délku procesorového cyklu. Právě procesory RISC jsou případem, kdy instrukční s je malá, vyžaduje málo logiky pro svoji implementaci, a tudíž i méně prostoru na výsledném čipu. Čím menší j obvody čipu, tím mohou pracovat rychleji a spotřebovávat méně energie. Tak návrh instrukční sady a organiz jednotlivých jednotek procesoru spolu s konkrétní implementací mají vliv na délku cyklu procesoru.

Existují samozřejmě další a dokonalejší metriky používané pro popsání výkonnosti počítačových systé např. **MIPS**[1], je jedním z parametrů, kterým může být procesorový systém popsán. Avšak s ohledem na var ve stylu instrukčních souborů, vnitřní organizaci a počtu procesorů v systému je obvykle pro porovnání d systémů nepoužitelná. Podobné je používané označení **MFLOPS**[2], které je používáno pro porovnání ope zejména matematických částí procesoru.

Příklad

Jestliže známe předem, kolik operací bude potřeba provést při běhu programu, můžeme dělit počet operací dobou jejich provádění, a tak zjistit hodnotu MFLOPS. Standardní algoritmus pro násobení matic s rozměrem n x n požaduje $2n^3 - n$ operací. Potřebujeme-li vypočítat součin dvou matic rozměru 100 x 100 v čase 0,35 s pak výpočtem zjistíme, že potřebný počítač musí mít minimální výkon

$$\frac{2.100^3 - 100}{0.35} = 5{,}714\ MFLOPS\ .$$

Tento typ výpočtu nebo porovnání zcela zřejmě ignoruje i další vlastnosti organizace počítače a může slou pouze jako orientační číslo pro odhad doby provádění výpočtu nebo požadovaných výkonů při zpracov složitějších aritmetických operací v pohyblivé řádové čárce. Proto se často používá standardní sada referenčn programů, kde se porovnávají dva systémy na základě doby potřebné pro zpracování definované úlohy, na Gaussovy eliminace s maticí 100x100 prvků.

Všechny výše uvedené metriky jako CPI, MIPS nebo MFLOPS, sloužící k porovnávání různých stro velmi výrazně závisí na programech, které jsou použity pro měření doby běhu programu. Existují však zvláš soubory programů, tzv. **Bench Mark**, které jsou speciálně napsány pro tyto účely. Jedním z nich je prá **Linpack**, který obsahuje sadu standardních programů z oboru lineární algebry napsaných v jazyce Fortran. P porovnávání výkonnosti paralelních systémů slouží soubory programů **Perfect Club** nebo **Spec**[3], které p hodnocení používají vzorky programů v jazyce C. Často používanou je testovací sada SPEC95, která nahrad původní SPEC92, a obsahuje dvě skupiny testů

- **SPEC CINT95** je sada osmi programů pro testování výkonu stroje v aplikacích, které nepoužív operace a čísla v plovoucí řádové čárce. Mezi programy, které jsou do této sady zařazeny, je naj hra "Go", kterou hraje stroj sám proti sobě, kompilace zdrojového kódu do referenčníl asembleru SPARC, komprese souborů nebo vytvoření a manipulace s databázovými soubory.

[1] Zkratka pro výraz "milión instrukcí za sekundu" - "million instruction per second"

[2] Zkratka pro "million floating point operation per second" - milión operací v plovoucí řádové čárce.

[3] Odvozeno od názvu organizace, zabývající se testováním výkonu počítačových systémů - Standard Performance Evaluation Corporation

- **SPEC CFP95** obsahující deset programů, ktere intenzivne testuji stroj s pouzivanim operaci v pohyblivé řádové čárce. Do této sady jsou zahrnuty převážně vědeckotechnické výpočty z nejrůznějších oborů, např. výpočet průběhu potenciálu v trojrozměrném modelu elektrického pole nebo simulace turbulencí.

ity SPEC jsou pravidelně uveřejňovány pro vybrané stroje ve čtvrtletníku vydávaném Standard Performance aluation Corporation a příklad výsledků testů vybraných strojů uvádí tabulka 1.

ulka 1 Výsledky testů SPEC CINT95 pro vybrané stroje

Výrobce	Typ stroje	Výsledek	Počet procesorů
BULL S.A.	EPC2400 (12 CPU)	2187	12
	EPC2400 (24 CPU)	4337	24
Compaq	Professional Workstation SP750	324	1
Dell Computer	Precision WorkStation 410 (650 M)	553	2
	Precision WorkStation 410 (700 M)	586	2
	Precision WorkStation 420 (600 M)	523	2
	Precision WorkStation 420 (733 M)	631	2
Hewlett-Packard	HP 9000 Model L2000	602	2
	HP 9000 Model L2000	1186	4
IBM Corporation	RISC System/6000 9076-N80 (2 CPU)	249	2
	RISC System/6000 9076-N80 (4 CPU)	493	4
	RISC System/6000 9076-N80 (6 CPU)	724	6
	RISC System/6000 9076-N80 (8 CPU)	966	8
Sun Microsystems	Ultra 80 Model 1450	177	1
	Ultra 80 Model 2450	347	2
	Ultra 80 Model 4450	665	4

spolehlivosti je nespolehlivý.
A. Bloch: "Murphy's Law"

14 Spolehlivost počítačových systémů

Spolehlivost je jedena z nejdůležitějších provozních charakteristik každého systému a jeho elementů. Závisí na složitosti zařízení, počtu a spolehlivosti prvků, ze kterých je zařízení sestaveno. Pro tak složité systémy, jako jsou počítače obsahující tisíce prvků s požadavkem bezporuchového provozu i celé roky, bude na návrháři systému, aby učinil spolehlivostní analýzu systému. Na základě výsledků této analýzy pak musí přijmou řešení spočívající např. ve výběru spolehlivějších, ale tím i dražších, prvků nebo vhodné volby redundance systému. Důležitou roli v tomto případě bude hrát i návrh nejnižší úrovně software, který by měl na zpracování situaci výpadku části systému reagovat a zejména u řídících systémů realizovat strategie směřující k zajištění co nejdelší doby provozu a nejmenších ztrát při výpadku systému.

14.1 Základní pojmy

Spolehlivost systému ***R(t)*** definujeme jako pravděpodobnost toho, že sledovaný systém bude v určitých podmínkách a po určitou dobu ***t*** plnit bez závad svoji funkci. Podmínky, za nichž má systém spolehlivě pracovat, jsou určeny např. rozmezím teplot, tlaku a vlhkosti vzduchu, kolísáním napájecích napětí atd. Funkce systému se považuje za správnou, odpovídá-li jeho činnost plánovaným požadavkům, od kterých mohou být stanoveny přípustné odchylky. Vybočí-li systém z přípustných mezí, kdy soustava přestane správně pracovat, hovoříme o **poruše**. U číslicových počítačů nebo jejich částí se porucha může projevit např. chybou ve výsledku. Poruchy jsou buď **stálé**, způsobené např. trvalým přerušením nebo zkratem, nebo **přechodné** (občasné), vzniklé např. tím, že pracovní podmínky náhodně a krátkodobě vybočí z přípustných mezí.

Přechodná porucha způsobí sice chybu přenášené nebo zpracovávané informace, ale neovlivní vlastní zařízení a jeho další správnou činnost v normálních podmínkách. Stálé poruchy zařízení jsou způsobeny poruchami jeho funkčních prvků a vyřazují zařízení z činnosti nebo znemožňují jeho správné používání. Vlivem stárnutí se pozvolna mění parametry všech fyzických prvků, ze kterých je systém sestaven. Pokud jsou tyto změny v přípustných tolerancích, s nimiž se při návrhu zařízení počítalo, zařízení pracuje normálně; blíží-li se přípustným mezním hodnotám, projevují se nejprve jako přechodné poruchy a po překročení mezních hodnot způsobují stálé poruchy zařízení. Při určování parametrů spolehlivosti prvků se sleduje výskyt stálých poruch.

