

8. PŘÍKLADY ARCHITEKTUR MIKROPOČÍTAČŮ

Mikropočítače první a druhé generace jsou v podstatě integrované verze minipočítačů a proto převzaly většinu prvků jejich architektury. Charakteristickým rysem jejich architektury i organizace je univerzálnost, která sice nepředstavuje zásadní omezení při jejich aplikaci, ale vynucuje si doplnění sestavy mikropočítače množstvím obvodů pro styk s okolím. Soubor instrukcí odpovídá minipočítačům druhé generace, neboť je orientován na řešení úloh obecného charakteru. Nedává možnost efektivního programování úloh probíhajících v reálném čase, který vyžaduje speciální instrukce pro řízení vstupně-výstupních operací a vhodné technické vybavení pro řešení systému přerušení.

Třetí generace mikropočítačů odstraňuje tyto zásadní nedostatky programového a technického vybavení a doplňuje prostředky zvýhodňující vstupně-výstupní operace, operace vyvolání a návratu z přerušování a rozšiřuje soubor aritmetických a logických operací.

8.1 Mikropočítač Intel MCS-48

Základními stavebnicovými prvky mikropočítačů Intel MCS-48 jsou tři typy mikroprocesorů, které jsou elektricky i mechanicky kompatibilní - včetně rozmístění vývodů pouzdra, liší se však typem paměti programu, která je integrována s mikroprocesorem v jediném pouzdru.

Mikroprocesor 8048 obsahuje paměť programu s kapacitou 1K bytů, která je programovaná maskou. Mikroprocesor 8748 obsahuje paměť programu se stejnou kapacitou, jež je však reprogramovatelná a lze ji mazat ultrafialovým světlem. Konečně mikroprocesor 8035 pevnou paměť programu neobsahuje; ve všech ostatních směrech je s předcházejícími kompatibilní.

Mikroprocesory všech tří typů obsahují následující základní subsystémy: procesor, paměť s libovolným výběrem s kapacitou 64 bytů, tři programovatelné osmibitové vstupně-výstupní vazební členy, programovatelný generátor časových prodlev (jehož lze též využít jako čítače událostí), prioritní řadič přerušení a zdroj hodinových signálů.

Na obr. 8.1 je uvedeno blokové schéma mikroprocesoru 8048 znázorňující jeho hlavní prvky a jejich vzájemná propojení. Komunikace mezi bloky se uskutečňuje prostřednictvím jediné vnitřní sběrnice. Rezidentní paměti (t.j. pevná paměť a paměť s libovolným výběrem) jsou adresovány nezávisle.

8.1.1 Funkce jednotlivých bloků

Aritmetická sekce obsahuje základní funkční bloky zpracování dat. Sestává z aritmetické jednotky, střadače, příznaku přenosu a dekodéru instrukcí. Jedním z operandů aritmetické nebo logické operace je vždy obsah střadače; zdrojem druhého operandu může být kterýkoliv zdroj informace připojený na vnitřní sběrnici (registr, paměťové místo nebo vazební člen).

Operační znak uložený v instrukčním registru určuje typ operace aritmetické jednotky, zdroj druhého operandu a příjemce výsledku operace.

Aritmetická jednotka může vykonávat následující operace: sčítání s přenosem a bez přenosu; logické operace součtu, součinu a komparace; inkrementaci, dekrementaci a komplementaci; posun vlevo a vpravo; výměnu čtveřic bitů; dekadickou korekci při zpracování desítkových čísel v úsporném formátu. Obsah střadače lze přesouvat nebo vyměňovat s obsahem registrů nebo paměťových míst. Jak generátor časových prodlev, tak stavové slovo programu jsou přímo přístupné ze střadače.

Paměť programu - rezidentní pevná paměť v mikroprocesorech 8048/8748 má kapacitu 1024 osmibitových slov. Je adresována obsahem čítače instrukcí, který je realizován dvěma čítači, z nichž jeden generuje nižších osm bitů adresy, druhý vyšší čtyři bity. K adresování rezidentní paměti se nepoužívají nejvyšší dva bity adresy, obsažené v čítači instrukcí. Jakmile obsah čítače instrukcí je

menší než 1024, automaticky se adresuje rezidentní paměť. Tři paměťová místa pevné paměti s adresami 0, 3 a 7 jsou rezervována. Aktivní signál na vstupu RESET převádí čítač instrukcí na adresu 0, aktivní signál na vstupu INT na adresu 3, a přetečení generátoru časových prodlev resp. čítače událostí na adresu 7. Na adrese 0 musí být tedy uložena první instrukce programu inicializace, na adrese 3 první instrukce podprogramu obsluhy vnějšího přerušení a na adrese 7 první instrukce podprogramu obsluhy generátoru časových prodlev, resp. čítače událostí. V pevné paměti může být uložen jak program, tak tabulky konstant. Práci s tabulkami usnadňují instrukce MOV_P, popřípadě MOV_P3. Mapa pevné paměti je znázorněna na obr. 8.2.

Paměť dat - rezidentní paměť dat má kapacitu 64 osmibitových slov. Všechny 64 paměťových míst lze adresovat nepřímo pomocí dvou ukazatelů paměti dat, které jsou obsaženy v registrech RO a R1. Kromě toho prvních osm paměťových míst s adresami 0 až 7 tvoří pracovní registry, které mohou být přímo adresovány některými instrukcemi. Vzhledem k snadnému přístupu k těmto paměťovým místům se využívají jako paměť často používaných mezivýsledků. Instrukce DJNZ, která umožňuje v jednom instrukčním cyklu dekrementovat obsah kterékoli z těchto pracovních registrů a současně testovat jeho nulový obsah, dovoluje další aplikaci těchto registrů ve funkci čítačů průchodů programovou smyčkou.

