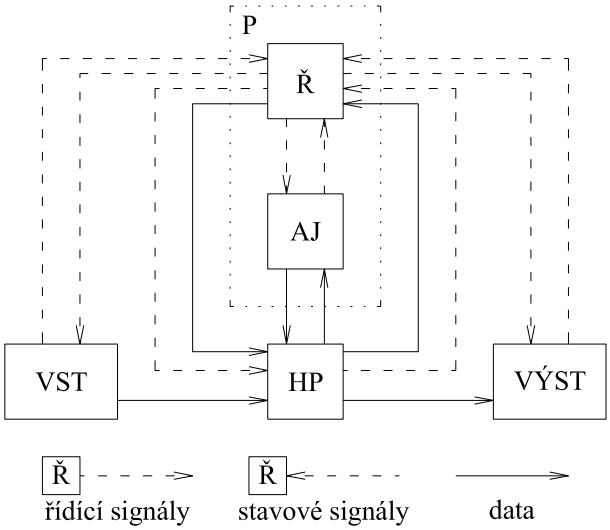
Kapitola 5

Architektura počítačů

Na otázku „*Co je počítač?*“ můžeme odpovědět, že je to elektronické zařízení, které zpracovává data (transformuje vstupní data na výstupní). Data (texty, obrázky) mohou být reprezentována analogově nebo číslicově. První počítače byly analogové, od konce druhé světové války existují i počítače číslicové. Již padesát let je základem většiny konstruovaných počítačů architektura, kterou navrhl v roce 1947 americký matematik maďarského původu *Johann von Neumann*, - tzv. Von Neumannova architektura.[1]

5.1 Von Neumannova architektura

Blokové schéma počítače s touto architekturou je na obr. 5.1. Počítač se (podle Neumanna) skládá z těchto částí: **řadič Ř, aritmeticko-logická jednotka AJ, hlavní paměť HP, vstupní** a **výstupní jednotky**.



Obrázek 5.1: Von Neumannova architektura počítače

Řadič počítače spolu s aritmetickou jednotkou tvoří *procesor*. Vstupní a výstupní jednotky (klávesnice, disketová jednotka, monitor, tiskárna) slouží pro vstup a výstup dat. Nazývají se také *periférie, periferní zařízení*. Charakteristiké znaky architektury je možné shrnout do několika bodů:

1. Architektura je nezávislá na zpracovávané úloze, činnost je řízena obsahem paměti - programem (tokem instrukcí).

2. Paměť je společná pro program i zpracovávaná data; data ani program nejsou nijak odděleny ani explicitně označeny.

[1] Architekturou počítače rozumíme, z jakých bloků se počítač skládá a jak jsou mezi sebou tyto bloky propojeny.

3. Paměť je rozdělena na buňky s lineární organizací; k obsahu buňky se přistupuje pomocí jejího pořadového čísla (adresy).

4. Pro reprezentaci instrukcí, adres, dat i řízení se používají dvojkové signály (dvojková soustava).

5. Instrukce se vykonávají v pořadí, jak jsou zapsány v paměti (zpravidla od nižších k vyšším adresám); pořadí lze změnit speciální instrukcí skoku.

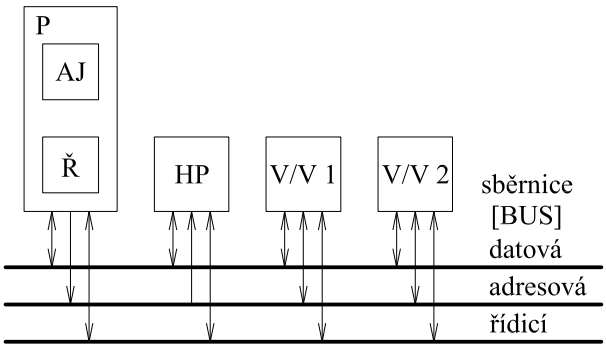
6. V instrukci není zpravidla uvedend operand (data, která se zpracovávájí), ale jeho adresa.

Většina vlastností architektury byla v době vzniku revoluční. Např. rys, že architektura je stálá a činnost počítače je řízena *programem*, což je posloupnost *instrukcí* (jednoduchých příkazů). Tím pádem není závislá na zpracovávané úloze. Analogové počítače a první číslicový počítač se totiž programovaly zapojením podle úlohy, kterou bylo třeba řešit. Vykonávání instrukcí v pořadí, v jakém jsou zapsány v paměti, zmenšuje prostor v paměti potřebný k uložení instrukcí. Některé první počítače měly v instrukci zapsánu adresu, ze které se má načíst následující instrukce, což se ukázalo jako zbytečné. Poslední vlastnost má za následek, že pokud chceme zpracovávat jiná data, není potřeba měnit program.

Načítání instrukcí z paměti, jejich vykonávání a řízení ostatních bloků zajišťuje *řadič* (angl. *controller*). Informace z ostatních jednotek přenášejí do řadiče *stavové signály*, řadič vysílá *řídicí signály*. V širším slova smyslu označujeme pojmem řadič každý obvod, který na základě vstupních stavových signálů generuje řídicí signály (řadič disku). *Aritmeticko-logická jednotka* (angl. *Arithmetical and Logical Unit*) vykonává aritmetické a logické operace nad daty na základě řízení z řadiče.

V *hlavní paměti* (angl. *main memory*), nazývané též *operační paměť*, jsou uloženy program a zpracovávaná data. Pro jejich reprezentaci se používá dvojková soustava a dvojkové signály (i pro řízení). Data ani program nejsou v paměti nijak odděleny ani označeny, explicitně nejsou označeny ani jednotlivé datové typy (podíváme-li se na obsah určitého místa v paměti, nepoznáme, zda jde o část programu nebo o data, ev. zda data reprezentují číslo, text, obrázek aj.). Je to jednoduché řešení, později se ukázalo, že je někdy výhodné paměť programu a dat oddělit (tzv. *harwardská architektura*).

Propojení jednotek na obr. 5.1 je zakresleno *dvoubodovými spoji*. Takto je možné jednotky v reálu samozřejmě propojit, většinou se ale propojuje pomocí *sběrnic*, obr. 5.2.



Obrázek 5.2: Propojení jednotek pomocí sběrnic

Sběrnice (angl. Bus) je *vícebodový spoj*, tj. fyzicky propojuje více než dvě zařízení. Je zřejmé, že vysílat (ovládat) vodiče může v určitém okamžiku jen jedno zařízení, ostatní musí být v režimu příjmu nebo být zcela odpojeny od sběrnice. Podmínkou připojení obvodů na sběrnici je, aby měly výstupy řešeny jako třístavové. Sběrnice v počítači jsou obvykle tři: *datová, adresová*

a *řídicí*. Datová slouží k přenosu dat a bývá obousměrná, tj. data mohou být přenášena z procesoru/do procesoru, z paměti/do paměti, z disku/na disk atd. Podle šířky sběrnice (počtu bitů, vodičů) hovoříme o osmibitové, šestnáctibitové aj. sběrnici. Na adresové sběrnici je platná adresa do paměti při čtení instrukce nebo čtení/zápisu dat do paměti, resp. adresa periférie při vstupně/výstupní operaci. Bývá jednosměrná, adresu generuje procesor, ve speciálních případech tzv. přímého přístupu do paměti (DMA - Direct Memory Access) ji generuje i vstupně/výstupní zařízení. Řídicí sběrnice je množinou řídicích signálů, jednosměrných i obousměrných.

Zařízení, které v daný okamžik ovládá (řídí) sběrnicové vodiče, je *master*. Vždy je třeba rozhodnout o tom, kdo bude pro daný okamžik (datovou tranksakci) sběrnici řídit a oznámit příslušné jednotce, že se stala masterem. Tomuto procesu se říká *přidělování sběrnice, arbitrace* (angl. *arbitration*). Přidělování může být buď *centrální* nebo *distribuované*.

U centrálního přidělování existuje v systému jediný prvek (přidělovač, arbiter), který rozhoduje o přidělování. V jednodušších počítačích to může být sám procesor. Obecně, centrální přidělování je možné realizovat *na výzvu* nebo *na žádost*. V prvém případě se přidělovač opakovaně dotazuje (pomocí vyčleněných signálů) postupně jednotlivých zařízení (tzv. pooling), zda mají zájem řídit sběrnici a podle toho přiděluje. Druhý mechanismus spočívá v tom, že zařízení samo žádá o přidělení sběrnice přidělovač, má-li zájem sběrnici řídit, a čeká na odpověď od přidělovače. Je možné implementovat přidělovací mechanismy s prioritou, kdy každé zařízení má přidělenu prioritu (důležitost), podle které přidělovač rozhoduje v případě, že se sejdou dva požadavky najednou. Běžné jsou proměnné priority, kdy je zařízení, kterému byla přidělena sběrnice, snížena priorita, aby byla zaručena určitá spravedlnost. Příkladem centrálního přidělování na žádost je systém přidělování sběrnice PCI u počítačů PC. Každé zařízení (karta), které může ovládat sběrnici, má implementovánu dvojici jednosměrných dvoubodových signálů REQ (Request, žádost) a GNT (Grant, vyhověno). Signál REQ směřuje od karty k obvodům čipové sady na desce (přidělovači), GNT opačným směrem. Např. síťová karta, která přijmula paket ze sítě, chce ovládat sběrnici místo obvodů čipové sady, aby mohla autonomně přenést data do paměti. Nejprve požádá o přidělení sběrnice aktivací signálu REQ a čeká, dokud ji není potvrzeno přidělení signálem GNT. Pak může ovládat signály sběrnice.

V případě distribuovaného přidělování centrální přidělovač neexistuje, jednotky rozhodují o přidělení samostatně na základě vzájemné interakce. Příkladem distribuovaného přidělování je sběrnice SCSI, pomocí níž se připojují k počítačům pevné disky, CD mechaniky, páskové jednotky zpravidla v serverech. Sběrnice je osmi (dnes se již téměř nepoužívá) nebo šestnáctibitová. Na jedné sběrnici může být připojeno osm nebo šestnáct zařízení podle její šířky, z nichž jedno je řadič sběrnice (obvod v počítači na desce nebo rozšiřující zásuvné kartě). Zařízení mají přidělenu adresu od 0 do 6, resp. do 14, 7 a 15 jsou vyhrazeny pro řadič. Protože jsou vnější paměťová zařízení obecně pomalá zařízení, existuje určitá přístupová doba, tj. čas mezi tím, kdy disk obdrží od řadiče požadavek na čtení dat a okamžikem, kdy jsou data načtena a připravena k přenosu směrem k řadiči. Aby se dosáhlo maximální efektivity, řeší se problém tak, že po příjmu požadavku od řadiče se disk autonomně odpojí od sběrnice, aby ji neblokoval, a mezitím může řadič zaslat požadavek na operaci jinému zařízení. Má-li disk data připravena, musí se opět připojit na sběrnici a řadiči data zaslat. Může se stát, že v jednom okamžiku se budou chtít připojit na sběrnici třeba dva disky. Ve fázi přidělování každé zařízení, které má zájem o přidělení sběrnice, nastaví na log. 1 na datové sběrnici bit shodný se svoji adresou (disk s adresou 3 nastaví bit 3). Zároveň elektronika zařízení sleduje, zda není nastaven také nějaký bit s vyšší hodnotou, než je vlastní adresa. Pokud tomu tak je, má zájem o sběrnici zařízení s vyšší adresou, než je vlastní adresa, a zařízení se automaticky sběrnice vzdá. Rozhodování provádí každé zařízení samostatně stejným algoritmem. Sběrnici získává zařízení s nejvyšší adresou. Fáze přidělování se opakuje po skončení každé transakce na sběrnici.

Na závěr terminologickou poznámku. Někteří autoři uvádějí, že sběrnice je logické spojení více zařízení a fyzicky v daný okamžik propojuje pouze dvě, resp. jedinou řídicí jednotku a

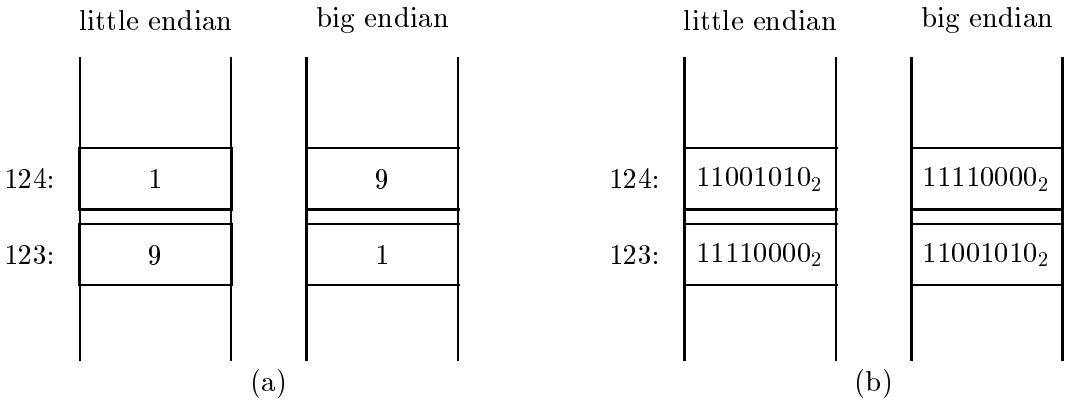
podřízené jednotky.