Veličiny, které charakterizují spolehlivost, mohou být získány z pozorování a statistického vyhodnocení chování objektů. Se zvětšujícím se počtem sledovaných objektů, popř. s rostoucí dobou sledování se hodnoty statistických ukazatelů blíží hodnotám odpovídajícím pravděpodobnostním ukazatelům, které jsou v jistém smyslu matematickou abstrakcí.

14.1.1 Spolehlivost R(t)

Označíme-li si časový interval od uvedení zařízení do provozu do vzniku první poruchy jako ξ, potom pravděpodobnost *spolehlivé činnosti* během doby t je pravděpodobnost toho, že $\xi > t$. Můžeme tedy psát

$$R(t) = P(\xi > t) \qquad \textbf{R - 14-1}$$

Veličina ξ je náhodná proměnná, protože poruchy obojího charakteru se vyskytují v náhodných okamžicích a místech systému. Statistické určení spolehlivosti vychází ze sledování dostatečně velkého počtu objektů - systémů nebo obecněji prvků, po dobu t od okamžiku, kdy byly uvedeny do provozu. Jejich spolehlivost se dá vyjádřit takto

$$R(t) = \frac{N(t)}{N_0} \qquad nebo \qquad R(t) = 1 - \frac{n(t)}{N_0} \qquad \textbf{R - 14-2}$$

kde

- N_0 je počáteční počet prvků uvedených do provozu,
- $N(t)$ je počet prvků, u nichž nevznikla do okamžiku t žádná porucha
- $n(t)$ je počet poruch do okamžiku t

ztahu pro podmíněnou pravděpodobnost. Za předpokladu, že do okamžiku t nedošlo k závadě, je

$$R(t,t+\Delta t) = P(\xi \geq t+\Delta t \mid \xi > t)$$

$$R(t,t+\Delta t) = \frac{R(t+\Delta t)}{R(t)} \qquad \text{R - 14-3}$$

atistická spolehlivost v intervalu (t, t + Δt⟩ bude

$$R(t,t+\Delta t) = \frac{N(t+\Delta t)}{N(t)} \qquad \text{R - 14-4}$$

4.1.2 Pravděpodobnost poruchy Q(t)

Pravděpodobnost opačného jevu - tedy, že v daném čase k poruše dojde

$$Q(t) = P(\xi \leq t) \qquad \text{R - 14-5}$$

což lze upravit s využitím R(t) na

$$Q(t) = 1 - R(t) \qquad \text{R - 14-6}$$

Obdobně lze určit i pravděpodobnost vzniku poruchy v intervalu (t, t + Δt⟩

$$Q(t,t+\Delta t) = 1 - R(t,t+\Delta t) \qquad \text{R - 14-7}$$

4.1.3 Hustota rozložení poruch q(t)

Z definice hustoty pravděpodobnosti plyne

$$q(t) = \frac{d}{dt}Q(t) = -\frac{d}{dt}R(t) \qquad \text{R - 14-8}$$

ož je hustota pravděpodobnosti vzniku chyby k okamžiku t. Pro statistické vyhodnocování zavedeme růměrnou intenzitu, nebo spíše relativní četnost, poruch v intervalu $(t, t + \Delta t)$

$$q(t) = \frac{N(t) - N(t+\Delta t)}{N_0\,\Delta t} = \frac{\Delta n(t,\Delta t)}{N_0\,\Delta t} \qquad \text{R - 14-9}$$

de Δt je dostatečně malý časový interval, během něhož zjišťujeme přírůstek počtu poruch Δn v intervalu $(t, t + \Delta t)$. Dále se zavádí pojem intenzity poruch $\lambda(t)$ v okamžiku t jako

$$\lambda(t) = \frac{q(t)}{R(t)} = \frac{\frac{-dR(t)}{dt}}{R(t)} \qquad \text{R - 14-10}$$

ož je hustota pravděpodobnosti vzniku chyby k okamžiku t za předpokladu, že zatím k chybě nedošlo. tatisticky se hodnota $\lambda(t)$ určí ze vztahu

$$\lambda(t) = \frac{N(t) - N(t,t+\Delta t)}{N(t)\,\Delta t} = \frac{\Delta n(t,\Delta t)}{N(t)\,\Delta t} \qquad \text{R - 14-11}$$

4.1.4 Střední doba bezporuchového provozu - MTBF

Střední doba bezporuchového provozu je matematická naděje doby činnosti do vzniku poruchy, tedy

$$T = \int_0^\infty t\,q(t)\,dt = \int_0^\infty t\frac{dQ(t)}{dt}dt \qquad \text{R - 14-12}$$

ýraz $\frac{dQ(t)}{dt}$ se nazývá **hustota poruch**. Dosadíme-li za $Q(t)$ výraz $1 - R(t)$ a použijeme-li metodu integrace er partes dostaneme

$$T = \int_0^\infty R(t)\,dt \qquad \text{R - 14-13}$$

Statistické určení střední doby bezporuchového provozu pak vede na vztah

$$T = \sum_{i=1}^{N_0} \frac{\xi_i}{N_0} \qquad \text{R - 14-14}$$

nebo

$$T = \sum_{i=1}^{N_0} \frac{(N_0 - i + 1)(\xi_i - \xi_{i-1})}{N_0} \qquad \text{R - 14-15}$$

Přitom se předpokládá, že $\xi_1 \leq \xi_1 \leq \xi_1 \leq \ldots \leq \xi_{N_0}$; $\xi_0 = 0$. Pomocí intenzity poruch je možno vyjádři spolehlivost jako

$$R(t) = \exp\left[-\int_0^t \lambda(\tau)\,d\tau\right] \qquad \text{R - 14-16}$$

Při praktickém hodnocení spolehlivosti systému se předpokládá, že po odstranění výrobních závad se intenzita poruch ustálí na dlouhou dobu na konstantní hodnotě. V tom případě se výraz pro spolehlivost zjednoduší na tvar

$$R(t) = e^{-\lambda t} \qquad \text{R - 14-17}$$

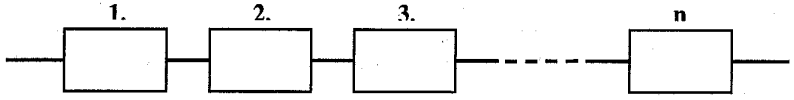
Zjednoduší se i vztah mezi střední dobou bezporuchového provozu a intenzitou poruch na

$$\lambda(t) = \frac{1}{T} \qquad \text{R - 14-18}$$

protože

$$T = \int_0^\infty R(t)\,dt = \int_0^\infty e^{-\lambda t}dt = \left[-\frac{1}{\lambda}e^{-\lambda t}\right]_0^\infty = \frac{1}{\lambda} \qquad \text{R - 14-19}$$

Při dlouhodobém sledování závad způsobených jednotlivými typy součástek získáme časovou závislost intenzity poruch, která má u elektronických zařízení typický vanovitý průběh. V první, poměrně rychle klesající části, se projevují poruchy způsobené výrobními závadami a nedostatečnou kontrolou. Pak se intenzita poruch ustálí a dlouhou dobu se nemění. Tento interval se označuje jako užitečná doba života a příslušná hodnota $\lambda = \frac{1}{t}$ charakterizuje spolehlivost sledovaných objektů. Třetí úsek, kdy se intenzita poruch opět zvětšuje je ovlivňován vnitřními změnami způsobenými stárnutím materiálu nebo mechanickým opotřebením.