Programovým přepnutím pomocí instrukce SELRB lze zvolit druhý soubor pracovních registrů s ekvivalentními vlastnostmi, který se nalézá na adresách 24 a 31. Tento druhý soubor pracovních registrů se používá buď pro rozšíření pevného souboru, anebo je rezervován pro první úroveň podprogramů obsluhy přerušení, což vytváří z pracovních registrů reentrantní¹⁾ prostředek alespoň pro tuto úroveň. Ošetření obsahu se uskuteční jedinou přepínací instrukcí. V případě, že se tento druhý soubor nepoužije, lze tuto zónu paměti s adresami 24 až 31 využít jako univerzální část paměti. Jelikož registry RO a R1 patří k pracovním registrům, možnost přepínání dvou souborů vytváří další dva ukazatele paměti dat, takže je možné v paměti dat

1) Reentrantní systémový zdroj může být kdykoliv přidělen jinému procesu, který o jeho přidělení žádá a jehož priorita je k tomu opravňuje.

vytvořit čtyři snadno přístupné pracovní oblasti. Jak je zřejmé z obr. 8.3, ve kterém je uvedena mapa paměti dat, je možné pamětovou oblast umístěnou mezi oběma soubory pracovních registrů (s adresami 8 až 23) využít buď jako univerzální oblast uživatele, nebo jako osmiúrovňový zásobník. Tento zásobník je adresován nejnižšími třemi bity registru stavového slova programu PSW. Každý registr zásobníku s délkou 16 bitů je tvořen dvěma paměťovými místy. Do každého registru zásobníku se ukládá obsah čítače instrukcí (12 bitů, z nichž 10 bitů adresuje rezidentní paměť programu a všech 12 bitů se používá pro adresování vnější paměti), a obsah části stavového slova programu (jeho nejvyšší čtyři bity obsahující příznaky popsané v obr. 8.4 a tab. 8.1). Ukazatel zásobníku je nastaven do počátečního stavu 0, které ukazuje na paměťová místa s adresou 8 a 9. Po prvním vyvolání podprogramu instrukcí CALL nebo přerušením se obsah čítače instrukcí a části slova PSW zapíše na tyto adresy, t.j. na adresu zásobníku 0 a obsah ukazatele zásobníku se inkrementuje o jedničku, takže ukazuje na paměťová místa s adresami 10 až 11. Při překročení přípustné úrovně podprogramu, t.j. při vložení více jak osmi podprogramů do sebe, dojde k přetečení zásobníku ze stavu 7 do stavu 0, takže při dalších vyvoláních podprogramů se přepisují postupně nejnižší registry zásobníku novým obsahem, takže dojde ke ztrátě informace. Při návratových instrukcích RET a RETR dochází k opačnému přesunu ze zásobníku do registru procesoru a k dekrementaci obsahu ukazatele.

Jestliže se nepoužívá všech úrovní přerušení, je možné uložit do nevyužitých paměťových míst zásobníku data uživatele.

Generátor časových prodlev - čítač událostí. Osmibitový čítač (na obr. 8.5) může pracovat ve dvou režimech činností předepsaných programem. Jeho obsah se předepisuje pomocí instrukce MOV T, A. Při počáteční inicializaci procesoru signálem RESET se jeho stav nemění. Čítání se zastavuje signálem RESET nebo instrukcí STOP TCNT a zahajuje se instrukcí START CNT. Po dosažení stavu FF přechází do stavu 0 a pokračuje v čítání až do zastavení. Při přechodu ze stavu FF do stavu 0 se nastavuje příznak přetečení generátoru časových prodlev (TIMER FLAG, obr. 8.1) a vyvolá se přerušení. Stav příznaku se může testovat instrukcí podmíněného skoku JFT. Přerušení vyvolané generátorem časových prodlev může být maskováno nezávisle na vnějších

přerušeních instrukcí EN TCNTI a DIS TCNTI. Vyvolání přerušení způsobuje přechod na adresu 7, kde začíná podprogram obsluhy. Jestliže dojde k vnějšímu přerušení obslouží se nejprve vnější požadavek na přerušení podprogramem s počáteční adresou 3. Nezpracovaný požadavek na přerušení generátoru časových prodlev resp. čítače událostí je zaznamenán v paměti příznaku přerušení, takže se rozezná a obslouží okamžitě po ukončení obsluhy vnějšího přerušení. V režimu činnosti jako generátor časových prodlev se na vstup čítače přivádí signál základního oscilátoru, dělený 480. Při nejvyšším možném kmitočtu oscilátoru dochází ke každé inkrementaci po asi 80 μ sec, to znamená každých 20 msec dojde k přetečení přes čítací rozsah 256 a tím k přerušení. Počátečním nastavením čítače lze zvolit časovou prodlevu v rozmezí 80 μ s - 20 ms v krocích po 80 μ s. Při menším požadovaném rozlišení lze pracovat v režimu činnosti čítače událostí a použít vnější hodinový signál ze vstupu T1. V tomto režimu se instrukcí START CNT připojí signál T1 z vnějšího vývodu pouzdra na vstup čítače a při přechodu tohoto signálu ze stavu 1 do stavu 0 dochází k inkrementaci obsahu čítače. Maximální kmitočet čítání je jednou za tři instrukční cykly (t.j. asi 100 KHz). Rozsah čítače lze snadno rozšířit programovými prostředky.

Vstupně výstupní obvody. Technické prostředky pro styk mikroprocesoru s okolím sestává ze tří vstupně-výstupních osmibitových vazebních obvodů a tří testovacích vstupů, jejichž stav lze testovat pomocí instrukcí podmíněných skoků.

Nejdůležitějším vstupně-výstupním vazebním obvodem je vazební obvod BUS vybavený obousměrným budičem a doplněný čtyřmi řídicími a vzorkovacími signály WR, RD, PSEN, ALE. Vazební obvod BUS slouží k obousměrnému styku s pamětmi a přídavnými zařízeními. Význam signálů je zřejmý z tab. 8.2 a obr. 8.6. Vazební člen může pracovat staticky, to znamená, že data zůstávají na sběrnici BUS platná až do další instrukce OUTL nebo instrukce s operandem BUS. V tomto režimu se data zapisují do paměti vstupního budiče instrukcí OUTL a přebírají na interní sběrnici instrukcí INS. Při provádění těchto instrukcí se generují odpovídající výstupní vzorkovací signály WR resp. RD. Ve funkci obousměrného vazebního členu, kdy vstupní a výstupní zařízení jsou mapována jako paměťová místa se používá instrukce MOVX jak pro zápis, tak čtení. Při zápisu jsou platná data během tylové hrany signálu WR, při čtení musí být platná data

během týlové hrany RD.