5.2 Organizace hlavní paměti

Hlavní paměť obsahuje instrukce a data. Obojí je kódováno ve dvojkové soustavě. Paměť je rozdělena na buňky, každá buňka má přiřazeno pořadové číslo (adresu). Přístup k obsahu buněk je právě adresní. Charakteristikou hlavní paměti je velikost (šířka) buněk (v bitech). Většinou je shodná se šířkou datové sběrnice procesoru počítače. Podle nejčastější šíře hovoříme o osmibitovém, šesnáctibitovém, dvaatřicebitovém atd. procesoru. Připomínáme, že 8 bitů označujeme termínem *slabika*[2], angl. *byte*, zkratkovitě B. *Slovo (word)* má zpravidla 16 bitů. Přiřazení slovu právě velikost 16 bitů se stalo zvykem zejména u procesorů Intel. Starší sálové počítače měly šířku dat i 12 bitů a také se používalo označení 12 bitové slovo. Podle firmy Intel se ujalo označení pro 32 slovo *dvojslovo (doubleword)*, pro 64 bitů *čtyřslovo (qword)*.

Kapacita paměti se uvádí v kilobytech (kB), resp. megabytech (MB) či gigabytech (GB). Čtenáři je jistě známo, že z důvodu používání dvojkové soustavy je $1\ kB = 2^{10}\ B = 1024\ B$, $1\ MB = 1024\ kB$, $1\ GB = 1024\ MB$. Čtenář se může setkat s označením kB i KB. Někdy je chápáno $1\ kB = 1000\ B$ a $1\ KB = 1024\ B$. My budeme dále v textu vždy rozumět pod označením $1\ kB = 1\ KB = 1024\ B$.

Problém nastává, pokud potřebujeme uložit data (číslo) do paměti, které má větší šířku, než je šířka buňky hlavní paměti. Uvažujme následující příklad. Máme paměť, kde do jedné buňky můžeme uložit jen jednu desítkovou číslici. Dostaneme úkol uložit od adresy 123 dvojciferné číslo 19. Musíme rozdělit číslo 19 podle řádů a uložit cifry 1 a 9 zvlášť, každou cifru do buňky na jiné adrese. Jsou dvě možnosti: na adresu 123 uložíme nižší řád, tj. cifru 9, a na adresu 124 (o 1 vyšší) cifru 1. Druhá možnost je opačná (obr. 5.3 (a)). Obdobně pro dvojkovou soustavu, máme paměť o velikosti buňky 1 slabiky (8 bitů) a máme uložit od adresy 123 šestnáctibitové slovo 1100101011110000_2. Rozdělíme je na osm bitů vyššího řádu 11001010_2 a na osm bitů nižšího řádu 11110000_2. Existují, stejně jako u příkladu s desítkovým číslem, dvě možnosti (obr. 5.3 (b)). Způsob, kdy ukládáme nižší řády na nižší adresy, se nazývá *little endian*. Opačný způsob, kdy ukládáme vyšší řády na nižší adresy, je označován jako *big endian*. Little endian používají procesory Intel a počítače od firmy DEC, big endian užívají Motorola, ze sálových počítačů užíval tento způsob např. IBM 360.



Obrázek 5.3: Způsoby uložení dat little endian a big endian

[2]Slabika je odborný výraz figurující v české normě pro terminologii výpočetní techniky od 70. let

5.3 Základní cyklus počítače

Činnost číslicového počítače je řízena programem, který je uložen v hlavní paměti. Program je posloupnost elementárních příkazů, tzv. *instrukcí*. Načítání a vykonávání instrukcí provádí řadič podle vývojového diagramu na obr. 5.4.

Obrázek 5.4: Základní cyklus počítače

Celý proces nazýváme základní cyklus počítače nebo také *instrukční cyklus*. Jednotlivé kroky instrukčního cyklu se nazývají *strojové cykly*. První strojový cyklus (načítání instrukce z hlavní paměti) se nazývá anglicky FETCH. Procesu rozpoznání, která instrukce byla načtena (tj. co se bude provádět - sčítat, násobit, ... a s jakými daty se bude pracovat), se říká dekódování instrukce. Podle typu instrukce se načítají data z paměti, se kterými se bude pracovat - *operandy*. Pak již může řadič vykonat instrukci - ovládat ostatní jednotky, např. ALU, aby sčítala. Po dokončení operace se uloží výsledek (do paměti nebo jsou data zaslána periferním zařízení). Před zahájením nového instrukčního cyklu se otestuje, zda není požadavek na přerušení. V kladném případě se přerušení obslouží. Někdy je nutné přejít na obsluhu přerušení dříve, např. při načtení neplatné instrukce.

5.3.1 Přerušení

Systém přerušení se využívá pro obsluhu asynchronních událostí v počítačích. Čtenář považuje za samozřejmé, že při stisku klávesy je na obrazovku vypsán znak, při pohybu myši dojde k překreslení kurzoru atd. Uvědomme si, že tyto události (stisk klávesy, pohyb myší) nemají žádný časový vztah k právě prováděnému instrukčnímu cyklu, jsou asynchronní. Obsluha událostí představuje vykonání podprogramů v jádře operačního systému. Je nutné přečíst kód stisknuté klávesy (resp. souřadnice myši) z periferních obvodů, nalézt tvar znaku (přepočítat

polohu korzoru), zapsat nová data do obrazové paměti apod. Pokud by neexistoval přerušovací systém, součástí operačního systému by musel být program, který by se periodicky dotazoval všech periferních zařízení, zda nedošlo k nějaké události. Vzhledem k poměru rychlosti procesoru vůči např. frekvenci stisku kláves uživatelem, resp. rychlosti vstupně/výstupních zařízení obecně, by většina dotazů měla negativní výsledek. Důsledkem by bylo zbytečné zpomalení běžících výpočtů. Problém řeší systém přerušení. Procesor má vstupní signál, označený zpravidla INT (Inrerrupt Request), žádost o přerušení. Pokud dojde k nějaké události, je to oznámeno procesoru právě aktivací signálu INT. Řadič počítače po skončení instrukčního cyklu testuje stav signálu INT. Je-li aktivní, přeruší se vykonávání aktuálního programu a procesor začne vykonávat program jiný - obslužnou rutinu přerušení. Ta přečte kód stisknuté klávesy, data od myši apod., vypíše znak či překreslí kurzor. Po skončení rutiny se procesor vrátí k vykonávání původního programu.

Čtenář si jistě položí otázku, jakým mechanismem je rozpoznán zdroj události a jak se určí, kde se v paměti nachází kód přerušovací rutiny. Většinou existuje více signálů žádosti o přerušení a každému periferním zařízení je přiřazen jeden signál. Pokud má procesor pouze jeden signál INT, přidává se k němu zvláštní obvod (řadič přerušení), který má několik vstupních žádostí o přerušení. Např. u počítačů PC jsou kaskádně zapojené dva čipy Intel 8259, dnes sdružené spolu s jinými obvody v některém VLSI obvodu čipové sady. Každý z obvodů má osm žádostí přerušení IRQ0-IRQ7. Na linku IRQ0 je připojen časovač, na linku IRQ1 obvody klávesnice atd. Objeví-li se na některém vstupu žádost o přerušení, obvod oznámí událost procesoru signálem INT. Na konci instrukčního cyklu řadič procesoru aktivuje signál INTA (Interrupt Acknowledge, potvrzení žádosti o přerušení), který je zapojen do obvodu 8259. Na základě potvrzení žádosti o přerušení zašle obvod 8259 procesoru po datové sběrnici číslo přerušení. V hlavní paměti jsou na vyhrazených adresách (u PC od adresy 0) uloženy adresy počátků kódu přerušovacích rutin pro jednotlivé žádosti o přerušení, tzv. *vektory přerušení*. Vektory přerušení a obslužné rutiny uloží do paměti operační systém při svém startu (rutiny jsou součástí ovladačů). Procesor (řadič) poté, co obdrží číslo přerušení, načte z hlavní paměti adresu obslužné rutiny a začne vykonávat instrukce od této adresy. Předtím uloží na zásobník návratovou adresu a příznakový registr (viz dále).

5.4 Registry

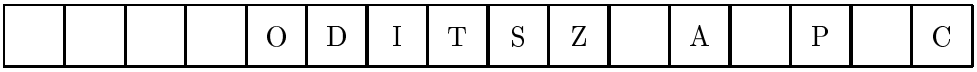
Na obr. 5.1 je orámován řadič počítače spolu s aritmetickou jednotkou a celek je označen jako *procesor*. Pokud bychom se chtěli obrázkem přiblížit ke struktuře reálného procesoru, patřila by do rámečku i část hlavní paměti. Procesor, resp. mikroprocesor[3], obsahuje speciální paměťové buňky (*registry*), do kterých se ukládají výsledky a mezivýsledky operací a stavové informace řadiče. Důvodem použití registrů je rychlý přístup k datům, protože operace čtení a zápisu z/do hlavní paměti jsou vzhledem k rychlosti řadiče pomalé.

Existují registry, které nalezneme v každém procesoru. *Programový čítač PC (Program Counter)* obsahuje adresu právě vykonávané instrukce. Načtení instrukce v prvním strojovém cyklu není nic jiného, než vyslání obsahu programového čítače na adresovou sběrnici a provedení operace čtení z hlavní paměti. Po skončení instrukčního cyklu je jeho obsah automaticky zvýšen. U procesorů Intel má programový čítač název Instruction Pointer (IP).

Výsledky a mezivýsledky aritmetických operací se ukládají do *střadače (Acumulator A)*. Procesor může mít větší počet střadačů. Některé aritmetické instrukce pracují jen se střadači. Stav procesoru se po ukončení operací ukládá do *příznakového registru (Flags F)*. Na základě hodnot příznaků se větví běh programu (podmíněnými instrukcemi skoku). Na obr. 5.5 je příznakový

[3]Od 70. let, kdy se podařilo celý procesor integrovat do jednoho pouzdra integrovaného obvodu, se označuje termínem mikroprocesor.

registr procesorů řady Intel 80x86.



Obrázek 5.5: Příznakový registr procesorů Intel

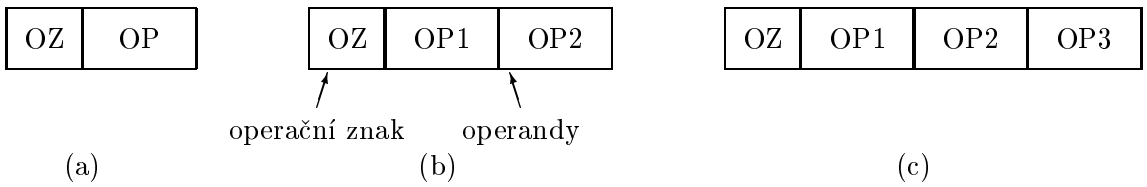
Popíšeme význam příznaků, i když význam většiny z nich pochopí čtenář až po výkladu aritmetiky počítačů:

- **C (carry flag)** - přenos - je nastaven, došlo-li při aritmetické operaci k přenosu z nejvyššího řádu
- **P (parity)** - parita - je nastaven při chybě parity paměti
- **A (auxiliary carry)** - pomocný přenos - je nastaven, došlo-li při aritmetické operaci k přenosu mezi bitem 3. řádu a 4. řádu (využívá se v desítkové aritmetice)
- **Z (zero)** - nulový příznak - je nastaven, byl-li výsledek předchozí aritmetické nebo logické operace nulový
- **S (sign)** - znaménko - je nastaven, byl-li výsledek předchozí aritmetické operace záporný
- **T (trap)** - trap - je nastaven v krokovacím režimu procesoru (specialita I80x86)
- **I (interrupt)** - povolení přerušení - zakazuje/povoluje obsloužit přerušení
- **D (direction)** - směr - udává směr čítání adres (nahoru/dolů) u některých speciálních instrukcí blokového přenosu dat
- **O (overflow)** - přetečení - je nastaven, byl-li výsledek předchozí aritmetické operace větší, než je možné zobrazit na počet bitů daný šířkou registrů

Příznaky C, A, Z, S, I, O nalezneme v každém procesoru. Obvykle mají procesory ještě sadu univerzálních registrů, které mohou být využity jako pomocné střadače, obsahovat pomocné adresy dat do paměti či čítače cyklů. Speciální registr *ukazatel zásobníku (Stack Pointer SP)* ukazuje na vrchol zásobníku. Zásobník je datová struktura, pro kterou je vyhrazena část hlavní paměti. Slouží také pro ukládání mezivýsledků a je využíván při přerušení, instrukcemi volání podprogramu a návratu z podprogramu pro uložení návratové adresy. Více o zásobníku v další podkapitole.