Obr. 151 Spojení prvků systému z hlediska spolehlivosti

14.2 Hodnocení spolehlivosti

Pro účely hodnocení spolehlivosti si počítačový systém můžeme představit jako systém, který je sám složený z prvků charakterizovaných parametry spolehlivosti. Předpokládejme exponenciální rozložení dob mezi poruchami a vzájemnou nezávislost poruch. To znamená, že porucha jednoho prvku neovlivní pravděpodobnost vzniku poruchy v ostatních prvcích. Pokud závada jednoho prvku způsobí poruchu celého systému, můžeme předpokládat, že prvky systému jsou z hlediska spolehlivosti celého systému spojeny sériově (viz obr. 151). To platí např., když při poruše prvku dojde, opět z hlediska spolehlivosti, k jeho přerušení; taková porucha jednoho prvku způsobí skutečně poruchu celého systému. Dojde-li však k takové závadě prvku systému, že jeho špatná funkce nemusí ovlivnit správnou činnost soustavy, pak se z hlediska spolehlivosti systém

ova jako kdyby jeho byly prvky spojeny paralelně. Spolehlivost z hlediska spolehlivosti tedy
olečného se sériovostí skutečného zapojení jednotlivých elementů systému.

Pravděpodobnost, že v sériovém systému nevznikne během doby t porucha, tj. jeho spolehlivost, je na součinem spolehlivostí jeho jednotlivých prvků (dále uvedené vztahy platí pro neopravovanou soustavu)

$$R_\Sigma(t) = \prod_{i=1}^{n} R_i(t) \qquad \textbf{R - 14-20}$$

a uvedeného předpokladu nezávislosti poruch a exponenciálního rozložení dob jejich výskytu platí

$$R_\Sigma(t) = \prod_{i=1}^{n} e^{-\lambda_i t} = \exp\left[-\left(\sum_{i=1}^{n}\lambda_i\right).t\right] = e^{-\Lambda t} \qquad \textbf{R - 14-21}$$

dnotlivé intenzity poruch se sčítají a intenzita poruch celé soustavy je

$$\Lambda = \sum_{i=1}^{n}\lambda_i \qquad \textbf{R - 14-22}$$

ravděpodobnost vzniku poruchy systému v době t je

$$Q(t) = 1 - e^{-\Lambda t} \qquad \textbf{R - 14-23}$$

ůležitým faktorem, který se udává u základních součástí systému, je střední doba mezi poruchami

$$MTBF = \frac{1}{\Lambda} \qquad \textbf{R - 14-24}$$

můžeme se s ní setkat např. u mechanik pevných disků. U opravovaných systémů je nutno vycházet ze ákladních parametrů opravovaných prvků a rozlišit dva pracovní režimy:

- Při poruše libovolného prvku se systém vyřazuje z činnosti a předpokládá se, že během opravy jednoho prvku nemůže dojít k poruše žádného dalšího prvku. To znamená, že v tomto intervalu je $\lambda_i = 0$.
- Systém zůstává při poruše jednoho elementu v provozu; z toho vyplývá, že intenzita poruch se během opravy nemění. Vznikne-li několik poruch současně, při výpočtech předpokládáme, že se všechny vadné prvky začnou okamžitě opravovat.

Vztahy pro stanovení pravděpodobností, zvláště pro druhý pracovní režim, jsou poměrně složité a věnuje se jim teratura z oboru spolehlivosti.

14.3 Zálohování

V případech, kdy systém pracuje správně, pokud alespoň jeden z prvků nemá poruchu, hovoříme z hlediska polehlivosti o paralelním spojení prvků a označujeme jej jako **redundantní** nebo **zálohovaný**. Pro stanovení ákladních vztahů rovněž předpokládáme, že závady se neopravují a prvky jsou vzájemně nezávislé. Analyzujme říklad prvku zálohovaného *(n-1)* paralelně pracujícím prvkem. Pravděpodobnost poruchy je potom dána oučinem pravděpodobností $R_i(t)$ vzniku poruchy v kterémkoli z n prvků. Pravděpodobnost bezporuchové činnosti se stanoví ze vztahu

$$R_\Sigma(t) = 1 - \prod_{i=1}^{n}[1 - R_i(t)] \qquad \textbf{R - 14-25}$$

a střední doba mezi poruchami

$$MTBF = \int_0^\infty R_\Sigma(t)\,dt \qquad \textbf{R - 14-26}$$

Vyhodnocení spolehlivosti systémů, v nichž záložní prvky nepracují současně se zálohovanými, je složitější. Jedním z nejdůležitějších parametrů, který charakterizuje spolehlivost takových soustav, je však stacionární složka pohotovosti. K jejímu výpočtu lze nakreslit spolehlivostní blokové schéma a použít známá pravidla pro sériové řazení a pro zálohování součástek.

14.4 Spolehlivost programového vybavení

Podobně jako v případě technologických prvků počítače – hardware můžeme očekávat i u programového vybavení, podobného artefaktu vytvořeného člověkem, pouze jistou a konečnou spolehlivost. Budeme-li předpokládat, že podobně jako v předchozích případech může být měřítkem spolehlivosti programu statistická pravděpodobnost jeho selhání, pak za střední dobu mezi poruchami můžeme považovat dobu, která uplyne do vyskytnutí chyby při testování programu náhodnými daty. Testovaný program bude přitom zpracovávat data předávaná testovacím zařízením, jejichž pravděpodobnostní charakteristiky budou odpovídat pravděpodobnostním charakteristikám dat při praktickém provozu systému. Takové charakteristice vstupních dat obvykle říkáme **operační profil**. Dlouhodobý průměr poměrů časů běhu programu končícího chybou a celkového počtu běhů programu bude určovat **intensitu výskytu chyb,** a tedy, abychom zachovali původní terminologii, **intensitu poruch**. Té bude odpovídat i známá střední doba mezi poruchami. Protože jednotlivé intenzity poruch se sčítají, intenzita poruch celé soustavy programového a technického vybavení bude dána vztahem obdobným R - 14-22

$$\Lambda = \sum_{i=1}^{n}\lambda_i^S + \sum_{i=1}^{m}\lambda_i^H \qquad \textbf{R - 14-27}$$

kde

λ_i^S - reprezentuje poruchy programového vybavení a

λ_i^H - reprezentuje poruchy technologie.

Střední doba mezi poruchami potom bude odpovídat převrácené hodnotě intenzity poruch, jako v R - 14-24

$$MTBF = \frac{1}{\Lambda} \qquad \textbf{R - 14-28}$$

K tomu, abychom získali relevantní hodnotu střední doby mezi poruchami způsobenými programovým vybavením, bychom potřebovali, aby testovací data prověřila všechny možné posloupnosti stavů systému, což je u větších projektů prakticky nemožné. Provádí se proto statistické odhady, jejichž přesnost závisí na velikosti vzorků použitých operačních profilů a na odhadech charakteristik programu (mnohdy zcela neznámých). Tento způsob ohodnocování sebou nese některá nebezpečí, zejména

- při příliš malých vzorcích operačního profilu mohou být tyto vzorky vybrány tak, že nadměrně zvýší počet nesprávných běhů, a tak odhadnutá intensita poruch bude nepřiměřeně vysoká,
- případ, kdy při testování se vyberou vzorky tak, že při běhu programu budou záměrně obcházena slabá místa a výsledný odhad intenzity poruch bude nekorektní a velmi nízký. Tento případ je daleko nebezpečnější než předcházející.

V praxi se velmi často setkáváme s problémy při získání správného operačního profilu, které jsou často reprezentovány

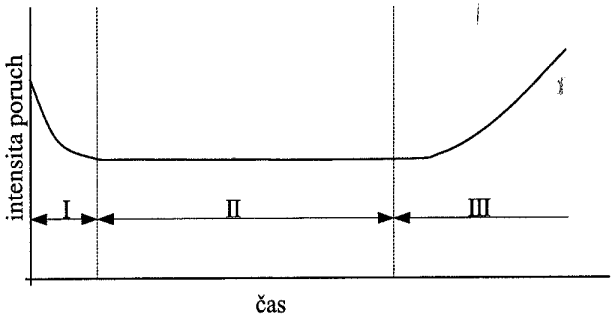
- nemožností získat správný vzorek operačních profilů nebo jejich získání je pracovně a finančně neúměrně nákladné,
- zcela různými operačními profily pro sortiment předpokládaných aplikací nebo pracovišť či klientů, které se v průběhu času mohou měnit.

Testování se špatnými operačními profily pak téměř vždy vede k optimističtějším výsledkům, což v praxi může mít dopad nejenom na vlastní provoz systému, ale zejména na jeho prodejnost a uplatnění.