Vazební obvody 1 a 2 jsou identické. Ve funkci vstupních obvodů nemají paměť. Pro funkci výstupních obvodů jsou vybaveny registrem typu střadače, takže jejich výstupní stav se nemění až do následujícího zápisu. Jejich budiče se označují jako nepravé obousměrné budiče, neboť obousměrnost se dosahuje tím, že výstupní budič má charakter budiče s otevřeným kolektorem a nikoliv třístavového budiče, jak je obvyklé. Jestliže vazební obvod pracuje ve funkci vstupu, musí se jeho paměťový registr pro výstup uvést do stavu FF(H). Pak vstupní signály mohou uvést výstupy s otevřeným kolektorem do předepsaného stavu. Vnitřní struktura nepravého obousměrného výstupu je znázorněna na obr. 8.7. Při manipulaci s jednotlivými bity se nemusí paměťový registr uvádět do stavu 1, ale jeho obsah můžeme chápat jako masku pomocí níž lze maskovat obsah vstupního slova. Vazební obvod 2 má kromě své základní funkce ještě dvě další. Během strojového cyklu převzetí instrukce z vnější paměti programu jsou na jeho čtyřech vývodech P20 až P23 multiplexovány čtyři nejvyšší adresové bity se vstupně-výstupními daty. Stejně vývody slouží též jako rozšíření vnitřní sběrnice při použití vstupně-výstupního expanderu 8243.

Obvody větvení programu - umožňují větvit program na základě vnitřních a vnějších podmínek. Pomocí instrukcí podmíněných skoků lze testovat stavy signálů, příznaků a registrů (tab. 8.3).

Systém přerušování. Obsluha přerušování se zahajuje na základě stavu 0 signálu INT, který je vzorkován v každém strojovém cyklu po dobu aktivního stavu signálu ALE. Po ukončení instrukčního cyklu se zahájí obsluha přerušování uložením obsahu čítače instrukcí a příznakové části stavového slova PSW do zásobníku a zápisem adresy 3 do čítače instrukcí. Na adrese 3 je obvykle instrukce nepodmíněného skoku na adresu první instrukce obsluhy přerušování. Podprogram obsluhy přerušování je ukončen instrukcí RETR, která kromě návratu na adresu přerušovaného programu obnovuje příznakovou část stavového slova.

Systém přerušování je jednoúrovňový, neboť další požadavky na přerušování jsou ignorovány až do ukončení obsluhy přerušování. V druhém strojovém cyklu instrukce RETR se ruší zákaz přerušování. Požadavek na přerušování musí být do tohoto okamžiku zrušen, neboť by ji-

nak vyvolal opětovný vstup do téže rutiny obsluhy. Tento požadavek se řeší u většiny přerušovacích zdrojů zrušením požadavku okamžitě poté, kdy procesor převezme nebo zapíše data z jejich vyrovnávacích pamětí. Jestliže se data při obsluze přerušování nepředávají, potvrzuje procesor dohodnutým stavem zvoleného vstupního signálu přijetí požadavku na přerušování, což vyvolává jeho zrušení.

Stav signálu na vývodu INT může být též testován (tab.8.3) a to především v době, kdy je přerušování zakázáno. Vývod INT je v této době ekvivalentní vývodům T0 a T1. Přerušování se zakazuje buď instrukcí DISI, nebo signálem RESET. Povoluje se instrukcí EN I.

8.1.2 Rozšíření mikropočítače MCS-48

Jestliže technické prostředky samotného mikropočítače nejsou dostatečné, lze je rozšířit následujícím způsobem: paměť programu do kapacity 4K bytů, paměť dat do 320 bytů a počet vstupů-výstupů neomezeně. Přitom lze využít speciální funkce, které vytvářejí doplňkové obvody systému 8080. K rozšíření lze použít v zásadě dvě metody: buď speciální vstupně-výstupní expandér 8243 anebo připojit paměti a přídatná zařízení ke vstupu-výstupu kteréhokoliv vazebního obvodu, neboť všechny tyto vstupy-výstupy mají standardní vlastnosti sběrnice systému 8080.

Rozšíření paměti programu

Styk s programovou pamětí v rozsahu do 1024 se odehrává bez generace vnějších signálů s výjimkou signálu ALE. Při překročení tohoto adresového rozsahu se zahajuje cyklus styku s vnější pamětí, během něhož se odehrává následující posloupnost činností:

- a) obsah dvanáctibitového čítače instrukcí se předá na osmibitový výstup vazebního obvodu 0 a nižší čtyři bity výstupu vazebního obvodu 2;
- b) platnost adresy se potvrdí signálem ALE. Sestupnou hranou tohoto signálu se obvykle vzorkuje adresa do adresovacích registrů vnějších zařízení;
- c) signál PSEN indikuje instrukční cyklus. Používá se jako výběrový signál vnějších zařízení;
- d) vazební obvod 0 (BUS) přechází do vstupního režimu a procesor přijímá osmibitové slovo jako instrukční slovo.

Aktivací signálu EA se procesor chová stejným způsobem i pro

adresy nižší než 1024. Procesor 8035 pracuje v tomto režimu stále, neboť nemá rezidentní paměť programu.

Vyšší adresy než 2047 lze dosáhnout pouze instrukcí pro výběr paměťové zóny 1 (SEL MBL, obr. 8.2 a 8.4), za kterou následuje instrukce JMP nebo CALL. Nejvyšší bit A 11 čítače instrukcí se totiž nemění při normálním inkrementování, ale při každé instrukci větvení se do něj zapisuje příznak paměťové zóny. Obsah tohoto příznaku se řídí instrukcemi SEL MBL a SEL MBO. Kdykoliv před instrukcí větvení lze tedy předepsat přechod do druhé paměťové zóny. Vzhledem k tomu, že se do zásobníku ukládá při vyvolávací instrukci i při přerušení všech dvanáct bitů čítače instrukcí včetně bitu A 11, je možné přecházet při vyvolávání přes hranice paměťových zón, neboť návrat do výchozí paměťové zóny je zajištěn. Při přerušení se pracuje vždy s paměťovou zónou 0, alespoň při vyvolání. Používání instrukcí SEL MBO nebo SEL MBL se v podprogramech obsluhy přerušení nedoporučuje, neboť se sice ovlivní stav příznaku paměťové zóny, ale k přepisu do bitu A 11 během přerušení nedojde, takže by změna jeho stavu mohla nežádoucím způsobem ovlivnit přerušovaný program.

Rozšíření paměti dat

Cyklus styku s vnější pamětí dat lze charakterizovat následující posloupností činností:

- a) Na výstup vazebního obvodu 0 se předává obsah registru R0 resp. R1.
- b) Signál ALE indikuje, že adresa je platná. Sestupná hrana signálu ALE se využívá k zápisu do adresového registru paměti nebo vnějšího zařízení.
- c) Signál WR nebo RD označuje druh cyklu. Výstupní data jsou platná při sestupné hraně WR, vstupní data musí být platná při sestupné hraně RD.
- d) Data se přesunou buď směrem z paměti nebo do paměti prostřednictvím vazebního obvodu 0.