5.5 Instrukční sada

Množinu instrukcí nějakého procesoru nazýváme *instrukční sada*. Instrukce je příkaz k provedení elementární operace, kódovaný číselně ve dvojkové soustavě. Záleží na návrháři procesoru, jakou instrukční sadu pro procesor vytvoří. Ačkoliv se sady instrukcí různých procesorů liší, určité skupiny instrukcí jako aritmetické, logické, skokové má každý procesor. Instrukce obecně se skládá z *operačního znaku* a *operandů* (obr. 5.6), tvaru instrukce se říká *formát instrukce*. Operační znak je kód operace, tj. co se bude provádět (sčítej, odčítej, ...). Operand je identifikace, s čím se bude operace provádět, resp. kam uložit výsledek. Označení operandů, které popíšeme dále, není jediné možné. Autor skripta zjistil, že v dokumentaci procesorů různých typů má totéž označení typu operandu zcela jiný význam. Je zde uvedeno to, na které je autor „zvyklý“. Příklady instrukcí, které budeme uvádět, jsou instrukce procesoru Intel 80x86. Instrukční sada



Obrázek 5.6: Obecný formát instrukce

procesoru není příkladem nejlepší sady, vzhledem k rozšíření řady procesorů v počítačích PC je ale nejznámější.

U některých instrukcí může operand i chybět, instrukce obsahuje pouze operační znak a operand určuje sám operační znak, Takový operand se označuje jako tzv. *implicitní*. Operandem může být také konstanta přímo uložená v instrukci a takový operand se nazývá *přímý (direct)*. Pokud je operandem adresa dat do hlavní paměti nebo identifikace registru, jde o *nepřímý operand (indirect)*. Adresa do paměti jako nepřímý operand může být buď zapsána přímo v instrukci (přímá adresa) nebo zde může být odkaz na registr, který tuto adresu obsahuje (nepřímá adresa). Počet operandů v instrukci bývá od jedné do tří (instrukce se označují jedno- až tříadresové, i když operandem je registr). U jednoadresových instrukcí (obr. 5.6 (a)) se provede unární operace nad operandem OP. U tříadresových instrukcí (obr. 5.6 (c)) se provede operace mezi operandy OP1, OP2 a výsledek se uloží do OP3 (OP3 musí být nepřímý operand). U dvouadresových instrukcí (obr. 5.6 (b)) se provede operace mezi dvěma operandy OP1, OP2 a výsledkem se přepíše první operand OP1 (musí být nepřímý). Tříadresové instrukce (které byly běžné u některých střediskových počítačů) zabírají velkou část operační paměti. Analýzy programů ukázaly, že jejich využití bylo minimální, a proto se dnes v instrukčních sadách nevyskytují.

Zápis instrukce ve dvojkové soustavě je pro čtenáře nepřehledný. Proto se používá mnemotechnického zápisu ve formě anglických zkratek a tento jazyk zápisu se označuje jako *jazyk symbolických instrukcí (assembler)*. Budeme zapisovat instrukce samozřejmě symbolicky, uvedeme jeden příklad zápisu ve dvojkové soustavě. Příkladem instrukce s implicitním operandem je `CLI` (clear interrupt), instrukce má jen operační znak. Instrukce vynuluje příznak I v příznakovém registru (zakáže obsluhu přerušení) - operační znak implikuje, s čím se pracuje. Jednoadresová instrukce s nepřímým operandem je např. `INC AX`, což je zvýšení obsahu střadače AX o jedničku. Dvouadresová je instrukce `MOV AX,5`, která znamená přesun konstanty 5 do registru AX. První operand je nepřímý (registr), druhý přímý (konstanta 5). Varianta instrukce přesunu `MOV BX,[3AF8]` má dva nepřímé operandy, druhý je přímá adresa do paměti. Význam instrukce je přesun 16 bitové hodnoty z hlavní paměti z adresy $3AF8_{16}$ do registru BX (16 bitová hodnota se přesouvá z důvodu, protože je registr BX 16 bitový). Ještě ukážeme možnost, kdy adresa je uložena v registru: `MOV BX,[BP]` značí přesun 16 bitové hodnoty do registru BX z paměti z adresy, která je uložena v registru BP, Base Pointer (nepřímá adresa).

Přímé a nepřímé operandy je možno kombinovat, ne však libovolně. Lze přesouvat, resp. provádět operace mezi registry, registrem a pamětí, registrem či pamětí a přímým operandem, samozřejmě podle typu instrukce. Nejsou obvyklé operace přímo mezi dvěma paměťovými buňkami, kdy by oba nepřímé operandy byly adresy do paměti (typ `MOV paměť, paměť`). Tuto možnost mají jen procesory Motorola. Poznáme, že u procesorů architektury RISC (viz dále) jsou dovoleny operace pouze mezi registry.

Po obecném seznámení s formáty instrukcí uvedeme běžné rozdělení instrukcí:

- aritmetické - ADD (sečti), SUB (odečti), MUL (vynásob), DIV (vyděl), INC (zvyš o 1), DEC (sniž o 1), CMP (porovnej)
- logické - AND, OR, XOR, NOT, TEST

- posuvy a rotace - SHL (posuv vlevo), SHR (posuv vpravo), ROL (rotace vlevo), ...
- přesuny dat - MOV, IN, OUT, PUSH, POP
- skoky - JMP, JZ, JAE
  volání CALL, RET, RETZ, RETI
  cykly - LOOP

Aritmetické instrukce mají, kromě INC a DEC, dva operandy, např. `ADD AX,BX` sečte obsahy registrů AX a BX a výsledek uloží do AX. Logické instrukce provádějí logické operace, tedy `AND [3AF8],BX` provede logický součin šestnáctibitové hodnoty v hlavní paměti na adrese $3AF8_{16}$ a obsahu registru BX; výsledek je uložen zpět do hlavní paměti. Instrukce `NOT` má jeden operand a provádí bitovou negaci. Aritmetické operace modifikují příznakový registr, tj. podle výsledku se nastavují příznaky C,S,Z,O. Instrukce pro posuv (shift) a rotaci provádějí bitové posuvy (rotace) s daty. Při posuvu vlevo bit nejvyššího řádu vypadává a bit nejnižšího řádu se nastavuje na 0, při rotaci vlevo se hodnota bitu nejvyššího řádu kopíruje do bitu nejnižžšího řádu. Posuvy a rotace vpravo jsou analogické. Předpokládejme, že obsah osmibitového registru AL je $AL = 11010010_2$, obsah registru $CL = 1$. Po provedení instrukce `SHL AL,CL` bude obsah registru AL posunut o jeden bit doleva, tj. $AL = 10100100_2$; po provedení `ROL AL,CL` bude obsah registru AL rotován o jeden bit doleva, tj. $AL = 10100101_2$.

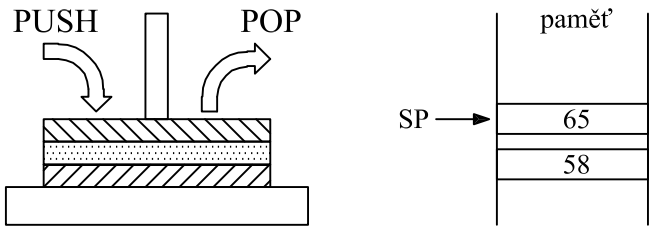
Instrukce přesunu dat MOV provádí přesun dat mezi registry a pamětí, příklad jsme již uvedli. Instrukce IN a OUT provádějí přesuny dat mezi registy a vstupně/výstupními zařízeními. Procesory Intel mají oddělený adresový prostor hlavní paměti a vstupně/výstupních zařízení. Princip je v obou případech stejný, na adresové sběrnici se objeví platná adresa a na datové přenášená data. Pouze na řídící sběrnici se buď aktivují signály čtení/zápis z/do paměti nebo čtení/zápis z/na periférii. Procesory firmy Motorola mají pouze instrukci MOV pro přesun a adresový prostor je paměťový. Periférie jsou mapovány do adresového prostoru paměti místo paměťových buněk.

Testovací instrukce jsou analogií aritmetických a logických instrukcí. Např. instrukce CMP (compare) provede odečtení operandů, ale neuloží výsledek, pouze nastaví příznak Z (nula) příznakového registru, byl-li rozdíl nulový, tj. operady stejné. TEST provede logický součin AND obou operandů, ale také pouze nastaví příznak.

Zvláštní kapitolu tvoří skoky a volání. Touto skupinou instrukcí je možné ovlivnit pořadí vykonávání instrukcí (viz charakteristiky Von Neumannovy architektury). Skoky jsou *nepodmíněné a podmíněné*. Nepodmíněný skok (jump) má tvar `JMP adresa`; význam je zřejmý - následující instrukce se načte z adresy `adresa`. Podmíněný skok se provede, je-li splněna podmínka; podmínkou je hodnota nějakého příznaku, např. skok `JZ adresa` (jump zero) se provede, je-li nastaven příznak Z, tj. výsledek předchozí operace byl nulový. Negovaný podmíněný skok má mnemoniku `JNZ adresa` (jump not zero). Podmíněných skoků je několik, např. JAE (jump above or equal) znamená skok, je-li nastaven příznak Z nebo S nenastaven, tj. výsledek operace (porovnání) je nulový nebo kladný.

Instrukce volání je zvláštní případ skokové instrukce. Při provádění instrukce se využívá zásobník. Zásobník je datová struktura typu LIFO (last in, first out). Nad zásobníkem se provádějí dvě operace: PUSH (uloží hodnotu na vrchol zásobníku) a POP (odebere hodnotu z vrcholu zásobníku (první se odebírá hodnota poslední vložená). Uložíme-li na zásobník pomocí několika operací PUSH řadu hodnot, první operace POP vyzvedne poslední uloženou hodnotu, další operace POP předposlední atd. Ukládání a výběr dat ze zásobníku si můžeme představit jako navlékání kroužků na osu (obr. 5.7). Odebrat můžeme poslední navlečený kroužek.

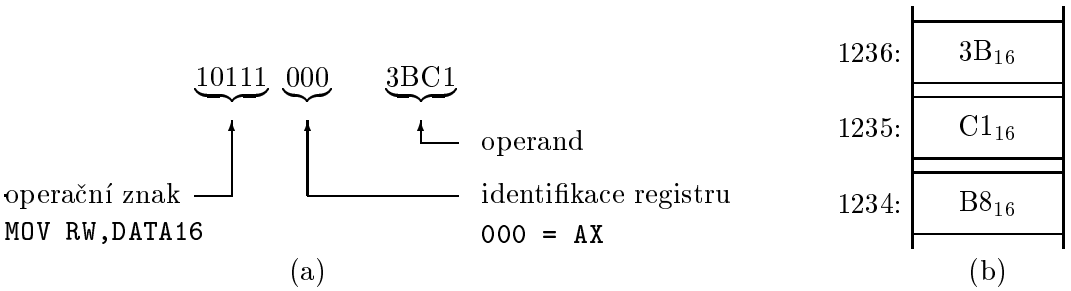
Pro zásobník je vyhrazena oblast v hlavní paměti a na jeho vrchol (top) ukazuje registr SP (Stack Pointer, ukazatel zásobníku). Instrukce `PUSH registr` uschovává na vrchol zásobníku obsah registru, instrukce `POP registr` ukládá do registru hodnotu z vrcholu zásobníku. Po vykonání instrukce dochází automaticky k modifikaci hodnoty SP.



Obrázek 5.7: Princip ukládání dat na zásobník

Instrukce `call adresa` se chová obdobně jako JMP, tj. provede se skok na adresu `adresa` (do tzv. podprogamu), ale předtím se na vrchol zásobníku uloží návratová adresa, tj. adresa instrukce následující za CALL. Na konci podprogramu je umístěna instrukce RET (return), která vyzvedne ze zásobníku návratovou adresu a provede se skok na tuto adresu. Běh programu se vrátí na původní posloupnost instrukcí za CALL. Existují i instrukce podmíněného návratu RETZ (podle příznaku Z). Pokud si podprogram uchová na zásobník během výpočtu nějaká data pomocí instrukce PUSH, musí je všechna odebrat instrukcí POP před instrukcí RET. Instrukce RET vyzvedne jako návratovou hodnotu hodnotu na vrcholu zásobníku; v případě zapomenuté hodnoty je pak adresa návratu nesmyslná. Stejným mechanismem je řešeno volání obslužné rutiny při přerušení, s tím rozdílem, že skok se neprovede na základě instrukce CALL, ale na základě vnějšího podnětu (signálu INT) a adresa je dána přerušovacím vektorem. Při volání přerušovací rutiny se na zásobník uloží navíc kromě návratové adresy i stavový registr procesoru. Přerušovací rutina musí být ukončena speciální instrukcí RETI (někdy značená IRET), která ze zásobníku kromě návratové adresy vyzvedne i stavový registr procesoru a obnoví jej.

Pro demonstraci zakódování instrukce ve dvojkové soustavě jsme zvolili instrukci `MOV AX,3BC1`, která má jeden nepřímý (registr) a jeden přímý operand (konstantu). Instrukce ukládá do šestnáctibitového registru AX hodnotu $3BC1_{16}$, která je uložena za operačním znakem. Pět bitů 10111_2 je operační znak instrukce MOV RW,DATA16, což znamená přesun šestnáctibitové hodnoty do šestnáctibitového registru. Tříbitová hodnota za operačním znakem adresuje registr, zde 000 registr AX. Operační znak spolu s adresou registru tvoří jednu slabiku $10111000_2 = B8_{16}$. Dvě následující slabiky jsou druhý operand - přímý, obr. 5.8 (a). Celkem instrukce zabírá v operační paměti 3 slabiky.