Koncepce operačních profilů poskytuje uspokojivý výklad efektu časové změny intenzity poruch v programovém vybavení, který je v praxi pozorován. Tento efekt je znázorňován "vanovitou" křivkou, jak je uvedena na obr. 152. Po uvedení programu na trh je průběh intensity poruch strmě klesající v části označené I. To je obvykle vysvětlováno tím, že program je pro uživatele nový a neznámý a tak zkouší nejrůznější neortodoxní datové sekvence, které mnohdy při přípravě programu ani nebyly předvídány. Tyto operační profily, nazývané "novické operační profily" vykazují jistou míru poruch, která se rychle snižuje tak, jak autoři reagují na zjištěné nedostatky.

lientů, kteří si zvykají na nový program a sami obezřetně postupují při zadávání dat tak, aby se uchránili možnosti vzniku chyby. Tím se postupně dostávají na úroveň průměrného operačního profilu, kde je intensita hyb nejmenší, neboť tento operační profil odpovídá profilu, který autor programu očekával a který byl tudíž používán pro testování programu. Tato fáze je ukázána v obr. 152 jako plató označené II.

Třetí a konečná fáze označená III připadá na dobu, kdy se uživatel stane expertem v užívání příslušného programu, a tak zkouší nejrůznější neobvyklé posloupnosti vstupů dat ve snaze donutit program k řešení nepředpokládaných zadání a ověřit jeho mezní možnosti. Pod tlakem tohoto postupu se intenzita chyb opět výší a jiné charakteristice příčin odpovídá i pomalejší nárůst intenzity poruch.



Obr. 152 Vanovitá křivka průběhu intenzity poruch po dobu životnosti programu

Předpoklad existence operačního profilu umožňuje ještě jiný, kvantitativní pohled na spolehlivost programového vybavení. Tato teorie však neposkytuje tak uspokojující výklad vzniku "vanové křivky". Předpokládejme, že existuje proměnná, vyjadřující celkovou intensitu poruch Λ a tudíž střední doba mezi poruchami bude podobně jako v R - 14-24

$$MTBF = \frac{1}{\Lambda} \qquad \textbf{R - 14-29}$$

Budeme-li předpokládat, že program poběží bez poruch po tuto střední dobu, pak můžeme nahradit čas ve vzorci R - 14-23 počtem běhů programu a obdržíme pravděpodobnost M běhů bez poruchy[1],

$$R(M) = e^{-\Lambda M} \qquad \textbf{R - 14-30}$$

Předpokládejme, že provedeme N testů s daným operačním profilem a pokusme se zkonstruovat takovou křivku, která určí horní hranici jistoty α, že intenzita poruch Λ je menší než zvolená konstanta χ. Jestliže z N testů skončí F testů chybou (N-F testů proběhne správně) pak bude platit vztah

$$1 - \sum_{j=0}^{F} \binom{N}{j} \chi^j (1-\chi)^{N-j} \geq \alpha \qquad \textbf{R - 14-31}$$

kde N je celkový počet provedených testů,

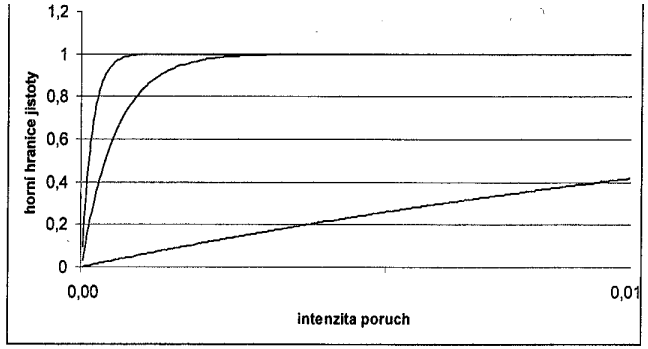
F je počet testů které skončily chybou,

α je horní hranice jistoty, že intenzita poruch nepřekročí hodnotu χ,

χ horní mez intenzity poruch.

Hodnotám parametrů F=0, N=3000 a χ=0,001 odpovídá horní hranice jistoty, že intenzita poruch nepřekročí hodnotu χ rovna 0,95. Tato hodnota se pro parametr F=0 mění podle počtu provedených testů jak je znázorněno v následujícím grafu:

Ta odpovídá spolehlivosti.



Průběhy křivek pro F=0 odpovídají vztahu

$$1 - (1-\chi)^N \qquad \textbf{R - 14-32}$$

kde pomaleji stoupající křivka odpovídá menšímu N, zatímco strmá křivka ukazuje průběh horní hranice jistoty pro velká N. Dosadíme-li mezní hodnotu χ za hodnotu intenzity poruch do vztahu R - 14-30, pak zjistíme, že spolehlivost bude 95,1% pro 50 běhů programu, 60,65% pro 500 běhů a pouze 0,67% pro 5000 běhů programu Střední doba bezporuchového provozu při téže hodnotě χ bude dosahovat 1000 běhů.

Rovnice R - 14-31 může sloužit pro predikci intenzity poruch programového vybavení i v případě, který se v praxi často vyskytuje, že program je dán do provozu po absolvování řady testů, které všechny skončily bezchybně. Na této rovnici je zajímavé, že není závislá na charakteristikách testovaného programu přesto, že intuitivně lze očekávat nižší horní hranici jistoty α pro danou intensitu poruch Λ v případě složitějšího programu. Jiný přístup k odvození vztahů mezi horní mezí jistoty, velikostí testovacího vzorku a intenzitou poruch může vycházet z pojímání testů jako experimentů, které mají prokázat hypotézu, je intenzita poruch bude nižší než zadaná hodnota. V literatuře [B1] se vyskytuje celá řada podobných modelů, jejichž závěry vedou k rovnici podobné R - 14-31. Závěry, které z těchto studií vyplývají, svědčí o obtížné realizovatelnosti ultraspolehlivých programových systémů, resp. testování jako o velmi nepraktickém způsobu získání informací o spolehlivosti programového systému.

Poznámka

Jak vyplývá z rovnice R - 14-31, k predikci střední doby mezi poruchami rovné M běhům programu budeme pro 90% jistotu potřebovat 2M testovacích běhů Předpokládáme-li např. požadavek na intenzitu poruch menší než 10^{-8} pak při době běhu jednoho testu zhruba jednu sekundu budeme potřebovat k testování téměř tři roky. K tomu je nutno si uvědomit, že testovací proces je nutno začít vždy od začátku po opravě chyby, která byla příčinou poruchy.

S ohledem na výše uvedené pesimistické závěry, ke kterým vede teoretická analýza zjišťování spolehlivosti programového vybavení, je zřejmé, že použití těchto metod je nutno omezit pouze na kritické aplikace nebo jen kritické části aplikací. Jinými slovy, s ohledem na rozsáhlou klientelu používající programové vybavení pro personální počítače lze téměř s jistotou očekávat, že příslušné programové vybavení se dostane do chybového stavu ještě ten den, kdy bylo uvolněno pro distribuci. Aby k tomu nedošlo, muselo by být příslušné programové vybavení testováno na ultraspolehlivost, což by vedlo k nárůstu ceny a prodloužení doby uvedení na trh[1] – tedy k jeho neprodejnosti.

[1] Tzv. "time to market" je velmi sledovaným parametrem programového vybavení pro personální počítače.