Styk s vnější pamětí dat se uskutečňuje na základě instrukcí MOVX A, @R nebo MOVX @R, A, které přesouvají data mezi střadačem a paměťovým místem vnější paměti. Toto paměťové místo je určeno ukazatelem RO, resp. R1, což dovoluje adresovat 256 paměťových míst. K 64 paměťovým místům rezidentní paměti dat lze tak doplnit 256 paměťových míst vnější paměti dat.

Rozšíření vstupů-výstupů

Rozšíření vstupů-výstupů je možné třemi metodami:
 pomocí expandéru 8243
 pomocí standardních vstupně-výstupních obvodů systému MCS-80
 pomocí kombinovaných obvodů paměť - expandér typu 8155, 8355 a 8755.

a) Pomocí expandéru

Expandér 8243 komunikuje s procesory řady MCS-48 pomocí čtyřbitové sběrnice, připojené na nižší čtveřici bitů vstupu-výstupu vazebního obvodu 2. Expandér 8243 obsahuje čtyři čtyřbitové vazební obvody, které slouží jako rozšíření rezidentních vazebních obvodů 0, 1 a 2 a které jsou adresovány adresami 4 až 7. Tyto vazební obvody dovolují provádět následující operace: přesun ze střadače do vazebního obvodu a naopak a logický součet nebo součin obsahu registru vazebního obvodu s obsahem bitů 0 - 3 střadače. Při přesunu z vazebního obvodu do střadače se do nejvyšších čtyř bitů střadače zapisuje nula. Uspořádání technických prostředků a časový diagram přenosu je uveden na obr. 8.8. Při každém přenosu se přenáší dvě čtyřbitová slova. V prvním řídicím slovu se předává adresa výstupního vazebního členu expandéru 8243 a operační kód předepisující typ přenosu. Význam řídicího slova je uveden v tab. 8.4. Řídicí slovo má platný obsah při sestupné hraně řídicího signálu PROG, kdy jej přebírá expandér. Druhé slovo je datové. Přenášená data musí být platná v okamžiku nástupní hrany signálu PROG.

b) Pomocí standardních vstupně-výstupních obvodů systému MCS-80
 Standardní vstupně-výstupní obvody systému MCS-80 mohou být připojeny prostřednictvím stejných technických a programových prostředků, které se používají k připojení vnější paměti dat. K nejužívanějším patří obvody 8255, 8251, 8253, 8259, 8279, se kterými jsme se seznámili v předchozích kapitolách.

c) Pomocí kombinovaných obvodů pamět-expandér

Kombinované obvody pamět-expandér zvyšují flexibilitu technických prostředků mikropočítačů řad MCS-80, MCS-85 a MCS-48 a umožňují vytvářet sestavy mikropočítačů splňující požadavky uživatele s minimálním počtem integrovaných obvodů. Jedním z představitelů těchto kombinovaných obvodů je obvod 8355, obsahující vazební obvody a pevnou pamět programovatelnou maskou, a jeho ekvivalent, obvod 8755, odlišující se od obvodu 8355 pouze tím, že pevná pamět je reprogramovatelná. Kapacita paměti je v obou případech 2048 osmibitových slov. Vazební obvody jsou dva se šíří toku 8 bitů. Jsou adresovány jako pamětová místa a jsou programově přístupné pomocí instrukce MOVX. Ke každému vazebnímu obvodu náleží jeden registr směru přenosu, který představuje další pamětové místo a jehož obsah definuje každý z jednobitových datových vývodů vazebního obvodu buď jako vstup nebo výstup. Obsah registru přenosu lze předepsat programem.

Dalším představitelem kombinovaných obvodů je obvod 8155, který obsahuje kromě paměti s libovolným výběrem a kapacitou 256 osmibitových slov tři vazební obvody: dva osmibitové a jeden šestibitový. Jejich činnost je řízena prostřednictvím řídicího registru, který plní funkci jak řídicího, tak stavového registru. Vazební obvody a řídicí registr představují z hlediska programátora pamětová místa přístupná pomocí instrukce MOVX. Obsah řídicího registru dovoluje předepsat čtyři pracovní režimy vazebních obvodů; základní vstupní a výstupní režim a vstupní a výstupní režim doplněný signály řízení asynchronního přenosu. V režimech s těmito řídicími signály je zdrojem a příjemcem řídicích signálů vazební šestibitový obvod. Obvod 8155 obsahuje též čtrnáctibitový programovatelný generátor časových prodlev. Jak jeho číací vstup, tak i výstup indikace přetečení jsou vyvedeny na zvláštní vývod pouzdra. Programem lze předepsat několik režimů jeho činnosti, např. zastavení při přetečení, trvalé čítání atd.

Příklad rozšíření mikroprocesoru řady MCS-48 pomocí kombinovaných obvodů paměť-expandér je uveden v obr. 8.9. Hlavním problémem je adresování jednotlivých obvodů. Pro výběrový vstup CE obvodu 8355 resp. 8755 se logicky sčítají adresové signály A 10 a A 11, čímž se zajistí přístup k této paměti pro adresový rozsah 1024 až 3072. Tím se vytvoří překrytí adres při přístupu k vazebním obvodům obvodu 8355/8755 a 8155, neboť výběrové vstupy obvodu 8155 jsou spojeny s adresovými signály A 8 a A 9. Předtím, než se vyvolá styk s pamětí s libovolným výběrem anebo s vazebními obvody obvodů 8355/8755 nebo 8155, musí se předepsat adresa do vazebního obvodu 2 podle tab. 8.5.

8.1.3 Operační kód

Operační kód mikropočítače obsahuje 96 instrukcí. Více než 70 % instrukcí je jednoslovných a více než 50 % instrukcí se uskutečňuje v jediném strojovém cyklu. Přehled instrukcí je uveden v tab. 8.6.

V instrukcích se používají tři základní adresovací metody: registrové adresování, nepřímé registrové adresování a okamžité adresování. Registrovým adresováním se adresuje jeden ze souborů osmi pracovních registrů vyznačených na obr. 8.3. Adresa registru se nezapisuje jako skutečná adresa v paměti dat, ale jako pořadové číslo registru v souboru, např. R 6. Zápis první střadačové instrukce v tab. 8.6 je např. následující:

ADD A, R 6 ; přičti obsah registru R 6 k obsahu střadače A
Soubor pracovních registrů se zvolí zápisem řídící instrukce SEL RBØ nebo SEL RB1, která musí této instrukci předcházet.