Obrázek 5.8: Zakódování a uložení instrukce v paměti

Ještě si nakreslíme, jak bude v paměti instrukce uložena, třeba od adresy 1234. Ačkoliv mají současné procesory Intel i paměťové čipy šířku datové sběrnice 64 bitů, paměť je organizována slabikově a je možné ji i po slabikách adresovat. Instrukce bude v buňkách za sebou na třech adresách. Na adrese 1234 bude uložena hodnota $B8_{16}$ (operační znak a identifikace registru). Protože procesory Intel ukládají data ve formátu little endian, bude v buňce s nižší adresou 1235 uložena slabika nižšího řádu konstanty $3BC1_{16}$, tedy $C1_{16}$, a na adrese 1236 slabika $3B_{16}$, obr. 5.8 (b).

K organizaci paměti počítačů PC ještě malou poznámku. Procesory Intel od 80386 mají rozšířené registry na 32 bitů, resp. 64 bitů, označené Extended, např. EAX. V instrukční sadě jsou navíc odpovídající instrukce pro práci s těmito registry (MOV EAX,data32). Samozřejmě je možné stále využívat šestnáctibitové a osmibitové přenosy, je to ale neefektivní, protože v rámci jednoho sběrnicového cyklu je možné přenést 32, resp. 64, bitů dat najednou. U osmi a šestnáctibitových přenosů je část datové sběrnice nevyužita. Sdružíme-li data do dvojslov (32 bitů) a uvážíme-li, že paměť je adresovatelná po slabikách (8 bitů), vychází, že celá 32 bitová slova začínají na adresách dělitelných 4. Proto je ve dvaatřicetibitových překladačích programovacích jazyků volba, aby překladač umísťoval proměnné na adresy dělitelné čtyřmi; pouze tak je přenos 32 bitových slov efektivní (jeden cyklus sběrnice).

V počátcích programování číslicových počítačů se programy zapisovaly přímo v instrukcích v zakódované podobě. Takovému programu se říká program ve strojovém kódu. Později se programy zapisovaly textově v jazyku symbolických instrukcí (assembleru) a do strojového kódu se převáděly překladačem. V současné době se v assembleru píší jen jádra operačních sysémů a někdy ovladače, ostatní software se programuje ve vyšších programovacích jazycích (Pascal, C, Java) a do kódu se program převádí překladačem.

Na závěr si ukážeme fragment podprogramu (funkce) zapsaný v assembleru. Půjde o funkci, která v poli, které je zakončeno číslem 0, vyhledává osmibitové číslo. V registru AL (8 bitový) je podprogramu předáno hledané číslo, v registru BX adresa počástku pole; počet výskytů znaku se vrací v registru CX:

```
        XOR CX,CX     ;nulování čítače
cykl:   MOV AH,[BX]   ;přesuň aktuální číslo  do AH
        CMP AH,0      ;je načteno číslo 0
        JZ pryc       ;pokud ano, konec cyklu
        CMP AL,AH     ;porovnání čísla v˜poli s hledaným
        JNZ nezvys    ;pokud není shoda - skok
        INC CX        ;jinak zvyš čítač o 1
nezvys: INC BX        ;posun adresu na další číslo
        JMP cykl      ;skok na začátek cyklu
pryc:   RET
```

Číslo z pole, tj. z adresy, která je uložena v registru BX (`MOV AH,[BX]`), se přesune do pomocného registru AH (8 bitů). Pak se porovná s nulou (`CMP AH,0`). Pokud číslo mělo hodnotu 0 (instrukce CMP nastavila příznak Z), pak se provede skok na adresu se symbolickým návěštím `pryc` (podmíněný skok `JZ`), kde je instrukce `RET` návratu z podprogramu. Jinak se porovná načtené číslo s hledaným, pokud je shodné, zvýší se čítač (`INC CX`).

Odpovídající funkce, zapsaná v jazyce C (ne příliš efektivně), by vypadala asi takto:

```
int vyhledej(unsigned char *data, unsigned char cislo)
{
  int citac=0;
  while(*data !=0)
  {
    if (*data == cislo) citac++;
  }
  return citac;
}
```

5.6 Počítače CISC a RISC, proudové zpracování instrukce

V podkapitole s tematikou o instrukční sadě jsme poznali typy instrukcí a operandů. Návrháři procesorů se od počátku snažili rozšiřovat instrukční sadu o stále složitější instrukce, které by více odpovídaly syntaktickým konstrukcím vyšších programovacích jazyků, aby usnadnili práci programátorům i tvůrcům překladačů. Zvětšil se i počet možností adresních módu u nepřímých operandů, např. u procesorů Intel existuje instrukce `MOV AX,[BP+SI]`. Způsob nepřímé adresace u druhého operandu se nazývá modifikace indexovým registrem - k adrese v registru BP (Base Pointer) se přičítá obsah registru SI (Source Index) a součet se použije jako adresa do paměti. Tato adresace se využívá při zpracování polí, kde do bázového registru se uloží adresa počátku pole a posuv (index) je v registru SI a modifikuje se.

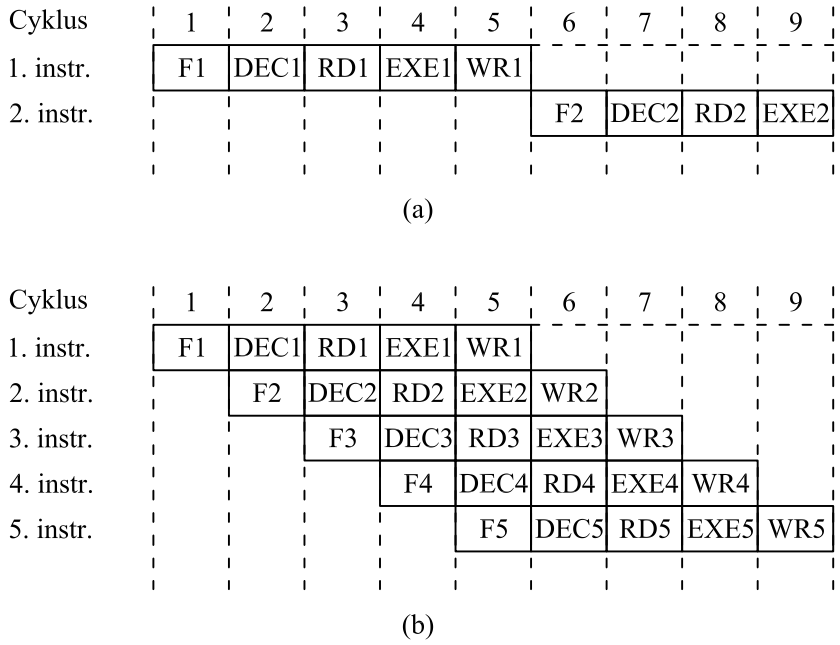
Důsledkem rozšiřování instrukcí i adresačních možností je složitější jádro procesoru (řadič). Kódování složitých instrukcí ve dvojkové soustavě nemá jednotný formát, zabírá různý počet buněk v paměti (díky variabilitě operandů), tím pádem je složitější i dekódování instrukce a načítání operandů. Analýzou programů ve strojovém kódu, psaných programátory i překládaných z vyšších jazyků, bylo zjištěno, že zastoupení složitých instrukcí není ve skutečnosti v kódu tak časté (detailněji např. [13]). Proto se koncem 70. let skupina odborníků na univerzitách v Berkley a Stanfordově univerzitě rozhodla optimalizovat procesor a instrukční sadu naopak redukcí instrukční sady. Počítače, resp. procesory, označili jako *RISC (Reduced Instruction Set Computer) - počítač s redukovanou sadou instrukcí.* Procesory se složitou instrukční sadou nesou pak označení *CISC (Complex Instruction Set Computer).* Základní rysy architektury RISC lze shrnout do následujích bodů:

- počet instrukcí i adresových módů je malý
- pro spolupráci s hlavní pamětí jsou použity dvě instrukce LD, ST
- instrukce mají pevnou délku a pevný formát s přesně vymezenými významy jednotlivých bitů
- procesor obsahuje velký počet registrů (registrově orientovaný systém)
- proudové zpracování instrukce

Počet instrukcí je redukován, operandy jsou pouze přímé a z nepřímých registry. Zatímco u procesorů Intel, typického zástupce CISC, lze napsat instrukci ve tvaru `ADD AX,[3456]` (k obsahu registru AX přičti data z adresy 3456 hlavní paměti), u procesorů RISC lze sčítat pouze obsahy registrů. Pro spolupráci s hlavní pamětí existují jen dvě instrukce. `LD regs,adresa` (LOAD) slouží k přesunu obsahu buňky hlavní paměti do registru, `ST regs,adresa` (STORE) slouží k přesunu obsahu rogistru do buňky hlavní paměti. Proto se architektura procesorů RISC označuje také jako *Load/Store architektura.* Pokud chceme sečíst dvě čísla z hlavní paměti, program u procesorů typu RISC musí nejprve pomocí dvou instrukcí LD přesunout data do registrů, pak hodnoty sečíst a uložit pomocí instrukce ST výsledek zpět do paměti. Čtenář může mít oprávněnou námitku, že takový program je složen z většího počtu instrukcí než program pro CISC, kde máme větší adresové možnosti (stačí přesunout jednu hodnotu do registru pomocí MOV a pak zapsat instrukci ADD [adresa],regs). To je jistě pravda, ale díky malému počtu adresních možností v instrukcích lze navrhnou formát instrukce s pevnou délkou, přesnými hranicemi mezi částmi instrukce (operačním znakem a operandy) a přesně vymezenými významy jednotlivých bitů. To vše usnadňuje návrh řadiče procesoru a urychluje načítání, dekódování instrukce, její vykonání i práci s operandy, navíc umožňuje proudové zpracování instrukce. Protože se operace provádějí výhradně mezi registry, jsou datové cesty jednodušší. Výsledkem je mnohem rychlejší procesor za stejných technologických podmínek (stupeň integrace, tepelné ztráty)

z hlediska dosažitelné frekvence hodin i měřeného výkonu. Výkon procesorů se udává v jednotkách MIPS (Mega Instruction per Second) - počet milionů vykonaných instrukcí za sekundu - a měření ukazují, že výsledný výkon procesoru je vyšší i přes to, že programy obsahují větší počet instrukcí.

Protože se provádějí operace výhradně mezi registry, mají procesory RISC registrů velký počet. Např. procesor Sun SPARC do pracovních stanic má sadu 255 registrů, z nichž je možné v daném okamžiku využít okno o velikosti 24 registrů. Okna je možné přepínat s překryvem. Mechanismu se využívá při volání procedur a funkcí, kdy se pomocí překrývajících oken předávají parametry a výsledky.



Obrázek 5.9: Neproudové a proudové zpracování instrukce

Vlastnosti architektury RISC umožňují snadnou implementaci *proudového zpracování instrukce (pipeline)*. Základní instrukční cyklus, který jsme popsali v podkapitole 5.3, je znovu na obr. 5.9 (a). Skládá se z pěti strojových cyklů: načtení instrukce F (jako FETCH), dekódování instrukce DEC, načtení operandů RD, vykonání instrukce EXE (Execution) a zápis výsledku WR. Strojové cykly jsou vykonávány sériově, pro první instrukci (F1, DEC1,...) a pak pro druhou instrukci (F2, DEC2,...). Pokud navrhneme řadič počítače z paralelních jednotek, které vykonávají strojové cykly samostatně, je možné zpracovat instrukci proudově, obr. 5.3 (b). V prvním cyklu se načítá první instrukce (F1), ve druhém strojovém cyklu, kdy se první instrukce dekóduje (DEC1), se zároveň již načítá z hlavní paměti druhá instrukce (F2) atd. Zatímco je při neproudovém zpracování během devíti cyklů zcela dokončena první instrukce a vykonání druhé právě probíhá, při proudovém zpracování jsou plně dokončeny čtyři instrukce a vykonání páté instrukce probíhá. Dá se říci, že během každého strojového cyklu je provedena jedna instrukce. Experimenty prokázaly, že rozdělení proudu do naznačeným pěti sekcí je optimální. Paralelu proudového zpracování instrukce můžeme najít v pásové výrobě např. v automobilkách. Pozorným rozborem obr. 5.3 (b) zjistíme, že dochází ke kolizi od strojového cyklu 3. Z paměti se načítá třetí instrukce (F3) a zároveň se načítají operandy (data) pro zpracování první instrukcí (RD1) atd. Pokud je operandem registr, ke kolizi nedochází, jde-li o instrukci LD (load), máme zde požadavek na současné čtení dat a instrukce. Proto musí mít počítače RISC *oddělenou paměť programu a dat*. Architektura s oddělenou pamětí programu a dat se označuje jako *harwardská architektura*. Další kolizi vidíme od cyklu 5, kdy je současný požadavek na čtení operandů

(dat) pro třetí instrukci a zápis výsledku (dat) od první instrukce. To je možné pouze s využitím dvoubránové synchronní paměti dat, která umožňuje současné čtení i zápis. Realizovat tímto způsobem celou hlavní paměť by bylo náročné, ale vzpomeňme, že operace se u RISCových procesorů provádějí pouze mezi registry; stačí vytvořit dvoubránové pole registrů uvnitř procesoru.