15 Přílohy

nehledate.
A. Bloch: "Murphy's Law"

Číslo	Obsah	Strana
15.1	Historický přehled významných typů počítačů	226
15.2	Kódovací tabulka EBCDIC	228
15.3	Kódovací tabulka ASCII	229
15.4	Kódovací tabulka Latin 2	230
15.5	Seznam nejdůležitějších anglických zkratek z výpočetní techniky	232

15.1 Historický přehled významných typů počítačů do roku 1978

Rok	USA	Česká republika	Ostatní
1938			Z1 (Zuse, Německo)
1939			Z2 (Zuse, Německo)
1940			
1941			Z3 (Zuse, Německo)
1942			
1943	ASCC (Harvard Mark I)		
1944			
1945			Z4 (Zuse, Německo)
1946	ENIAC		
1947	SSEC (IBM)		
1948	IBM604		
1949			Manchester MARK I
			EDSAC (Anglie)
1950			
1951	UNIVAC I (EMCC)		MESM (SSSR)
1952	IBM701		BESM1 (SSSR)
1953			STRELA (SSSR)
			Z5 (Zuse, Německo)
1954	B205 (Borrough)		
	IBM650		
1955			MINSK1 (SSSR)
1956	IBM704		
1957		SAPO	URAL1 (SSSR)
			Z11 (Zuse, NSR)
1958			Z22 (Zuse, NSR)
1959			
1960	IBM7030 – Strech		MINSK2 (SSSR)
	CDC3000		Z31 (Z31, NSR)
1961			
1962	ATLAS		
1963		EPOS1	
		MSP	
1964	IBM360	EPOS2	
1965		ZPA601	
1966			
1967	ILLIAC IV		BESM6 (SSSR)
1968	B500		
	CDC7600		
1969			
1970		EC1021	
1971			URAL16
			ICL4/72
1972	STARAN	EC1025	
	IBM370	Tesla 200	
1973	B700		
1974		ADT4000	
1975		EC1027	ICL2900
1976			
1977	CRAY1		
1978	SYSTEM38 (IBM)	EC1025	

5.2 Kódovací tabulka EBCDIC

bity 4 - 7 →

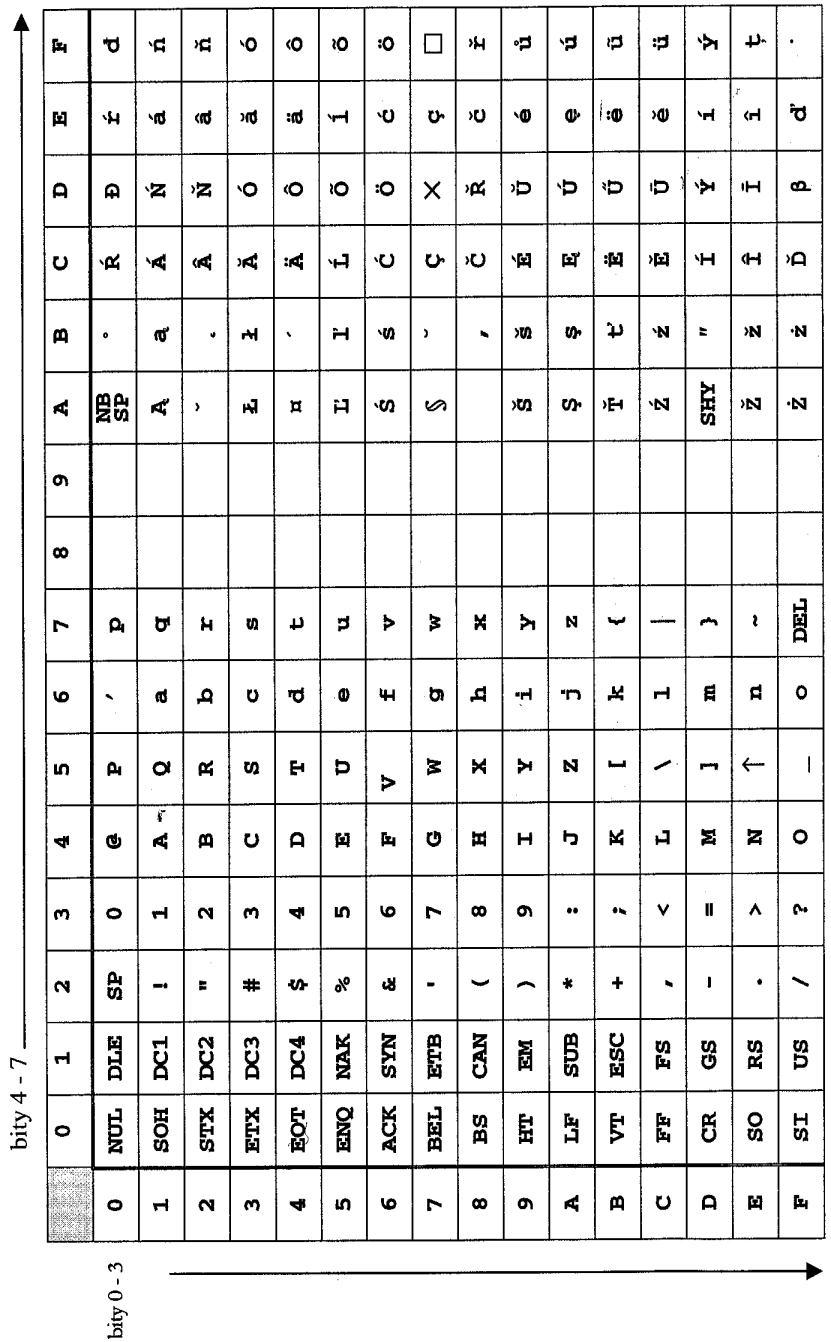
bity 0 - 3 ↓	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&	-						{	}	\	
1	SOH	DC1	SOS				/		a	j	~		A	J		
2	STX	DC2	FS	SYN					b	k	s		B	K	S	
3	ETX	TM							c	l	t		C	L	T	
4	PF	RES	BYP	PN					d	m	u		D	M	U	
5	HT	NL	LF	RS					e	n	v		E	N	V	
6	LC	BS	ETB	UC					f	o	w		F	O	W	
7	DEL	IL	ESC	EOT					g	p	x		G	P	X	
8	GE	CAN							h	q	y		H	Q	Y	
9	RLF	EM						`	i	r	z		I	R	Z	
A	SMM	CC	SM		¯	!	|	:							
B	VT	CUL	CU2	CU3	.	$	,	#							—	
C	FF	IFS		DC4	>	*	%	@								
D	CR	IGS	ENQ	NAK	(	)		'						≠		
E	SO	IRS	ACK		+	;	<	=							¬	
F	SI	IUS	BEL	SUB	|		?	"								

15.3 Kódovací tabulka ASCII

bity 4 - 6 →

bity 0 - 3 ↓	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	↑	n	~
F	SI	US	/	?	O	_	o	DEL

15.4 Kódovací tabulka Latin 2



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SP	0	@	P	`	p			NB SP	°	Ŕ	Đ	ŕ	đ
1	SOH	DC1	!	1	A	Q	a	q			Ą	ą	Á	Ń	á	ń
2	STX	DC2	"	2	B	R	b	r			˘	Ł	Â	Ň	â	ň
3	ETX	DC3	#	3	C	S	c	s			Ł	ł	Ă	Ó	ă	ó
4	EOT	DC4	$	4	D	T	d	t			¤	´	Ä	Ô	ä	ô
5	ENQ	NAK	%	5	E	U	e	u			Ľ	Ľ	Ĺ	Ő	ĺ	ő
6	ACK	SYN	&	6	F	V	f	v			Ś	ś	Ć	Ö	ć	ö
7	BEL	ETB	'	7	G	W	g	w			§	ˇ	Ç	×	ç	□
8	BS	CAN	(	8	H	X	h	x				¸	Č	Ř	č	ř
9	HT	EM	)	9	I	Y	i	y			Š	ą	É	Ů	é	ů
A	LF	SUB	*	:	J	Z	j	z			Ş	ş	Ę	Ú	ę	ú
B	VT	ESC	+	;	K	[	k	{			Ť	ť	Ë	Ű	ë	ű
C	FF	FS	,	<	L	\	l	|		Ź	Ź	Ě	Ü	ě	ü
D	CR	GS	-	=	M	]	m	}			SHY	˝	Í	Ý	í	ý
E	SO	RS	.	>	N	←	n	~			Ž	ž	Î	Ţ	î	ţ
F	SI	US	/	?	O	_	o	DEL			Ż	ż	Ď	ß	ď	˙