Nepřímé adresování je omezeno na rezidentní paměť dat, přičemž nepřímou adresu mohou obsahovat pouze první dva registry souboru RØ a R1. Zápis druhé střadačové instrukce z tab. 8.6 je např. následující:

ADD A, @ R 1 ; přičti obsah paměťového místa, adresovaného obsahem registru R1 k obsahu střadače A.

Volba souboru registrů je stejná jako v předchozím případě.

Okamžité adresování se uskutečňuje zápisem přímého operandu

do instrukce, která je v tomto případě dvoubytová. Např.:

ADD A, # OAFH ; přiřti konstantu OAF (šestnáctkově) k obsahu
střadače.

Hodnotu konstanty lze napsat šestnáctkově, desítkově, osmičkově, dvojkově i symbolickým jménem; stejně jako ve zdrojových jazycích symbolických adres MCS-80 a MCS-85.

Ve vstupně-výstupních instrukcích se identifikátor vazebního členu zapisuje pomocí pořadového čísla. První tři instrukce mohou používat pouze identifikátor P1 a P2, poslední čtyři pouze identifikátory P4 až P7, což jsou vazební členy expanderu 8243. Např.:

ANL P1, # CONST ; proved' logický součin obsahu vazebního členu
P1 a konstanty se symbolickým jménem CONST.

Konstanta musí být předem definována pseudoinstrukcí.

Vstupně-výstupní instrukce dovolují, kromě přesunů obsahu vazebního členu do střadače a naopak, uskutečňovat logické operace maskování s maskou uloženou ve střadači (vazební členy P4 až P7) nebo definované přímým operandem (vazební členy P1 a P2).

Z přenosových instrukcí si povšimneme možnosti převzít stavové slovo do střadače a naopak. Tyto instrukce dovolují např. měnit adresu zásobníku anebo podmíněné skoky na základě některého z bitů stavového slova, např. BS, pomocí instrukce JBB. Instrukce MOVX umožňuje pracovat s obsahem vnější paměti dat. Instrukce MOVP a MOVP3 dovolují pracovat s konstantami uloženými v paměti programu. Vzhledem k tomu, že obsahem střadače lze adresovat jen 256 pamětových míst, lze tento rozsah označit jako stránku. V instrukci MOVP se obsah střadače zapisuje do nejnižších osmi bitů čítače instrukcí; jeho vyšší bity se nemění. Adresuje se tedy v rozsahu běžné stránky, určené obsahem vyšších bitů čítače instrukcí. V instrukci MOVP3 se do nejvyšších třech bitů čítače instrukcí zapisuje 011, takže obsah střadače představuje adresu stránky 3. Tato stránka může tedy sloužit jako tabulka obecných konstant, např. tabulka pro konverzi kódů. Po provedení instrukce MOVP, resp. MOVP3 se obnovuje původní obsah čítače instrukcí. Tyto dvě instrukce reprezentují další adresovací metodu operačního kódu, a to stránkové adresování. Příklad použití je následující:

```
MOV A, #128 ;
```

```
MOV A, @A ;
```

Tato posloupnost instrukcí přesune do střadače konstantu uloženou na 129 paměťovém místě běžné stránky.

Ze skokových instrukcí si povšimněme nejdříve nepodmíněných skoků. První je instrukce JMP. Operand této instrukce může být adresa 0 až 2048. Předcházející instrukce SEL MBO, resp. SEL MBL určuje dvanáctý bit čítače instrukcí a tím paměťovou zónu, v jejímž rozsahu se skok uskuteční (obr. 8.2). Všechny ostatní skokové instrukce, včetně nepřímého nepodmíněného skoku JMPP @A se uskutečňují v rámci běžné stránky, t.j. jsou stránkově adresovány. Instrukce DJNZ, která dekrementuje zvolený registr a na základě jeho nenulového obsahu provádí skok, je určena k vytváření programových smyček, přičemž registr R, uvedený v operandu slouží jako čítač průchodů. Při inicializaci smyčky se předepíše jeho obsah např. instrukcí MOV R, # NUMB, kde NUMB je požadovaný počet průchodů smyčkou.

Povšimněme si ještě instrukcí JTO, JNT0, JT1, JNT1, které dovolují větvit program přímo na základě stavu vnějších signálů T0 (resp. T1) bez přenosových instrukcí. Podobně instrukce Jb dovoluje větvit program přímo na základě stavu některého bitu střadače, takže odpadají maskovací operace nutné např. v mikropočítačích MCS-80. Např. instrukce:

```
JB5 NEXT ; předepíše skok na adresu NEXT, jestliže stav  
bitu 5 střadače je nenulový.
```

Z instrukcí pro práci s podprogramy je instrukce CALL podobně jako JMP adresována v rámci paměťové zóny zadané předtím instrukcí SEL MBO, resp. SEL MBL. Návrátové instrukce však vracejí činnost programu do paměťové zóny, ze které byl podprogram vyvolán.

8.2 Mikropočítač Zilog Z-80

Mikropočítač Zilog Z-80 je odvozen z mikropočítače Intel 8080. Jeho operační kód je obohacen o instrukce, které usnadňují programování úloh probíhajících v reálném čase. Jedná se především o instrukce umožňující manipulaci s jednotlivými bity, rozšíření vstupně-výstupních instrukcí a instrukce pro práci se zónami dat. Ope-

rační kód mikropočítače Zilog Z 8 obsahuje všech 78 instrukcí operačního kódu mikropočítače Intel 8080 a navíc má dalších 80 instrukcí. Technické vybavení mikropočítače zvyšuje jeho funkční možnosti. K významným zlepšením patří jediné napájecí napětí + 5V. Samotný mikroprocesor obsahuje mnohé obvody, které v systému 8080 jsou realizovány samostatnými stavebnicovými prvky. Minimální sestavu systému tvoří mikroprocesor, generátor hodinových impulsů, paměť programu a paměť dat se vstupně-výstupními obvody. K rozšíření systému jsou k dispozici další stavebnicové prvky: generátor časových prodlev a čítač událostí, vstupně-výstupní adaptor, řadič přímého styku s pamětí a programovatelný komunikační adaptor.