Proudové zpracování instrukce může být narušeno při podmíněných skocích, kdy není předem známo, jaká bude další načítaná instrukce. Problém se řeší překladač vhodným pořadím instrukcí (např mezi aritmetickou instrukci, která ovlivní skok a vlastní instrukcí skoku vloží nějakou, která neovlivní příznaky) a speciální jednotkou řadiče, tzv. *blokem predikce skoku*. Pokud přesto dojde dopředu k nahrání nesprávné větve programu, jsou instrukce „zahozeny“ a v proud zpracování je na několik cyklů přerušen.

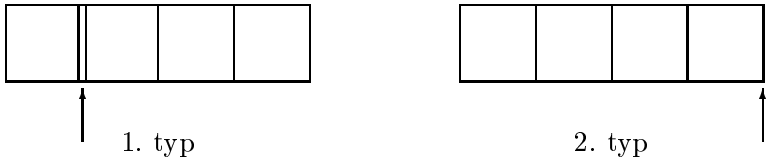
Moderní procesory jsou spojením procesorů RISC a CISC. Nejznámější zástupci „čistých“ RISC procesorů jsou Sun SPARC, Alpha a PowerPC. Moderní procesory Pentium mají rysy architektur CISC i RISC. Čtenář si možná položí otázku ohledně souvislosti mezi dvoubránovými pamětmi a harwardskou architekturou, což v počítačích PC není, složitou instrukční sadou a proudovým zpracováním instrukce se všemi kolizemi. Právě z tohoto důvodu jsou v procesoru „alespoň“ odděleny vyrovnávací paměti (cache) instrukcí a dat. Podrobnější informace o RISC architekturách najde čtenář v literatuře [13].

5.7 Aritmetika počítačů

Aritmetické a logické operace v počítači provádí aritmeticko-logická jednotka (ALU - Arithmetical and Logical Unit). Zpracovávaná data jsou zobrazena ve dvojkové soustavě. V úvodní kapitole jsme rozebírali číselné soustavy a převody mezi nimi. Oproti matematice musíme u reálných počítačů uvažovat konečnou šířku registrů procesoru (počet bitů), konečný počet bitů aritmetických obvodů a tím i omezený rozsah zobrazovaných a zpracovávaných dat.

5.7.1 Čísla v počítačích

Konečná velikost registrů omezuje počet zobrazených řádů čísel. Schéma, které vyjadřuje zobrazitelné řády, se nazývá *řádová mřížka*, obr. 5.10. Řádová mřížka je charakterizována celkovým počtem řádů a polohou řádové čárky, ve formě čísel n a m. Číslo n je nejvyšší zobrazitelný řád, číslo m je nejnižší řád (tím je dán jednoznačně celkový počet řádů i poloha řádové čárky). K úplnému určení je ještě potřeba dodat základ soustavy z. V počítačích jsou nejpoužívanější dva typy řádových mřížek, obr. 5.10. První typ se používá při zobrazování desetinných čísel, druhý typ pro zobrazování celých čísel. První mřížku na obrázku charakterizují čísla $n = 0$, $m = -3$, druhou mřížku dvojice $n = 3$, $m = 0$.



Obrázek 5.10: Příklady řádové mřížky

Kromě čísel n a m je možné charakterizovat mřížku číslem Z a ε. $Z = z^{n+1}$ je *modul řádové mřížky*, $\varepsilon = z^m$ je jednotka řádové mřížky. Jednotka řádové mřížky je nejmenší nenulové číslo, které lze v dané mřížce zobrazit. Obecně platí, že v dané mřížce lze zobrazit jen celistvé násobky jednotky řádové mřížky. Modul Z je nejmenší celočíselný násobek jednotky řádové

mřížky, který již není zobrazitelný v řádové mřížce. Modul a jednotku řádové mřížky můžeme vypočítat podle výše uvedených vztahů, mnemotechnicky je lze vytvořit následujícím postupem: modul řádové mřížky dostaneme tak, že řádovou mřížku vyplníme nulami a před ni napíšeme 1. Jednotku řádové mřížky obdržíme, když do nejnižšího řádu napíšeme 1, zbytek vyplníme nulami. S velikostí řádové mřížky souvisí již zmiňovaný rozsah zobrazitelných čísel. V řádové mřížce s parametry Z a ε lze zobrazit celočíselné násobky jednotky řádové mřížky $A = k\varepsilon$ v rozsahu $0 \leq A < Z$. Největší zobrazitelné číslo je $A_{max} = Z - \varepsilon$, počet zobrazitelných čísel $N = z^{n+1-m}$.

Příklad 5.1:

Uvažuje řádovou mřížku prvého typu s parametry $n = 0$ a $m = -3$ podle obr. 5.10. Určete modul Z, jednotku ε, rozsah zobrazitelných čísel jak pro desítkovou ($z = 10$) tak pro dvojkovou ($z = 2$) soustavu.

Řešení:

Pro desítkovou soustavu je modul roven $Z = 10,000_{10}$ a jednotka řádové mřížky $\varepsilon = 0,001_{10}$. Nejmenší zobrazitelné číslo je 0, největší $A_{max} = 10 - 0,001 = 9,999_{10}$, počet zobrazených čísel je $N = 10^4 = 10000$.

Pro dvojkovou soustavu je modul roven $Z = 10,000_2 = 2_{10}$ a jednotka řádové mřížky $\varepsilon = 0,001_2 = 0,125_{10}$. Nejmenší zobrazitelné číslo je 0, největší $A_{max} = 1,111_2 = 1,875_{10}$, počet zobrazených čísel je $N = 2^4 = 16$.

∘

Příklad 5.2:

Uvažuje řádovou mřížku druhého typu s parametry $n = 3$ a $m = 0$ podle obr. 5.10. Určete modul Z, jednotku ε, rozsah zobrazitelných čísel jak pro desítkovou ($z = 10$) tak pro dvojkovou ($z = 2$) soustavu.

Řešení:

Pro desítkovou soustavu je modul roven $Z = 10000_{10}$ a jednotka řádové mřížky $\varepsilon = 0001_{10}$. Nejmenší zobrazitelné číslo je 0, největší $A_{max} = 100001 = 9999_{10}$, počet zobrazených čísel je $N = 10^4 = 10000$.

Pro dvojkovou soustavu je modul roven $Z = 10000_2 = 16_{10}$ a jednotka řádové mřížky $\varepsilon = 0001_2 = 1_{10}$. Nejmenší zobrazitelné číslo je 0, největší $A_{max} = 1111_2 = 15_{10}$, počet zobrazených čísel je $N = 2^4 = 16$.

∘

Příklad 5.3:

Uvažuje osmibitový registr procesoru pro zobrazení celých čísel ve dvojkové soustavě. Určete modul Z, jednotku ε, rozsah zobrazitelných čísel.

Řešení:

Registr představuje řádovou mřížku druhého typu s parametry $n = 7, m = 0$. Modul je roven $Z = 2^8 = 100000000_2 = 256_{10}$ a jednotka řádové mřížky $\varepsilon = 1_2$. Jsou zorazitelné násobky 1, nejmenší zobrazitelné číslo je 0, největší $A_{max} = 11111111_2 = 255_{10}$, počet zobrazených čísel je $N = 256$.

∘

S řádovou mřížkou a aritmetickými operacemi souvisí pojmy *přetečení a podtečení*. Přetečení (overflow) nastane, jestliže je výsledek aritmetické operace větší než největší zobrazitelné číslo v řádové mřížce. Podtečení (underflow) nastane, jestliže je výsledek aritmetické operace menší než nejmenší zobrazitelné číslo v řádové mřížce. Mějme desítkovou soustavu a dvojcifernou řádovou mřížku druhého typu. Sčítáme-li 25+34=59, je vše v pořádku. Máme-li sečíst 75+65=130, došlo k přetečení, protože výsledek je větší než 99, což je největší dekadické číslo zobrazitelné v dané řádové mřížce.

5.7.2 Sčítání ve dvojkové soustavě, dvojková sčítačka

Máme-li sečíst v desítkové soustavě např. $35 + 28$, postupujeme od nejnižšího řádu: nejprve sečteme $5 + 8 = 13$. Do výsledku napíšeme v řádu pouze 3 a jedničku přičteme k vyššímu řádu, tedy: $1 + 2 + 3 = 6$. Přičítané 1 se říká přenos do vyššího řádu. Ve dvojkové soustavě je to obdobné. Začínáme sčítat od nejnižšího řádu a případný přenos připočítáme do vyššího řádu. Sčítání dvojkových cifer vypadá takto:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ přenos do vyššího řádu $= 1$

Součet ukážeme na konkrétním příkladě:

	1	1_2	3_{10}
	1	0_2	2_{10}
přenos $\rightarrow$	$\overset{1}{}$		
	1 0	1_2	5_{10}

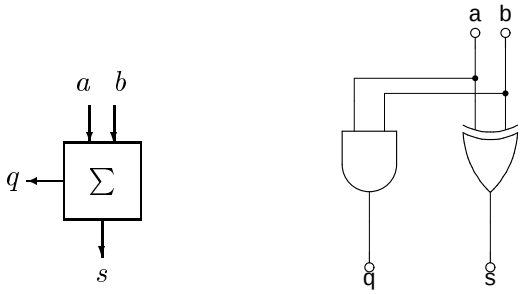
Obvod, který provádí sčítání pro 1 řád, se nazývá jednobitová dvojková poloviční sčítačka (half adder). Je to kombinační obvod se dvěma vstupy a, b, což jsou bity sčítanců, a dvěma výstupy - součtem s a přenosem do vyššího řádu q. Používá se pro součet v nejnižším řádu. Pravdivostní tabulka sčítačky je v tab. 5.1 spolu s rovnicemi. I bez zápisu funkcí do map lze rovnice snadno zapsat.

a	b	s	q
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$s = a \oplus b$$
$$q = a \cdot b$$

Tabulka 5.1: Poloviční dvojková sčítačka - tabulka a rovnice

Používaný symbol pro sčítačku je na obr. 5.11 (a), vnitřní zapojení poloviční sčítačky je na obr. 5.11 (b).



Obrázek 5.11: Jednobitová poloviční sčítačka - schéma

Pro sčítání vyšších řádů se používá úplná jednobitová sčítačka - obvod se třemi vstupy a, b, p a dvěma výstupy s, q. Vstup p je přenos z nižšího řádu, význam ostatních signálů je identický s poloviční sčítačkou. Pravdivostní tabulka úplné sčítačky je zachycena v tab. 5.2 i s rovnicemi.

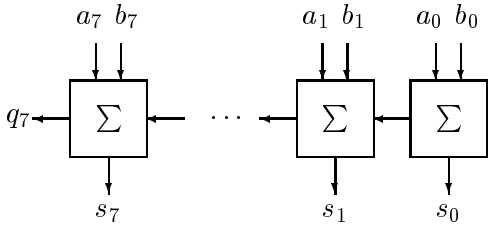
Kaskádním spojením poloviční a úplných sčítaček dostaneme vícebitovou sčítačku (obr. 5.12 pro osmibitovou sčítačku). Výstup přenosu do vyššího řádu p_i u řádu i je přiveden jako vstup

a	b	p	s	q
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{aligned} s &= a \oplus b \oplus p \\ q &= a \cdot b + b \cdot p + a \cdot p \end{aligned}$$

Tabulka 5.2: Úplná dvojková sčítačka - tabulka a rovnice

q_{i+1} přenos z nižšího řádu u řádu $i+1$. Výstup q_7 (přenos z nejvyššího řádu) signalizuje *přetečení*. Dvojková sčítačka je základem aritmetické jednotky každého počítače. V podobě, jak jsme ji nakreslili, je „pomalá“, protože signál přenosu prochází celou kaskádou. Proto ve skutečných procesorech jsou vylepšené sčítačky s predikcí přenosu, uschovávanými přenosy atd., které zrychlují činnost obvodu. Zájemce odkazujeme na literaturu [3].