Použité zkratky

Zkratka	Význam	Zkratka	Význam
ACK	Ackowledge	IGS	Interchange Group Separator
BEL	Bell	IL	Idle
BS	Backspace	IRS	Interchange Record Separator
BYP	Bypass	IUS	Interchange Unit Separator
CAN	Cancel	LC	Lower Case
CC	Cursor Control	LF	Line Feed
CR	Carriage Return	NAK	Negative Acknowledge
CU1	Customer Use 1	NL	New Line
CU2	Customer Use 2	NUL	Null
CU3	Customer Use 3	PF	Punch Off
DC1	Device Control 1	PN	Punch On
DC2	Device Control 2	RES	Restore
DC4	Device Control 4	RS	Reader Stop
DEL	Delete	SI	Shift In
DLE	Data Link Escape	SM	Set Mode
DS	Digit Select	SMM	Start of Manual Message
EM	End of Medium	SO	Shift Out
ENQ	Enquiry	SOH	Start of Heading
EOT	End of Transmission	SOS	Start of Significance
ESC	Escape	SP	Space
ETB	End of Block	STX	Start of Text
ETX	End of Text	SUB	Substitute
FF	Form Feed	SYN	Synchronous Idle
FS	Field Separator	TM	Tape Mark
HT	Horizontal Tab	UC	Upper Case
IFS	Interchange File Separator	VT	Vertical Tab

techniky

ACM (Association for Computing Machinery) - Asociace pro výpočty pomocí strojů (USA)

AI (Artificial Intelligence) - umělá inteligence

ALGOL (Algorithmic Language) - programovacíjazyk pro vědecko-technické výpočty

ALU (Arithmetic-Logic Unit) - aritmeticko-logická jednotka

AMD (Advanced Micro Devices) - americký výrobce integrovaných obvodů

ANSI (American National Standards Institute) - Americký národní ústav pro normalizaci

ARPA (Advanced Research Project Agency) - americká vládní výzkumná agentura, která vedla projekt, při kterém vznikla jedna z prvních sítí - ARPANET

ASCII (American Standard Code for Information Interchange) - standardní kód pro přenos informací

BASIC (Beginner's All-purpose Symbolic Instruction Code) - symbolický programovací jazyk pro začátečníky

BCD (Binary-Coded Decimal) - kód desítkových číslic

BIOS (Basic Input-Output System) - základní systém vstupů a výstupů, nejnižší programová úroveň systému

bit (binary digit) - dvojková číslice

BIU (Bus Interface Unit) - jednotka styku se sběrnicí

bpi (bits per inch) - bitů na palec

CAD (Computer-Aided Design) - programy pro podporu projektováni pomocí počítače

CAE (Computer-Aided Engineering) - programy pro podporu vývojových prací počítačem

CAM (Computer-Aided Manufacturing) - podpora výroby pomocí počítačů

CAM (Content-Addressable Memory) - asociativní poměď

CCD (Charge-Coupled Device) - nábojově vázaný obvod, integrované obvody používané např. ve snímačích obrazu

CD (Compact Disc) - kompaktní disk, záznamové médium

CDC (Control Data Company) - americký výrobce počítačů

CD-ROM (Compact Disc Read-Only Memory) - pevná vnější paměť realizovaná na kompaktním disku

CGA (Color Graphics Adapter) - adaptér - karta do PC pro řízení barevného monitoru

CISC (Complex Instruction Set Computer) - počítač s úplným souborem instrukcí

COBOL (Common Business-Oriented Language) - programovací jazyk pro ekonomické aplikace

CP/M (Control Program for Microcomputers) - operační systém používaný v mikropočítačích

CPU (Central Processing Unit) - centrální procesor

CRC (Cyclic Redundancy Check) - zabezpečovací data tvořená cyklickým kódem

DARPA (Defense Advanced Research Project Agency) - americká vládní výzkumná agentura, dříve ARPA

DEC (Digital Equipment Corporation) - americký výrobce počítačů

DMA (Direct Memory Access) - metoda přímého přístupu do paměti

DMX (Demultiplexer) - demultiplexor

DRAM (Dynamic RAM) - dynamická paměť s libovolným výběrem

EARN (European Academic Research Network) - síť propojující evropské univerzity a výzkumné laboratoře na bázi techniky BITNET

EAROM (Electrically Alterable Read-Only Memory) - elektricky zapisovatelná semipermanentní paměť

EBCDIC (Extended Binary-Coded Decimal Interchange Code) - rozšířený dvojkový kód desítkových číslic pro výměnu informací

ECC (Error Check and Correction) - kontrola a oprava chyb

ECL (Emitter-Coupled Logic) - bipolární technologie integrovaných obvodů

ECMA (European Computer Manufacturers Association) - evropská asociace výrobců počítačů

EDP (Electronic Data Processing) - elektronické zpracování dat

EEPROM (Electrically Erasable Programmable Read-Only Memory) - elektricky mazatelná semipermanentni paměť

EGA (Enhanced Graphics Adapter) - karta pro řízení barevného monitoru

EISA (Extended Industry Standard Architecture) - sběrnice počítačů PC sestavených na bázi procesorů 80836 a 80846

ENIAC (Electronic Numerical Integrator and Calculator) - první elektronkový počítač

EPROM (Erasable Programmable Read-Only Memory) - mazatelná semipermanentní paměť

ESC (Escape) - zvláštní znak používaný k odlišení přicházejícího kódu u tiskáren nebo při sériovém přenosu

EU (Execution Unit) - jednotka provádějící instrukce v procesoru

FDDI (Fiber Distributed Data Interface) - protokol používaný pro spojení optickými vlákny

FIFO (First In - First Out) - postupná paměť fronty

FLOPS (Floating-Point Operations per Second) - počet operací s pohyblivou řádovou čárkou za sekundu

FORTRAN (Formula Translator) - programovací jazyk pro vědeckotechnické výpočty

FPGA (Field-Programmable Gate Array) - integrovaný obvod - hradlové pole programovatelné u zákazníka

FPP (Floating-Point Processor) - procesor pro operace s pohyblivou řádovou čárkou

FPU (FIoating-Point Unit) - jednotka pro operace s pohyblivou řádovou čárkou

GB (Gigabyte) - 1 073 741 824 bytů (cca miliarda bytů)

GFLOPS (Giga Floating-point Operations Per Second) - 1G operací s pohyblivou řádovou čárkou za sekundu

GIPS (Giga Instructions per Second) - miliarda operací za sekundu

GPIB (GeneraI-Purpose Interface Bus) - univerzální sběrnice

HGC (Hercules Graphics Card) - karta pro řízení monochromatického monitoru v PC

HLDA (Hold Acknowledge) - potvrzení stavu HOLD

HP (Hewlett-Packard) - americký výrobce počítačů

HPIB (Hewlett-Packard Interface Bus) - sběrnice používaná firmou Hewlett-Packard

I/O (Input/Output) - vstup a výstup

IBM (International Business Machines) - výrobce počítačů

IC (Integrated Circuit) - integrovaný obvod

IEC (International Electronic Comission) - Mezinárodní komise pro elektroniku

IEE (Institution of Electrical Engineers) - Svaz elektrotechnických inženýrů (Británie)

IEEE (The Institute of Electrical and Electronics Engineers) - Svaz elektrotechnických a elektronických inženýrů (USA)

IFAC (International Federation of Automatic Control) - Mezinárodní federace pro automatické řízeni

IFIP (International Federation for Information Processing) - Mezinárodní federace pro zpracování informací

IIL, I2L (Integrated Injection Logic) - bipolární technologie používající injekční logiku

INT (Interrupt) - přerušení

INTA (Interrupt Acknowledge) - potvrzení přijetí přerušení

INTE (Interrupt Enable) - přerušení povoleno

IOP (Input/Output Processor) - procesor pro obsluhu vstup/výstupu

IR (Instruction Register) - registr instrukce

ISA (Industry Standard Architecture) - sběrnice počítače IBM PC/AT

ISO (International Standards Organization) - Mezinárodní organizace pro normalizaci