8.2.1 Mikroprocesor Zilog Z-80

Mikroprocesor obsahuje obvody generující signály pro řízení sběrnic a pamětí. Dále obsahuje časovací signály, které jsou shodné se signály systému Intel 8080; osm univerzálních šestnáctibitových registrů a aritmetickou jednotku s řídicími obvody. Ve vnitřní struktuře nalezneme registry mikroprocesoru Intel 8080 t.j. registry A, B, C, D, E, H, L, PC, SP. Soubor univerzálních registrů je však zdvojen, takže struktura obsahuje registry A', B', D', E', H', L' a navíc dva šestnáctibitové indexové registry IX, IY, osmibitový registr vektoru přerušení I a osmibitový registr R patřící k systému obnovení informace dynamických pamětí. Všechny instrukce se vztahují k základnímu souboru univerzálních registrů. Přístup k druhému pomocnému souboru se uskutečňuje prostřednictvím instrukcí pro výměnu obsahu těchto dvou souborů. Jedna z instrukcí vyměňuje obsah dvojice registrů PSW, druhá obsahy zbývajících šesti dvojic registrů. Tyto dvě jednobytové instrukce mohou být uskutečněny ve čtyřech strojových cyklech, t.j. v době kratší než 1 μ sec, takže představují výhodné řešení pro jednoúrovňové přerušení, které lze pak uskutečnit velmi rychle.

Indexové registry umožňují rozšířit operační kód o instrukce s indexovým adresováním, které má např. mikropočítač Motorola M 6800. Indexové adresování se výhodně uplatní při práci s tabulkami dat a při předávání parametrů podprogramů. Vzhledem k tomu, že pole posunutí je interpretováno jako číslo v druhém doplňku, je možné indexovat kladné i záporné.

Technické vybavení mikroprocesoru Zilog Z-80 zajišťuje obnovování informace dynamických pamětí. Zdrojem adresy pro obnovovací cyklus je registr R, jehož obsah se inkrementuje při každém instrukčním cyklu a předává vždy po převzetí instrukce do nejnižších sedmi bitů adresové sběrnice. Stav řídicí sběrnice signalizuje platný cyklus obnovení, který probíhá v době, kdy mikroprocesor dekóduje převzatou instrukci, takže nenarušuje instrukční cyklus.

Obsah registru I se využívá při přerušení, neboť v režimu vektorového přerušení tvoří vyšších osm bitů adresy, zatímco nižších osm bitů generuje zdroj přerušení. Takto vytvořená nepřímá adresa je operandem vyvolávací instrukce vyvolané přerušením. Ukazuje na paměťové místo, ve kterém je uložena adresa první instrukce podprogramu obsluhy přerušení.

Signály na vývodech pouzdra mikroprocesoru jsou patrné z obr. 8.10. Lze je rozdělit do tří sběrnic: adresovací, datové a řídicí. Řídicí signály se dále dělí do tří skupin. Význam signálů je vysvětlen v tab. 8.7. Systém přerušení mikroprocesoru Z-80 má tři režimy činnosti, které se volí programem. V režimu 0 předává zdroj přerušení libovolnou instrukci operačního kódu, kterou mikroprocesor vykoná. V režimu 1 přechází mikroprocesor vždy na adresu 0038, na které zahajuje podprogram obsluhy. V režimu 2 t.j. režimu vektorového přerušení, se přerušení zahajuje nepřímým vyvoláním.

Každý z registrů stavového slova F obsahuje příznaky, shodné s příznaky používanými ve stavovém slovu mikroprocesoru Intel 8080 - CARRY, ZERO, SIGN, PARITY, AUXILIARY CARRY. Navíc obsahuje příznak SUBTRACT, který dovoluje rozlišit předcházející operaci a upřesnit desítkovou korekci při zpracování zobrazených čísel v BCD kódu v úsporném formátu.

Operace posunu se na rozdíl od mikropočítače Intel 8080 mohou provádět nejen s obsahem střadače, ale s obsahem kteréhokoliv paměťového místa. Vstupně-výstupní operace se provádějí s obsahem kteréhokoliv registru, nikoliv jenom s obsahem střadače. Přímou lze adresovat zaváděcí i ukládací instrukce nejen obsahem dvojice registrů H, ale též B, D, IX, IY. Tím se omezí počet instrukcí v programu. Rovněž aritmetické operace s dvojnásobnou délkou slova, používající ve funkci střadače dvojice registrů H, jsou rozšířeny a zahrnují sečtení s přenosem a odečtení s výpůjčkou.

8.2.2 Operační kód

Soubor instrukcí, které obsahuje operační kód mikropočítače Z-80, lze rozdělit do sedmi základních skupin. Kromě adresovacích metod, použitých v instrukcích systému MCS-80, t.j. registrového, neúplného stránkového, okamžitého a registrového nepřímého adresování, používá operační kód Z-80 relativní a indexové adresování a adresování vztažené k bitům.

V tab. 8.8 jsou uvedeny některé vybrané instrukce operačního kódu mikropočítače Z-80, na kterých lze demonstrovat rozdíly proti operačnímu kódu systému MCS-80. V první skupině instrukcí je kromě dvou instrukcí odpovídajících kódu MCS-80 uvedena další instrukce používající indexové adresování jednoho z operandů. Z druhé skupiny instrukcí si povšimneme instrukce umožňující použít přímé adresování k přesunu do kterékoliv dvojice registrů. V mikropočítači MCS-80 má podobná instrukce LHLD jediného implicitně adresovaného příjemce informace - dvojici registrů HL. Ve skupině výměnných přenosových a prohlédávacích instrukcí uvádíme jednu z instrukcí používaných k výměně obsahů souborů univerzálních registrů. Instrukce LDI je orientována na výměnu obsahů paměťových míst z jedné paměťové zóny adresované obsahem dvojice registrů HL do druhé paměťové zóny adresované obsahem dvojice registrů DE. Instrukce LDIR používá obsahu dvojice registrů BC jako čítače délky bloku a provádí instrukci opakovaně až do ukončení přenosu celého bloku dat, jehož délka je zadána do dvojice registrů BC předem. Podobně dovolují instrukce prohlédávání CPI a CPID vyhledat v zadané paměťové zóně zvolený znak předem zadaný ve střadači. Z řídicích instrukcí je uvedena jedna z instrukcí předepisujících režim přerušení. Instrukce posuvu RLC není omezena jen na střadač, ale lze ji aplikovat na libovolné paměťové místo. Instrukce pro práci s jednotlivými bity umožňují buď testovat zvolený bit zvoleného registru nebo paměťového místa, adresovaného obsahem registru HL i indexovým adresováním nebo jej nastavit do stavu 1 nebo 0.

Vstupně-výstupní instrukce mohou přesunovat data z libovolného a do libovolného univerzálního registru. Instrukce INI, INIRD, INDR a v tabulce neuvedené OUTI, OUTD a OTDR jsou orientovány na přenosy datových bloků.