Obrázek 5.12: Osmibitová binární sčítačka

5.7.3 Zobrazení záporných čísel

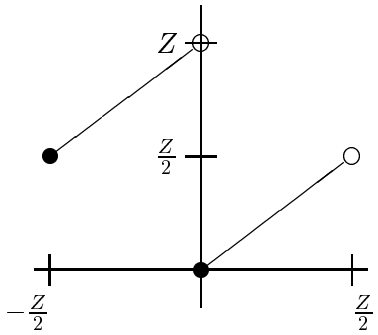
Zápis čísla $A = a_n a_{n-1} \ldots a_0, a_{-1} \ldots a_m$ v soustavě o základu z vyjadřuje hodnotu $a_n z^n + a_{n-1} z^{n-1} + \cdots + a_0 z^0 + a_{-1} z^{-1} + \cdots + a_m z^m$, jak bylo uvedeno v úvodní kapitole. Protože je $z \geq 2$ a $a_i \geq 0$, lze tímto zápisem vyjádřit pouze nezáporná čísla. Ale záporná čísla u reálných výpočtů potřebujeme. Musíme nadefinovat vhodné zobrazení záporných čísel. Čtenáře jistě napadne, že ve dvojkové soustavě bychom se mohli inspirovat zápisem, který používáme v běžném životě, tj. zápisem znaménka. Vyhradíme nejvyšší řád pro znaménko, 0 bude značit kladné číslo, 1 záporné číslo. Zbylé nižší řády určují absolutní hodnotu. Tomuto kódu pro zobrazení záporných čísel říkáme *přímý kód*. U přímého kódu se změní rozsah zobrazitelných čísel, existuje kladná a záporná 0. Lze zobrazit čísla v rozsahu $-(z^n - z^m), \ldots, 0, \ldots, +(z^n - z^m)$ neboli $-(\frac{Z}{2} - \varepsilon), \ldots, 0, \ldots, +(\frac{Z}{2} - \varepsilon)$. Např. pro osmibitový registr (mřížka $n = 7$, $m = 0$) jsou zobrazitelná celá čísla v rozsahu -127 (11111111_2) až +127 (01111111_2), kladná nula má vyjádření 00000000_2, záporná nula 10000000_2, např. číslo -5 je vyjádřeno (10000101_2). Přímý kód je srozumitelný člověku, nehodí se však pro provádění aritmetických operací v počítači. Algoritmus výpočtu by byl stejný, jako když provádí výpočty člověk - porovnáme znaménka, pokud jsou stejná, sečteme, absolutní hodnoty, výsledek má stejné znaménko jako sčítance atd. Proto byl vynalezen tzv. doplňkový kód.

Doplňkový kód

Uvažujme řádovou mřížku se $Z = 1000$, $\varepsilon = 1$, $z = 10$. Vypočítáme rozdíl $729 - 875 = -146$. Nyní vypočteme rozdíl ještě jednou, ale bez ohledu na to, že číslo -875 má větší absolutní hodnotu než 729. Začneme nejnižším řádem a počítáme: „5 a kolik je 9: 4“. Přejdeme k prvému řádu: „7 a kolik je je 12: 5“. Vyjde nám číslo 854, zůstane nám 1 jako přenos. Přičtěme nyní k modulu Z rozdíl -146: 1000-146=854. Vyšlo nám (překvapivě) číslo 854, které je tzv. *doplňkem* čísla -146 do modulu. V doplňkovém kódu jsou kladná čísla zobrazena beze změny, záporná čísla jako doplněk do modulu řádové mřížky. Obraz čísla A v doplňkovém kódu označíme D(A) a bude definován:

$$\mathcal{D}(A) = \begin{cases} A & \text{je-li } A \geq 0, \\ Z + A & \text{je-li } A < 0 \end{cases}$$

Zobrazitelná čísla jsou všechny násobky jednotky řádové mřížky v rozsahu $-\frac{Z}{2} \leq X \leq \frac{Z}{2} - \varepsilon$, obr. 5.13.



Obrázek 5.13: Rozsah zobrazitelných čísel v doplňkovém kódu

Příklad 5.4:
Je dána řádová mřížka s parametry $Z = 1000$, $\varepsilon = 1$, desítková soustava $z = 10$. Najděte obrazy následujících čísel v doplňkovém kódu: $A = 35$, $B = -57$, $C = 725$, $D = -500$
Řešení:
Rozsah zobrazitelných čísel je $-500 \ldots 499$. Obraz kladných čísel je číslo samo, obraz záporných čísel je doplněk do modulu, tedy:

$\mathcal{D}(A) = 35$ $\quad$ $\mathcal{D}(C) =$ není v této řád. mřížce zobrazitelné
$\mathcal{D}(B) = 1000 - 57 = 943$ $\quad$ $\mathcal{D}(D) = 1000 - 500 = 500$

∘

Všechny obrazy záporných čísel mají v doplňkovém kódu v desítkové soustavě v nejvyšším řádu jednu z číslic 5,6,7,8,9, obrazy kladných čísel mají v doplňkovém kódu v desítkové soustavě v nejvyšším řádu jednu z číslic 0,1,2,3,4. Proto bylo obecně zvoleno nejvyšší kladné zobrazitelné číslo $\frac{Z}{2} - \varepsilon$, zde pro desítkovou soustavu 499, aby mohl být obraz $\mathcal{D}(-500) = 500$. Doplňkový kód má pak vlastnost kódu se snadným určením znaménka.

Příklad 5.5:
Je dána řádová mřížka s parametry $Z = 100000000_2 = 256_{10}$, $\varepsilon = 1$, dvojková soustava $z = 2$. Tato řádová mřížka představuje de facto o osmibitový registr. Najděte obrazy následujících čísel v doplňkovém kódu ve dvojkové soustavě: $A = 00010101_2 = 21_{10}$, $B = -00001110_2 = -14_{10}, C = -01000101_2 = -69_{10}$

Řešení:

Rozsah zobrazitelných čísel je $-10000000_2 = -128_{10}$ až $01111111_2 = 127_{10}$. Stejným postupem dostaneme:

$\mathcal{D}(A) = 00010101_2 = 21_{10}$
$\mathcal{D}(B) = 100000000_2 - 00001110_2 = 11110010_2 = 242_{10}$
$\mathcal{D}(C) = 100000000_2 - 01000101_2 = 10111011_2$

∘

Zpětný převod provádíme následovně: je-li obraz $\mathcal{D}(A) < \frac{Z}{2}$, jde o obraz kladného čísla a $A = \mathcal{D}(A)$. Je-li obraz $\mathcal{D}(A) \geq \frac{Z}{2}$, jde o obraz záporného čísla a $A = \mathcal{D}(A) - Z$ (vztah jsme obdrželi z definice doplňkového kódu).

Příklad 5.6:

Je dána řádová mřížka s parametry $Z = 1000,\ \varepsilon = 1$, desítková soustava $z = 10$. Následující hodnoty jsou obrazy čísel v doplňkovém kódu; nalezněte zobrazená čísla: $\mathcal{D}(A) = 352,\ \mathcal{D}(B) = 743,\ \mathcal{D}(C) = 225,\ \mathcal{D}(D) = 684$.

Řešení:

$\mathcal{D}(A) = 352 < \frac{Z}{2} \quad \Rightarrow \quad A = 352$
$\mathcal{D}(B) = 743 \geq \frac{Z}{2} \quad \Rightarrow \quad B = 743 - 1000 = -257$
$\mathcal{D}(C) = 225 < \frac{Z}{2} \quad \Rightarrow \quad C = 225$
$\mathcal{D}(D) = 684 \geq \frac{Z}{2} \quad \Rightarrow \quad D = 684 - 1000 = -316$

∘

Výhoda doplňkového kódu je ve snadném provádění operací sčítání a odčítání. Sčítáme obrazy čísel v doplňkovém kódu bez ohledu na znaménka a zanedbáme přenos z nejvyššího řádu. Zájemce o matematické odvození odkazujeme na literaturu [3]. Vlastnosti ukážeme na příkladu.

Příklad 5.7:

Je dána řádová mřížka $Z = 1000,\ \varepsilon = 1$, desítková soustava $z = 10$ a čísla $A = 325,\ B = 121,\ C = -327,\ D = 15$. Vypočtěte v doplŇkovém kódu $A + B,\ B + C,\ C - D$.

Řešení:

Převedeme čísla do doplňkového kódu. Při výpočtu výrazu $C - D$ si uvědomíme, že odečíst kladné číslo znamená přičíst číslo opačné, budeme tedy přičítat číslo -15 ($C - D = -327 - 15 = -327 + (-15)$). Nejprve určíme obrazy čísel:

$\mathcal{D}(A) = 325,\ \mathcal{D}(B) = 121,\ \mathcal{D}(C) = 673,\ \mathcal{D}(-D) = 985$

Počítáme:

325	121	673
+121	+673	+985
446	794	1 658

$\mathcal{D}(A + B) = 446 \quad \Rightarrow \quad A + B = 446$
$\mathcal{D}(B + C) = 794 \quad \Rightarrow \quad B + C = -206$
$\mathcal{D}(C - D) = 658 \quad \Rightarrow \quad C - D = -342$

Sčítáme obrazy čísel v doplňkovém kódu zcela běžně, zanedbáme přenos z nejvyššího řádu (zde neznamená přetečení), a výsledek je správný v doplňkovém kódu. Detekce přeplnění je obecně obtížná. K přeplnění dojde, sčítáme-li dvě kladná čísla a vyjde obraz záporného čísla nebo sčítáme-li záporná čísla a výsledek je kladné číslo. Ve dvojkové soustavě se pozná přeplnění snáze - je-li různý přenos do a z nejvyššího řádu.

Obrazy záporných čísel v dopňkovém kódu získáme snadno metodou tzv. *inverzních číslic*. Máme-li soustavu o základu z a z-adické číslice z-adické číslice $a = 0, \ldots, z-1$, pak inverní číslice $\tilde{a}$ k číslici a je definována:

$$\tilde{a} = z - 1 - a$$

Dosazením do vztahu sestavíme tabulku inverzních číslic pro desítkovou a dvojkovou soustavu, tab. 5.3

a	0	1	2	3	4	5	6	7	8	9
$\tilde{a}$	9	8	7	6	5	4	3	2	1	0

desítková soustava

a	0	1
$\tilde{a}$	1	0

dvojková soustava

Tabulka 5.3: Tabulka inverzních číslic

Mějme číslo $A > 0$. Je možné dokázat, že obraz opačného čísla v doplňkovém kódu je $\mathcal{D}(-A) = \tilde{A} + \varepsilon$. Věta platí i v rozšířeném znění: máme-li obraz nějakého čísla v doplňkovém kódu, obraz opačného čísla získáme inverzí a přičtením jedničky do nejnižšího řádu, tzv. *horké jedničky*.

Příklad 5.8:
Najděte obrazy opačných čísel v doplňkovém kódu pomocí inv. číslic: $A = 35,\ B = 257$.
Řešení:
$\mathcal{D}(-A) = \tilde{0}\tilde{3}\tilde{5} + 1 = 964 + 1 = 965$
$\mathcal{D}(-B) = \tilde{2}\tilde{5}\tilde{7} + 1 = 742 + 1 = 743$

∘

Výhoda metody inverzních číslic se projeví hlavně ve dvojkové soustavě. Inverzní číslici získáme negací všech bitů čísla. Dvojková sčítačka/odčítačka v doplňkovém kódu, která má sčítat nebo odčítat sčítance A a B v doplňkovém kódu, se příliš neliší od obyčejné sčítačky. V nejnižším řádu má úplnou sčítačku. Jako sčítanec A přivedeme číslo v doplňkovém kódu. Pokud chceme sčítat, přivedeme jako sčítanec B také číslo v dplňkovém kódu, přenos do nejnižšího řádu je nulový. Při odčítání, kdy vlastně přičítáme číslo opačné, přivedeme na B obraz opačného čísla v doplňkovém kódu, který získáme jednoduše negací bitů, a přenos do nejnižšího položíme roven 1 (horká 1).

Kód, který má obraz záporného čísla inverzní číslo, se nazývá *inverzní*. Má podobné vlastnosti jako doplňkový kód. Inverzní kód se např. používá v IP protokolu při výpočtu kontrolního součtu.

Známý je také aditivní kód, neboli kód s posunutou nulou. Obraz všech čísel (kladných i záporných) se získá tak, že se k číslu přičte vhodná konstanta:

$$\mathcal{A}(A) = A + K, \text{ kde } K > 0 \text{ je konstanta}$$

Výhodná volba je $K = \frac{Z}{2}$. Rozsah zobrazitelných čísel je pak $-K \leq X \leq Z - \varepsilon - K,\ X = k\varepsilon$, pro $K = \frac{Z}{2}$ je rozsah stejný jako v doplňkovém kódu $-\frac{Z}{2} \leq X \leq \frac{Z}{2} - \varepsilon,\ X = k\varepsilon$. Obraz nuly je $\mathcal{A}(0) = K$, nula je obrazem čísla: $\mathcal{A}(-K) = 0$.

5.7.4 Posuvy v řádové mřížce

V podkapitole o instrukcích byla zmínka o posuvech dat. Posuvy dat mají blízko k aritmetickým operacím. Uvažujme opět řádovou mřížku 2. typu s modulem $Z = 1000$, $\varepsilon = 1$ a desítkovou soustavou. Vezmeme číslo 030 a posuneme o jeden řád doleva, do nejnižšího řádu vložíme 0; obdržíme číslo 300. Posuneme 030 jeden řád doprava a do nejvyššího řádu vložíme 0: dostane

číslo 003. Posuvem doleva jsme získali desetinásobek původního čísla, posuvem doprava jsme dostali původní číslo dělené 10. Obecně platí, že posuvem čísla A v řádové mřížce o l míst doleva získáme násobek $A \cdot z^l$, posuvem doprava o l míst získáme celočíselný podíl $A \div z^l$. K přeteční dojde, vysune-li se z nejvyššího řádu nenulová číslice; pokud se vysune nenulová číslice z nejnižšího řádu, můžeme říci, že došlo ke ztrátě přesnosti. Tedy, pokud provedeme ve dvojkové soustavě s registem operaci posuvu o 1 místo doleva, vynásobíme uložené číslo 2, pokud provedeme posuv o 2 místa doleva, vynásobíme uložené číslo 4 atd. Analogicky posuvem o jeden bit doprava vydělíme číslo celočíselně 2.