LAN (Local Area Network) - místní počítačová síť

LCD (Liquid Crystal Display) - zobrazovací jednotka z kapalných krystalů

LIFO (Last In - First Out) - zásobníková paměť

LQ (Letter Quality) - označení vysoké kvality tisku mozaikových tiskáren

LRU (Least Recently Used) - nejméně v poslední době používaný

LSB (Least Significant Bit) - bit s nejnižší vahou

LSI (Large-Scale Integration) - integrované obvody velké integrace

MAN (Metropolitan Area Network) - městská počítačová síť

MB (Megabyte) - 1 048 576 B (cca milion slabik)

MCA (Micro Channel Architecture) - sběrnice vyšších modelů řady personálních počítačů PS/2

MFLOPS (Mega Floating-point Operations Per Second) - milión operací s pohyblivou řádovou čárkou za sekundu

MIMD (Multiple Instruction stream - Multiple Data stream) - architektura procesoru, ve které je současně aplikováno více instrukcí na více datových položek

MIPS (Mega Instructions per Second) - milión instrukci za sekundu

MISD (Multiple Instruction stream - Single Data stream) - hypotetická architektura, ve které je současně aplikováno více instrukcí na jednu datovou položku

MLIPS (Mega Logical Inferences Per Second) - milión logických referencí za sekundu

MODEM (Modulator-Demodulator) - kombinace modulátoru a demodulátoru, součást jednotka pro přenos dat

MPU (Microprocessor Unit) - mikroprocesor

MS DOS (Microsoft Disk Operating System) - diskový operační systém firmy Microsoft

MSB (Most Significant Bit) - bit s nejvyšší vahou

MSI (Medium-Scale Integration) - integrované obvody střední integrace

MSIMD (Multiple Single Instruction stream - Multiple Data stream) - systém obsahující několik nezávislých počítačů typu SIMD

MTBF (Mean Time Between Failures) - střední doba mezi poruchami

MTTF (Mean Time To Failure) - střední doba do poruchy

MTTR (Mean Time To Repair) - střední doba do opravy

NCR (National Cash Registers) - americký výrobce počítačů

NEC (Nippon Electric Company) - japonský výrobce počítačů

NLQ (Near Letter Quality) - označení vyšší kvality tisku mozaikových tiskáren vhodné pro dopisové dokumenty

NMI (Non-Maskable Interrupt) - nemaskovatelné přerušení

OS (Operating System) - operační systém

OSF (Open Systems Foundation) - konsorcium firem usilující o normalizaci operačního systému UNIX

OSI (Open Systems Interconnection) - mezinárodní program normalizace propojení počítačů různých výrobců

PC (Personal Computer) - osobní počítač

PC (Program Counter) - čítač instrukcí

PCL (Printer Control Language) - jazyk pro řízení tiskáren

PIA (Peripheral Interface Adapter) - obvod pro připojení periferních zařízení

PIO (Programmable Input-Output) - programovatelný vstupní a výstupní obvod

PL/1 (Programming Language One) - programovací jazyk pro vědeckotechnické výpočty

PL/M (Programming Language for Microprocessors) - programovací jazyk pro mikroprocesory

PLA (Programmable Logic Array) - programovatelné logické pole

PLD (Programmable Logic Device) - programovatelný logický obvod

PROM (Programmable Read-Only Memory) - programovatelná permanentní paměť

R/W (Read/Write) - čtení nebo zápis

RAID (Redundant Array of Inexpensive Disks) -redundantní pole levných disků

RAM (Random-Access Memory) - paměť s libovolným výběrem

REPROM (Reprogrammable Read-Only Memory) - opětovně programovatelná semipermanentní paměť

RISC (Reduced Instruction Set Computer) - počítač s redukovaným souborem instrukcí

ROM (Read-Only Memory) - permanentní paměť

RWM (Read-Write Memory) - paměť s možností čtení i zápisu

SBC (Single-Board Computer) - jednodeskový počítač

SCSI (Small Computer Systems Interface) - standardizované rozhraní pro připojení periferií

SIMD (Single Instruction stream - Multiple Data stream) - architektura, ve které je jedna instrukce aplikována na více datových položek

SIMM (Single In-line Memory Module) - integrovaný modul s jednou řadou vývodů

SIO (Serial Input/Output) - sériový vstup/výstupní

SISD (Single Instruction stream - Single Data stream) -procesor von Neumannova typu

SP (Stack Pointer) - ukazatel zásobníku

SPARC (Scalable Processor ARChitecture) - modulární architektura procesorů

SQL (Structured Query Language) - jazyk pro přístup k relačním bázím dat

SPMD (Same Program - Multiple Data) - počítač, v němž stejný program probíhá asynchronně v různých procesorech

SRAM (Static RAM) - statická paměť s libovolným výběrem

TB (Terabyte) - 1 099 511 627 776 B (cca bilión slabik)

TCP/IP (Transmission Control Protocol / Internet Protocol) - protokol přenosu dat v počítačové síti

TFLOPS (Tera FLoating-point Operations Per Second) - 10^{12} operací s pohyblivou řádovou čárkou za sekundu

TMR (Triple Modular Redundancy) - redundance 2 ze 3 modulů

TTY (Teletype) - dálnopis

UART (Universal Asynchronous Receiver-Transmitter) - univerzální asynchronní přijímač a vysílač

USART (Universal Synchronous - Asynchronous Receiver - Transmitter) - univerzální synchronní a asynchronní přijímač a vysílač

VGA (Video Graphics Array) - karta pro řízení barevného monitoru

VMA (Valid Memory Address) - platná adresa paměti

WAN (Wide Area Network) - počítačová síť, která pokrývá rozsáhlé území

reference.
A. Bloch: "Murphy's Law

16 Literatura

[A1] Gorsline G. H.:*"Computer Organization - Hardware/Software"*, Prentice Hall International, London 1986

[A2] Grohoski G. F.:*"Machine organization of the IBM RISC System/6000 Processor"*, IBM J. Res. Develop. 43(1), January 1990, pp. 37-58

[A3] Flynn M.:*"Some Computer Organizations and Their Effectivness"*, IEEE Trans. Comput., Vol. C-21, 1972, p.94

[A4] Hwang K.:*"Advanced Parallel Processing with Supercomputer Architecture"*, Proc. IEEE, Vol. 75, 1987, pp. 1348 - 1379

[A5] Bell G.:*"The Future of High Performance Computers in Science and Engineering"*, Comm of ACM, Vol. 32, 1989, pp. 1091 - 1101

[A6] Hockney R., Jesshope C.:*"Parallel Computers 2"*, Adam Hilger Ltd., Bristol (UK) 1988

[A7] Seitz L.:*"The Cosmic Cube"*, Comm. ACM 28(1), Jan 1985, pp.25 - 32

[A8] Marovac N.:*"A Systematic Approach to the Design and Implementation of Computer Instruction Set"*, Computer Architecture News 11(1), 1983, pp. 19-24

[A9] Fisher W. D.:*"Microprocessor Assembly Language Draft Standard"*, Computer 12 (12), 1979, pp. 96-109

[B1] Belady L. A.:*"A study of replacement algorithms for a virtual-storage computer"*, IBM Systems Journal Volume 5, Number 2, 1966, pp. 78 – 101

[B2] Hamlet D.:*"Random testing"*, in *Encyclopedia of Software Engineering*, Wiley, New York, 1994, pp. 970-978

[B3] I.B.M Inc.:*"Power PC Microprocessor Common Hardware Reference Platform: A System Architecture"*,Morgan Kaufmann Publishers, New York 1996

[B4] *"Pentium II Processor Developers Manual"*, at www.intel.com

[B5] *"P6 Family of Processors Hardware Developers Manual"*, at www.intel.com

[H1] Hlavička J.:*"Architektura počítačů"*, Vydavatelství ČVUT, Praha 1996

[H2] *"SPARC Architecture Manual (Version 8)"*, at www.sun.com

[H5] Bochner L., Salomon K.:*"John von Neumann"* in *Biographical Memoirs*, Vol.32, 1958, National Academy of Science, pp.451-456