Mikropočítač Zilog Z-80 patří svým programovým a technickým vybavením mezi špičkové osmibitové mikropočítače. Jeho velkou předností je programová kompatibilita s nejrozšířenějším mikropočítačem Intel MCS-80.

8.3 Mikropočítač Intel MCS-86

Mikropočítač Intel MCS-86 patří ke třetí generaci šestnáctibitových mikropočítačů. Přestože se jedná o unipolární mikropočítač, je svými parametry srovnatelný s bipolárními mikropočítači se stavebnicovou délkou slova a asi o řád předčí osmibitové mikropočítače MCS-80 a MCS-85.

K jeho hlavním přednostem patří rozšíření operačního kódu o aritmetické operace s operandy s délkou osm i šestnáct bitů včetně násobení a dělení, o operace s textovými řetězci a operace orientované na práci s jednotlivými bity. Tyto přednosti se uplatní i při řešení úloh jednoduššího charakteru. Při řešení složitějších úloh, kdy je nutné implementovat operační systém, se uplatní další výhodné prvky jeho architektury. Orientace na báze registry umožňuje snadnou implementaci dynamicky relokovatelných programů jejichž strojový kód je nezávislý na alokaci. Značnou výhodu představuje v těchto případech i možnost přímého adresování operační paměti v adresovacím rozsahu 1 MByte.

Kompatibilita s operačními kódy mikropočítačů MCS-80 a MCS-48 se zajišťuje zvláštním překladačem, takže programové vybavení vyvinuté pro tyto mikropočítače lze snadno převést do strojového kódu mikropočítače MCS-86.

Základní uspořádání mikroprocesoru 8086 je znázorněno na obr. 8.11. Sestává ze dvou základních jednotek: jednotky styku se sběrnicí a výkonné jednotky.

8.3.1 Jednotka styku se sběrnicí

Jednotka obsahuje především soubor čtyř segmentových registrů, které spolu s adresovými částmi instrukcí umožňují adresovat celý adresový prostor. Tento adresový prostor má dvě složky: adresový prostor operační paměti s adresovým rozsahem 1 MByte a adresový prostor vnějších zařízení s adresovým rozsahem 64 KByte. In-

strukce a jejich operandy se nalézají v adresovém prostoru operační paměti. Adresy vnějších zařízení s výjimkou zařízení, která jsou interpretována jako paměťová místa operační paměti, jsou umístěny v adresovém prostoru vnějších zařízení.

Adresovaným prvkem může být buď byte nebo šestnáctibitové slovo. Slovo může být uloženo v jakýchkoliv dvou paměťových místech, jejichž adresy bezprostředně následují. Počáteční adresa může být buď lichá nebo sudá. Převzetí prvku adresovaného lichou adresou vyžaduje dva cykly operační paměti. Je-li však adresován sudou adresou, postačuje jeden cyklus.

Vzhledem k tomu, že aritmetická jednotka procesoru zpracovává pouze šestnáctibitová slova a výpočet skutečné adresy vyžaduje zpracování dvacetibitového slova a též vzhledem k tomu, že jednotka styku se sběrnici pracuje do jisté míry nezávisle na výkonné jednotce, je jednotka styku se sběrnici vybavena sčítačkou pro výpočet skutečné adresy. Adresový prostor operační paměti může být rozdělen na libovolný počet segmentů, z nichž každý má adresový rozsah nejvýše 64 KByte. Počáteční adresa segmentu musí být beze zbytku dělitelná šestnáctí, t.j. nejnižší čtyři bity počáteční adresy jsou nulové. Čtyři z těchto segmentů, jejichž počáteční adresy jsou uloženy v segmentových registrech, jsou bezprostředně adresovatelné. Nazývají se pak běžnými segmenty. Nemusí tvořit spojitý adresový prostor a mohou se též překrývat. Jestliže adresy segmentů splynou a jsou rovny nule, pak adresový prostor degeneruje na adresový prostor mikropočítače MCS-80.

Skutečná adresa fyzického adresového prostoru operační paměti se vypočte jako součet adresy segmentu s adresovou částí instrukce - posunutím (obr. 8.12). Adresa segmentu se vytvoří doplněním obsahu některého ze segmentových registrů čtyřmi bity s nulovým obsahem.

Z dalšího obr. 8.13, ve kterém jsou znázorněny programátoru přístupné registry je zřejmé, že jednotlivým segmentovým registrům jsou přidělena symbolická jména odpovídající jejich využití.

Obsah segmentového registru CS definuje běžný segment programu. K jeho obsahu jsou vztaženy všechny paměťové reference při převzetí instrukce, přičemž obsah čítače instrukcí je interpreto-

ván jako posunutí.

Obsah registru DS definuje segment dat. Všechny paměťové reference na data, výjimkou těch, které se vztahují k ukazatelům zásobníku SP nebo báze BP, jsou implicitně vztaženy k obsahu registru DS. To zahrnuje všechny operace se zásobníkovou pamětí, vyvolávané jak instrukcemi PUSH a POP tak instrukcemi pro práci s podprogramy CALL a RET tak i přerušením.

Pokud však instrukci předchází speciální byte, t.zv. prefix, pak se ruší implicitně definovaný vztah paměťové reference k registru DS nebo SS a prefix může definovat referenci vůči kterémukoliv ze zbývajících tří segmentových registrů. Obsah registru ES definuje speciální segment, který nemá předem dané užití, takže není implicitně v instrukcích definován a používá se jako další segment pro data s explicitní paměťovou referencí pomocí prefixu.

Dynamicky relokovatelné programy nemění obsah segmentových registrů. Instrukce, umožňující změnu obsahu segmentových registrů patří v tomto případě mezi privilegované instrukce operačního systému. Dynamicky relokovatelný program má vlastní virtuální adresový prostor, vztažený většinou k adrese 0. Naplněním segmentových registrů vhodnými konstantami přiřazuje operační systém virtuálním adresovým prostorům jednotlivých programů různé části společného fyzického prostoru.

Možnost dynamické relokače programů řeší problém reentrantnosti operační paměti, kterou je možno pak kdykoliv odebrat kterémukoliv probíhajícímu procesu a přidělit jinému, neboť lze snadno vyměnit část obsahu operační paměti a vnější paměti a přidělit žádajícímu procesu s vyšší prioritou požadovaný rozsah operační paměti.