Je možné provádět posuvy i s čísly v doplňkovém kódu, řídí se ale trochu jinými pravidly. Začneme ilustračním příkladem. Uvažujme stejnou řádovou mřížku jako v předešlém odstavci. Číslo $A = -30$ má v doplňkovém kódu obraz $\mathcal{D}(-30) = 970$. Desetinásobek čísla A je -300. Posuňme obraz čísla -30 v dopňkovém kódu o jeden řád vlevo: 700. Snadno nahlédneme, že 700 je v doplňkovém kódu obrazem čísla -300. Obecně, posuv vlevo je možné aplikovat i na čísla v doplňkovém kódu, pro detekci přeteční je třeba porovnávat znaménko původního čísla a výsledku.

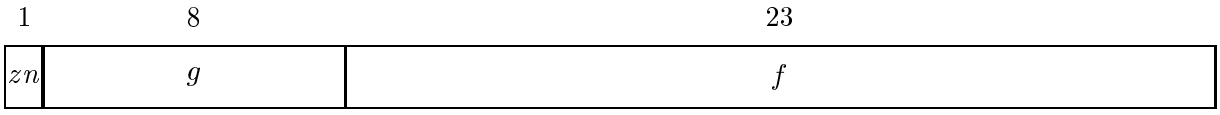
Podíl $\frac{A}{10}$ má hodnotu -3. Obraz čísla -3 v doplňkovém kódu je $\mathcal{D}(-3) = 997$. Číslo 997 bychom pomocí 970 (což je obraz -30) dostali posuvem doprava s tím rozdílem, že do nejvyššího řádu vložíme místo 0 číslici 9. Obecně, v doplňkovém kódu musíme provést posuv doprava tak, že u záporných čísel vložíme do nejvyššího řádu 9.

Pro dvojkovou soustavu a doplňkový kód je posuv doleva identický s „normálním“ posuvem, při posuvu doprava kopírujeme do nejvyššího řádu hodnotu bitu původního nejvyššího řádu (u obrazu kladného čísla zůstane 0, u obrazu záporného čísla zůstane 1). Uvedený posuv se nazývá *aritmetický posuv* a příslušnou instrukci, zvanou `SAR` mají procesory běžně v instrukční sadě.

5.7.5 Zobrazení desetinných čísel

Zobrazení pomocí pevné řádové mřížky (prvého i druhého typu) se nazývá zobrazení čísel *v pevné řádové čárce*. Používá se pro uložení celých čísel (typy `char, int, long` v programovacím jazyce C). Pro uložení desetinných čísel se používá zobrazení *v pohyblivé řádové čárce* (typy `float, double` v programovacím jazyce C, `real` v jazyce Pascal). Úmyslně jsme nepoužili označení reálná čísla, protože reálná čísla, která mohou mít nekonečný rozvoj zlomkové části, uložit do paměti počítače s konečnou velikostí nelze.

Zobrazení desetinných čísel odpovídá *semilogaritmickému tvaru*, který je čtenáři znám z fyzikálních výpočtů, např. $-3,5 \cdot 10^{-3}$. Při stanovení základu soustavy z je desetinné číslo jednoznačně určeno dvojicí *mantisa* m (zde $m = -3,5$) a *exponent* exp (zde $exp = -3$). Obecně zapíšeme číslo v semilogaritmickém tvaru podle naší symboliky $m \cdot z^{exp}$. V počítačích se dnes nejvíce používá formát, který stanovuje norma ANSI/IEEE Std 754 z roku 1985. Formát 32 bitového čísla je na obr. 5.14. Mantisa je zobrazena v přímém kódu, znaménko je uloženo v bitu nejvyššího řádu ($zn = 0$ znamená kladné číslo), mezi znaménko a absolutní hodnotu mantisy je vložen exponent g, který je kódován v kódu s posunutou nulou. Přičítaná konstanta má hodnotu 127 dekadicky ($g = exp + 127$).



Obrázek 5.14: Formát čísla v pohyblivé řádové čárce podle normy ANSI

Víme, že jedno desetinné číslo lze vyjádřit v semilogaritmickém tvaru nekonečně mnoha zápisy, lišící se polohou řádové čárky: $-3,5 \cdot 10^{-3} = -35 \cdot 10^{-4} = -0,35 \cdot 10^{-2}$. Norma ANSI

stanoví, že číslo má být uloženo ve dvojkové soustavě v *normalizovaném tvaru*. Dvojková čárka se posune, aby mantisa byla zapsána dvojkově jako $m = 1, a_{-1}a_{-2}\ldots_2$. Celočíselná část mantisy má tak vždy hodnotu 1. Tato jednička se neukládá (tzv. *skrytá jednička, hidden one*) a uloží se pouze zlomková část mantisy, kterou označíme f (má šířku 23 bitů). Normalizovaný tvar zjednodušuje algoritmy operací s čísly v pohyblivé čárce. Norma ještě stanovuje konvenci pro speciální případy: jsou-li všechny bity nulové, jde o obraz 0. Pokud je $g = 255_{10}$ a $f = 0$, znamená to ∞. Dvojice $g = 255_{10}$ a $f \neq 0$ je interpretována jako NaN (Not a Number), např. výsledek dělení 0:0.

Norma definuje i 64 bitový formát (dvojitá přesnost). Pro znaménko je vyhrazen 1 bit, pro exponent g 11 bitů, pro zlomkovou část mantisy f 52 bitů. Princip skryté jedničky a kód s posunutou nulou je zachován, konstanta je rovna 1023.

Příklad 5.9:

Nalezněte reprezentaci desítkového čísla -5 v pohyblivé řádové čárce podle normy ANSI ve 32 bitovém formátu.

Řešení:

Číslo -5 má ve dvojkové soustavě vyjádření -101_2. V semilogaritmickém tvaru můžeme psát $-101_2 \cdot 2^0$. Posuneme řádovou čárku mantisy, aby byl splněn požadavek normalizovaného tvaru: $-101_2 \cdot 2^0 = -1,01_2 \cdot 2^{(2_{10})}$. Znaménkový bit má hodnotu $zn = 1$. Číslo f je zlomková část mantisy $f = 01000000000000000000000_2$. Exponent je $exp = 2_{10}$. Jeho vyjádření v kódu s posunutou nulou je $g = (2 + 127)_{10} = 129_{10} = 10000001_2$. Číslo -5_{10} má v pohyblivé řádové čárce vyjádření

$$11000000101000000000000000000000_2$$

∘

Příklad 5.10:

Nalezněte, jaké číslo v desítkové soustavě reprezentuje vyjádření v pohyblivé řádové čárce $11000010011010000000000000000000_2$.

Řešení:

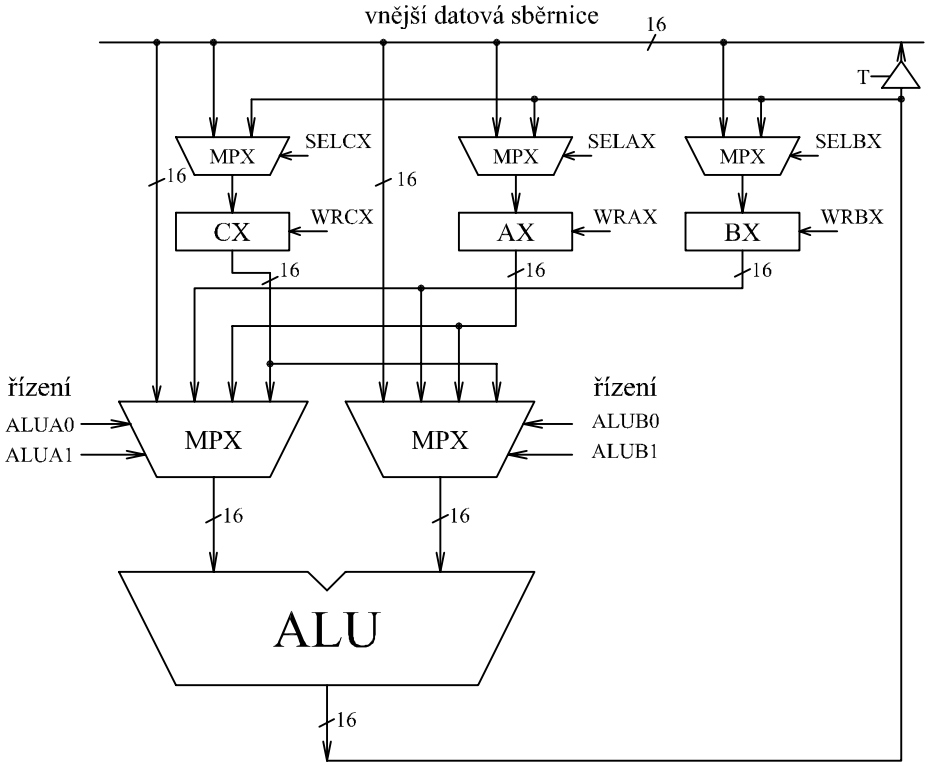
Nejvyšší bit (znaménkový) má hodnotu 1, číslo je záporné. Exponent v kódu s posunutou nulou má hodnotu $g = 10000100_2 = 132_{10}$. Odtud $exp = 132 - 127 = 5_{10}$. Zlomková část mantisy je $f = 1101_2$. Připočítáme skrytou jedničku, tedy $m = 1,1101_2$. Semilogaritmický tvar čísla ve dvojkové soustavě je $-1,1101_2 \cdot 2^5 = -111010_2 = -58$. Zápis reprezentuje číslo -58.

∘

Operace v pohyblivé řádové čárce jsou mnohem náročnější než v pevné čárce. Při sčítání/odčítání je třeba převést čísla do tvaru, aby měly stejné exponenty, a sečíst/odečíst mantisy, výsledek se ještě převede do normalizovaného tvaru. Při násobení se sečtou exponenty a vynásobí mantisy; výsledek se musí převést do normalizovaného tvaru. Bez reprezentace v pohyblivé řádové čárce by nebyly myslitelné vědecko-technické výpočty. Proto je součástí procesorů druhá aritmetická jednotka, zvaná *Floating Point Unit (FPU), jednotka pro výpočty v pohyblivé řádové čárce*, speciálně navržená pro operace v pohyblivé čárce. Součástí jednotky jsou i střadače (registry) pro uložení dat v tomto formátu. Instrukční sada je rozšířena o speciální aritmetické instrukce. V počátcích obvodů LSI nebyla jednotka integrována do procesoru a dodávala se jako samostatný obvod, *matematický koprocesor*. Koprocesor četl instrukce paralelně s hlavním procesorem a vykonával jen instrukce pro FPU. Pokud koprocesor chyběl, vyvolala instrukce typu FPU u procesoru bez jednotky FPU interní přerušení „neznámá instrukce“ a vykonání instrukce se emulovalo softwarově. U firmy Intel měly koprocesory označení 80x87, tj. k procesoru 8086 bylo možné dokoupit koprocesor 8087, k procesoru 80286 existoval koprocesor 80287 atd. Integrovaný koprocesor měl až procesor 80486DX (u 80486 byl deaktivován). Všechny procesory Pentium mají již koprocesor integrován.

5.7.6 Řešení datových cest aritmetické jednotky

U procesorů typu RISC se provádějí operace mezi daty uloženými v registrech, u procesorů CISC, kde existuje mnoho adresových módů instrukcí, se provádějí operace i mezi registry a daty uloženými v hlavní paměti. To znamená, že při aritmetických operacích je potřeba přivést na vstup aritmeticko-logické jednotky data z různých zdrojů. Při návrhu procesoru je nutné věnovat pozornost datovým cestám a jejich řízení z řadiče. Existují podstatě dvě možnosti: *multiplexová architektura* a *sběrnicová architektura*. Příklad multiplexorové architektury datových cest je na obr. 5.15.