[L1] Peterson D.:*"Reduced Instruction Set Computers"*, Communication of the ACM, Vol.28, No.1, January 1985, pp. 9-21

[L2] Kai Hwang, Briggs F.A.:*"Computer Architecture and Parallel Processing"*, McGraw Hill, New York 1985

[L3] Siewiorek D. P., Bell G., Newell A.:*"Computer Structures: Principles and Examples"*, McGraw Hill, Auckland 1985

[M1] Shor P.:*"Algorithms for Quantum Computations: Discrete Log and Factoring"*, Proc. 35th Ann. Symp. Foundations of Computer Science, 1994

[M2] Cirac J. I., Zoller P: in *Physical Revue Letters* 74, p.4091, 1995

[M3] Lent C. S. et al.:*"Quantum Cellular Automata: The Physics of Computing with Arrays of quantum Dot Molecules"*, in *Proc. of the Workshop on Physics and Computation*, PhysComp94, 1994, pp. 5-13

[M4] Korotkov A. N.:*"Wireless Single-Electron Logic Biased by Alternating Electric Field"*, Appl. Phys. Lett., Vol. 67, 1995, pp. 2412-2414

[M5] Weiss R., Homsy G., Knight T.:*"Toward in vivo Digital Circuits"*, in *Proceedings of the Dimacs Workshop on Evolution as Computation*, January 1999

[M6] Roessler O.:*"Lecture Notes in Biomathematics"*, Springer, Berlin 1974

[M7] Seabaugh A. C., Bascombe J. H., Randall J. N. *Quantum Functional Devices: Prospects*", Future Electron Devices (FED) Journal, Vol. 3, Suppl. 1, 1993, pp. 9-20

[M8] Kastner M. A.:*"The Single-Electron Transistor"*, Review of Modern Physics, Vol. 64, 1992, pp. 849-858

[M9] Bandyopadhyay S., Das B., Miller A. E.:*"Supercomputing with Spin-Polarized Single Electrons in a Quantum Coupled Architecture"*, Nanotechnology, Vol.5, 1994, pp. 113-133

[N1] Feynman R. P.:*"There's Plenty of Room at the Bottom: An Invitation to Enter a New Field of Physics"*, Engineering an Science, Vol. 23, 1960, pp.22-36

[N2] Rabaey J.M.:*"Digital Integrated Circuits"*, Prentice Hall, New Jersey 1996

[N3] Drexler K.E.:*"Engines of Creation"*, Anchor Press, Garden City New York, 1986

[N4] Adleman L.:*"Molecular Computation of Solutions to Combinatorial Problems"*, Science, Vol. 266, November 11, 1994, pp.1021-1023

[N5] Kolata G.:*"A Vat of DNA May Become a Vast Computer of the Future"*, N.Y. Times, April 11, 1995. p. C1

[N6] Lipton R.J.:*"Using DNA to solve NP-complete problems"*, Science, Vol. 268, April 28 1995, pp.542-545

[N7] Birge R.:*"Protein-Based Three Dimensional Memory"*, American Scientist, July-August 1994, pp. 348-355

[N8] Benioff P.:*"The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines"*, J. Statistical Physics, Vol. 22, 1980, pp. 563-591

[N9] Deutsch D.:*"Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer"*, Proc. Roy. Soc. Lond., Vol. A400, 1985, pp. 96-117

[P1] Plášil F.:*"Operační systémy"*, Elektrotechnická fakulta ČVUT Praha, Praha 1989

[R1] Wada Y. et al.:*"A Proposal of Nanoscale Devices Based on Atom Molecule Switching"*, J. Appl.. Phys., 15 December 1993, pp. 7321-7328

[R2] Lent C. S. et al.:*"Quantum Cellular Automata,"*, Nanotechnology, Vol. 4, 1993, p. 49

[R3] Behrman E. C. et al.:*"A Quantum-Dot Neural Network"*, Proceedings for the PhysComp96 Conference, November 22-24, Boston (MA) 1996

[R4] Šnorek M., Jiřina M.:*"Neuronové sítě a neuropočítače"*, ČVUT Praha, 1996

[S1] Scott N.R.:*"Computer Number Systems and Arithmetic"*, Prentice Hall, New York 1985

[X1] Blatný J., Krištoufek K., Pokorný Z.: *"Číslicové počítače"*, SNTL Praha 1980

[X2] Krištoufek K. a kol.:*"Encyklopedie výpočetní a řídící technika"*, SNTL Praha 1982

[X3] Bywater R. E.H.:*"Hardware/Software design of digital systems"*, Prentice Hall International, London 1981

[X4] Drábek: V.:*"Výstavba počítačů"*, VUT Brno – PC-DIR s.r.o, Brno 1995

[X5] Babbage Ch.:*"A Letter to Sir Humphry Davy, Bart. PRS, on the Application of Machinery to the Purpose of Calculating and Printing Mathematical Tables"*, J. Booth, London, 1822

[X6] Babbage Ch.:*"On the Economy of Machinery and Manufactures"*, Charles Knight, London 1832

[X7] Babbage Ch.:*"Table of the Relative Frequency of Occurrence of the Causes of Breaking of Plate Glass Windows"*, Mechanics Mag. **66,** p. 82, London 1857

[X8] Babbage Ch.:*"Passages on the Life of a Philosopher"*, Longman-Green, London 1864

[X9] Babbage Ch.:*"The Exposition of 1851; or Views of the Industry, the Science, and the Government of England"*, John Murray, London 1851, 2nd ed.

Ústřední knihovna

KOMERČNÍ POJIŠŤOVNA a.s.

Tuto publikaci vydala MFF UK v rámci spolupráce s Komerční pojišťovnou, členem Finanční skupiny Komerční banky.

Komerční pojišťovna, založená na progresívních pojistně technických principech, cíleně podporuje rozvoj exaktních věd. Řada absolventů MFF UK už také našla v pojišťovně své pracovní uplatnění. Výsledkem této spolupráce je důsledná a rychlá aplikace nejnovějších poznatků pojistně matematické teorie při tvorbě všech druhů pojištění.

Komerční pojišťovna nabízí širokou škálu produktů – riziková životní pojištění k úvěrům KB, kapitálová životní pojištění s podíly na zisku, důchodové pojištění, úrazové pojištění dětí a dospělých, úrazové pojištění na návratný vklad, kolektivní pojištění, sdružené pojištění dětí, pojištění průmyslu a podnikatelů, cestovní pojištění, pojištění majetku a odpovědnosti občanů.

Čtenáře této publikace, tedy především studenty, si dovolujeme upozornit na pojistné produkty zvláště vhodné pro jejich věk:

- **úrazové pojištění**
- **kapitálové životní pojištění s podíly na zisku a s možností výběru prémie, která přispěje k financování dalšího studia**
- **cestovní pojištění**

Podrobné informace o všech produktech Komerční pojišťovny obdržíte ve všech pobočkách KB a na centrále Komerční pojišťovny.

Komerční pojišťovna, a.s.
centrála
Jindřišská 17
P.O.Box 353
111 21 Praha 1

Tel.: (02) 220 951 11
(02) 242 301 06
Fax: (02) 242 366 96

Buďte si jisti

Ing. Václav Jirovský, CSc.

PRINCIPY POČÍTAČŮ

Vydal
MATFYZPRESS
vydavatelství
Matematicko-fyzikální fakulty
Univerzity Karlovy
Ke Karlovu 3
121 16 Praha 2
jako svou 53. publikaci

Recenzenti:
Prof. ing. Jiří Bíla, DrSc., Mgr. Jakub Yaghob

Z připravených předloh v systému MS-Word 97
vytisklo Reprostředisko MFF UK
Malostranské nám. 25, 118 00 Praha 1

Vydání první

Praha 2000

ISBN 80-85863-51-0