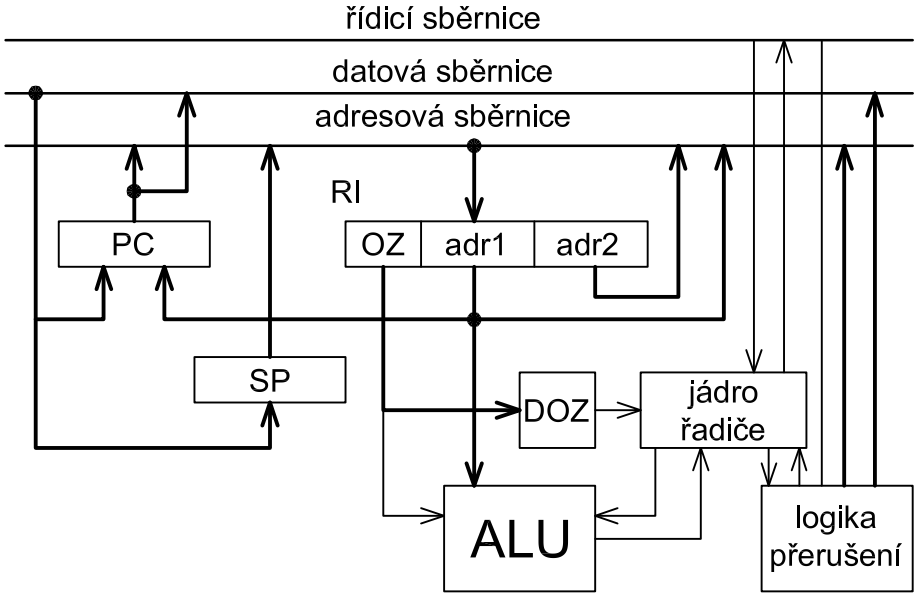
Obrázek 5.15: Multiplexorová architektura datových cest procesoru

Uvažujme 16 bitový procesor typu CISC se třemi střadači AX, BX, CX, které mohou být použity libovolně v aritmetických instrukcích (tzv. ortogonalita), jako operand je dovolena i přímá adresa, tj. zpracovává se číslo z operační paměti. Na každý datový vstup aritmetické jednotky ALU může být přiveden jeden ze čtyř zdrojů dat (tři registry a vnější datová sběrnice). Výběr je proveden multiplexory s řídicími signály ALUAx, ALUBx. Do střadačů mohou být zapsána nová data buď z vnější datové sběrnice (při instrukci typu MOV AX,DATA) nebo výsledek aritmetické operace (instruce typu ADD AX,BX). Zdroj dat se vybírá pomocí dvouvstupových multiplexorů s řídicími signály SELAX, SELBX, SELCX. Zápis do střadačů je povolen signály WRAX, WRBX, WRCX (můžeme je ztotožnit s povolovacími signály klopných obvodů CE). Protože může být výsledek z ALU zapsán i do operační paměti (ADD [adresa],AX), existuje i datová cesta z výstupu ALU na vnější datovou sběrnici. Připojení ke sběrnici je realizováno třístavovým budičem (třetí stav se ovládá signálem T). Všechny řídicí signály jsou ovládány z řadiče podle operačního znaku instrukce a příslušných operandů. Na obr. 5.15 není pro přehlednost nakreslen žádný řídicí signál aritmetické jednotky.

U sběrnicové architektury jsou multiplexory nahrazeny lokální sběrnicí. Zdroje dat jsou vybaveny třístavovými výstupy; namísto řízení multiplexorů se ovládají tyto třístavové výstupy. Doporučujeme čtenáři, aby se pokusil překreslit obr. 5.15 na sběrnicovou architekturu.

5.8 Řadič

Řadič počítače vykonává instrukce a řídí ostatní bloky počítače. Jeho chování je popsáno základním cyklem počítače na obr. 5.4. Vývojový diagram na obrázku připomíná automatový model chování sekvenčního obvodu. Skutečně, jádro řadiče je popsáno automatovým modelem a realizováno sekvenčním obvodem. K řadiči počítače bychom logicky přidružili některé bloky, které jsme popsali dříve, např. registry programový čítač, ukazatel zásobníku, příznakový registr. Příklad architektury řadiče je na obr. 5.16. Datové a adresové cesty jsou kresleny silnými čarami, stavové a řídicí signály tenkými čarami. Z důvodu přehlednosti nejsou všechny řídicí a stavové signály kresleny. Obrázek je inspirován literaturou [3].

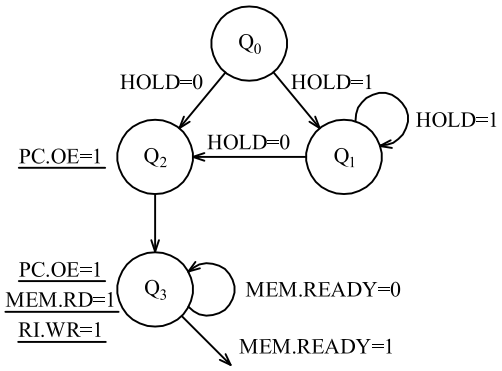


Obrázek 5.16: Architektura řadiče

Kromě známých bloků obsahuje blokové schéma ještě registr instrukce RI, kam se ukládá instrukce po prvním strojovém cyklu. Dále jsou zde dekodér operačního znaku DOZ a logika řízení přerušení. Dekodér operačního znaku může být kombinační obvod a může usnadnit návrh jádra řadiče. Mnoho instrukcí má velmi podobné řízení, např. aritmetické operace, a někdy stačí jako vstupní informace jádru řadiči informace o skupině operací. Proto je také zakreslena cesta od operačního znaku OZ k aritmetické jednotce. Operační znak bývá pomyslně hierarchicky rozdělen právě na typ operace (aritmetická) a operaci (sčítání). Při vhodném kódování operací může být ta část, která určuje operaci, přivedena přímo na vstup aritmetické jednotky a řídit ji přímo. Existence datové cesty od programového čítače na vnější adresovou sběrnici je zřejmá (načítání instrukce). Na datovou sběrnici je potřeba vyslat adresu při instrukcích CALL a přerušení, kdy se ukládá návratová adresa na zásobník. První operand instrukce se ukládá do registru PC, pokud jde o instrukci skoku. Operandy mohou být vyslány na adresovou sběrnici (nepřímý operand) nebo se pomocí nich vybírá registr jako zdroj dat pro aritmetickou operaci (datové registry nejsou kresleny, jsou součástí ALU). Do obrázku se již nevešlo propojení datové sběrnice a ALU.

Jako ukázku automatového popisu jádra řadiče uvedeme část grafu přechodů, který představuje řízení načítání instrukcí z paměti fiktivního počítače. Graf je opět inspirován literaturou [3]. Předpokládejme, že přístup na vnější sběrnici je řízen přidělovačem, procesoru je přidělení sběrnice oznámeno signálem HOLD=0. Výstup programového čítače na datovou sběrnici se povoluje signálem PC.OE=1, čtení z paměti je řízeno signálem MEM.RD=1. Dokončení cyklu signalizuje paměť signálem MEM.READY=1. Zápis instrukce do registru RI povoluje signál RI.WR=1.

Graf má podobu Mooreova automatu, obr. 5.17. Protože nebylo možné zapsat hodnoty výstupních signálů do uzlů grafu, jsou umístěny vně a podtrženy, aby je bylo možné odlišit od podmínek přechodů.



Obrázek 5.17: Část automatového modelu jádra řadiče - načítání instrukce

5.9 Paměti počítačů

Paměti je možné rozdělit podle způsobu výběru informace na paměti s adresním výběrem a paměti s asociativním výběrem. Popíšeme stručně paměti s adresním výběrem.

5.9.1 Paměti RAM a ROM

Podle možnosti čtení a zápisu dělíme paměti na dvě velké skupiny: paměti typu RAM a ROM. Díky technologickému pokroku se kromě zvyšování kapacity postupně objevilo mnoho variant pamětí typu ROM. Odstraňují hlavní nevýhodu původní technologie ROM - nemožnost opakovaného zápisu.

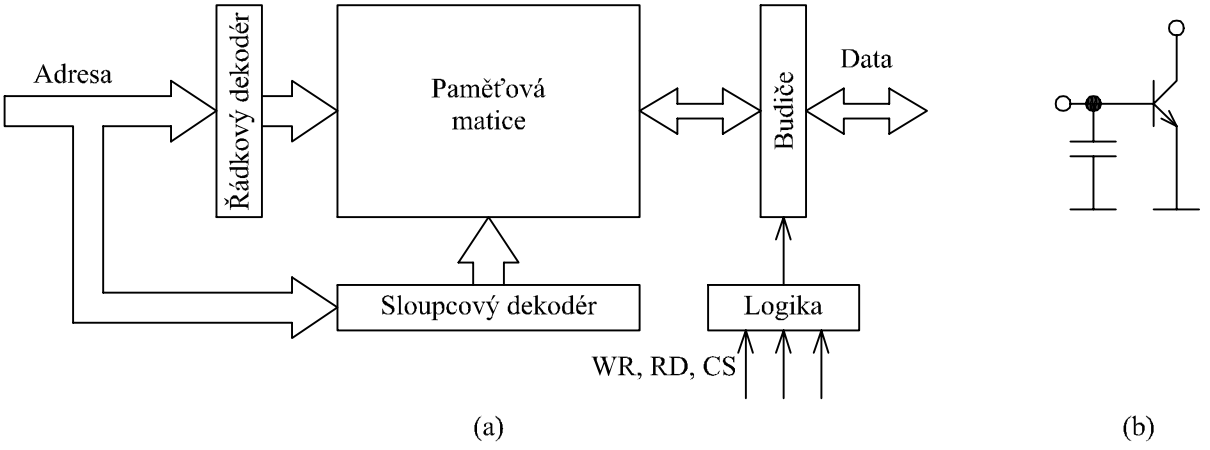
RAM (Random Access Memory) je paměť s „náhodným přístupem“, tj. je možno z ní číst i do ní zapisovat. Bývá označována také zkratkou RWM (Read-Write Memory) - paměť pro čtení i zápis. Její nevýhodou je, že po vypnutí napájení ztrácí svůj obsah. RAM paměti se dělí na *statické* a *dynamické*. U statických pamětí je buňka tvořena bistabilním klopným obvodem se dvěma tranzistory. U dynamických pamětí je buňka tvořena jedním tranzistorem a paměťovým kondenzátorem (obr. 5.18 (b)). Informace je uchována v podobě elekrického náboje na kondenzátoru. U dynamických pamětí lze dosáhnout větší kapacity (realizace 1 tranzistoru v buňce místo 2). Protože se kondenzátor vybíjí, je nutné při provozu paměti náboj pravidelně občerstvovat, tj. přídavná logika musí náboj "doplňovat" na kondenzátor. U statických pamětí není občerstvování třeba, proto se statické paměti používají především v malých a jednoduchých zařízeních ve spojení s jednočipovými mikropočítači.

ROM (Read Only Memory) je paměť pouze pro čtení. Informace zapíše natrvalo výrobce do paměti při výrobě a není možné jí přepsat. Obsah zůstává zaznamenán i po vypnutí napájecího napětí. V počítačích obsahují ROM paměti základní programy (firmware) důležité pro start počítače (u PC je to BIOS). Dalším technologickým pokrokem byla paměť typu *PROM* (Programming ROM). Paměť má stejné vlastnosti jako ROM, ale obsah si může (pouze 1 krát) naprogramovat uživatel (od výrobce je paměť čistá). Děje se tak přiložením vyššího napětí při programování na vývody obvodu - úmyslně se tím zničí tranzistory v buňkách, kde má být zapsána (většinou) log. 0 (analogie s obvody PLA). Programuje se v zařízení zvaném programátor. Paměť se vloží do patice programátoru, ten je připojen k běžnému PC. Obsah se zapíše pomocí ovládacího programu. Pamět typu *EPROM* (Erasable PROM) je mazatelná programovatelná

paměť. Opět, naprogramuje si ji sám uživatel, po vypnutí napájení obsah zůstává. Tentokrát je informace zapsána pomocí tzv. izolace náboje. Paměť má na pouzdru okénko. Chceme-li ji smazat, provedeme to osvícením UV zářením v mazačkách pamětí. Pak je ji možné znovu naprogramovat. Vyrobci garantují spolehlivost dat asi do opakování 10 000 cyklů výmaz/programování cca na 20 let. Pamět typu *EEPROM* (Electric Erasable PROM) je mazatelná programovatelná paměť, tentokrát elektricky. Můžeme do ní zapisovat i číst elektronicky jako do RAM, ale po vypnutí napájení její obsah zůstává. Existuje varianta FLASH, kterou je možné naprogramovat pouze naráz. Na technologii EEPROM jsou založeny obvody CPLD.

5.9.2 Architektura paměti s adresním výběrem

Každý paměťový obvod má adresové vodiče, na které je přivedena adresa buňky, ze které chceme číst nebo do ní zapisovat. Po datových vodičích jsou přenášena zapisovaná/čtená data. Z dalších řídicích signálů jmenujme signál rozlišení čtení/zápisu (RD/WR), jde-li o paměť s možností čtení/zápisu. Většinou má každý čip ještě signál aktivace obvodu (CS = chip select, výběr obvodu). Není-li aktivní, obvod nereaguje na ostatní řídicí signály.



Obrázek 5.18: Struktura paměťových obvodů

Základem každého obvodu je matice paměťových buněk (obr. 5.18 (a)). Struktura buňky závisí na typu paměti (viz výše). Řádkový a sloupcový dekodér vybere podle adresy z matice příslušné buňky. Další logika spolu se zesilovači, budiči podle typu operace čtení/zápisu nasměrují obsah buněk na datové vodiče (čtení), popř. se obsah datových vodičů zapíše do buněk.