Další součástí jednotky styku se sběrnici je vyrovnávací paměť se šesti paměťovými místy, sloužící jako paměť převzatých instrukcí. Jednotka styku se sběrnici přebírá nezávisle na výkoné jednotce instrukce z operační paměti a řadí je do fronty předem převzatých instrukcí. Tato technika používaná u větších počítačů umožňuje překrývání fází výběru a převzetí instrukce, neboť jednotka styku se sběrnici využívá dob, ve kterých výkonná jednotka ani jiný systémový prostředek nevyžaduje styk s operační pamětí,

k převzetí instrukcí a zvyšuje tak využití operační paměti. Výkonná jednotka přebírá instrukce připravené ve frontě a provádí je dostatečně rychle, takže nevznikají ztrátové časy při převzetí instrukcí. Technika překrývání zvyšuje rychlost výpočtu a snižuje nepříznivý vliv blokových přenosů na činnost výkonné jednotky.

8.3.2 Výkonná jednotka obsahuje všechny programátoru přístupné registry (obr. 8.13) s výjimkou čítače instrukcí a segmentových registrů, které patří k jednotce styku se sběrnicí. Známý soubor registrů mikroprocesoru 8080 tvoří podmnožinu tohoto souboru. Odpovídající si registry však nenesou stejná symbolická jména, neboť se liší užitím.

Soubor univerzálních registrů AX, BX, CX, DX odpovídá dvojicím registrů A, B, D, H mikroprocesoru 8080 s tím rozdílem, že pro registr stavového slova je vytvořen zvláštní registr příznaků. Obsah kteréhokoliv ze souboru univerzálních registrů mikroprocesoru 8086 může vystupovat jako operand v aritmetických nebo logických operacích. Jsou adresovatelné podobně jako paměťová místa. Adresovatelným prvkem je buď byte nebo šestnáctibitové slovo, takže je lze interpretovat buď jako soubor čtyř šestnáctibitových nebo osmi osmibitových registrů.

Soubor ukazatelů a indexových registrů obsahuje dva ukazatele SP a BP a dva indexové registry SI a DI. Obsah těchto registrů má význam posunutí a využívá se k určení adresy v rámci segmentu.

Rozdíl mezi dvojicemi těchto registrů spočívá pouze v tom, že zatímco u ukazatelů se předpokládá, že obsahují posunutí definující adresu v běžném segmentu zásobníku, u indexových registrů se naopak předpokládá, že obsahují posunutí definující adresu v segmentu dat.

Registr příznaků obsahuje kromě příznaků, odpovídajících příznakům A, AC, Z, S a P, jejichž význam byl popsán v předcházejících kapitolách, která však nesou mnemonická jména CF, AF, ZF, SF, PF, další příznaky OF, DF, IF a TF.

Stav příznaku DF (Direction Flag) rozhoduje o tom, zda při operacích s textovými řetězci se provádí autoinkrementace nebo autodekrementace obsahu ukazatele datového zásobníku. Stav přízna-

ku IF (Interrupt Flag) rozhoduje o přijetí vnějších přerušení. Konečně stav příznaku TF (Trap Flag) umožňuje převést mikroprocesor do režimu provádění programu instrukce po instrukci.

8.3.3 Operační kód mikropočítače MCS-86 obsahuje jako podmnožinu většinu instrukcí operačního kódu mikropočítače MCS-80. Je však orientován na snadnou implementaci vyšších problémově orientovaných jazyků s blokovou strukturou, např. PL/M, používá několik adresovacích metod. Standardní logické operace, posuvy a přesunové operace lze provádět s osmibitovými a šestnáctibitovými operandy. K instrukcím rozšířeného souboru instrukcí patří instrukce pro práci s textovými řetězci, pro práci s jednotlivými bity, pro přechody do jiných programových segmentů a některé aritmetické instrukce.

Adresovací metody

V dvouadresových instrukcích může být jedním z operandů obsah pamětového místa operační paměti, druhým z operandů obsah některého z registrů nebo přímý operand. Operandy obsažené v operační paměti se adresují buď přímo pomocí šestnáctibitového pole posunutí, nebo nepřímo, pomocí obsahu bázevého nebo indexového registru sešteného s polem osmibitového nebo šestnáctibitového posunutí. Výsledek dvouadresových instrukcí může být uložen na jednu z adres zdrojových operandů. Jednoadresové instrukce mohou pracovat s operandy uloženými jak v registrech, tak v operační paměti. Nepoužívají se však přímé operandy.

Operandy v operační paměti se tedy adresují v podstatě čtyřmi metodami:

- a) přímo šestnáctibitovou adresou
- b) nepřímo pomocí součtu obsahu bázevého registru s osmi nebo šestnáctibitovým posunutím
- c) nepřímo pomocí součtu obsahu indexového registru s osmi nebo šestnáctibitovým posunutím
- d) nepřímo pomocí součtu obsahů bázevého a indexového registru s osmi nebo šestnáctibitovým posunutím.

Jako bázevý registr může sloužit buď bázevý registr BX ze souboru univerzálních registrů nebo ukazatel báze BP. Jestliže jako bázevý registr slouží registr BX, vyplývá z toho implicitně,

že operand leží v běžném datovém segmentu. Jestliže však jde o registr BP, pak adresová část instrukce implicitně ukazuje na běžný segment zásobníku. Jestliže se současně využívají jak bazový tak indexový registr, pak je běžný segment určen použitým bazovým registrem. Jestliže se adresuje pouze pomocí indexového registru, pak je operand obsažen v datovém segmentu.

Obsahy univerzálních registrů a ukazatelů a indexových registrů mohou vystupovat jako operandy v operacích s šestnáctibitovými operandy s výjimkou operací násobení, dělení a operací s textovými řetězci, které implicitně definují obsah registru AX. V operacích s osmibitovými operandy platí stejná omezení pro obsah registru AL; jinak lze pracovat s obsahy všech osmi osmibitových registrů. Instrukce násobení, dělení a instrukce pro práci s textovými řetězci nedovolují využít přímý operand.

Příklady použití adresovacích metod

Jednoduché proměnné se adresují přímým adresováním. Prvek pole proměnných se adresuje nepřímou jako součet obsahu indexového registru, např. SI a obsahu pole posunutí. Indexový registr obsahuje index prvku pole, posunutí udává počáteční adresu pole, vztaženou k počátku běžného segmentu dat. Proměnné vztažené k bázi, t.j. k bazové adrese se nalézají na adrese, na kterou ukazuje jiná proměnná. Jestliže obsah registru BX reprezentuje hodnotu této proměnné, pak pro adresování proměnné vztažené k bázi se může použít nepřímé adresování pomocí tohoto bazového registru. Pro adresování pole proměnných vztaženého k bázi používáme nepřímé adresování využívající součet obsahu indexového a bazového registru a pole posunutí. V bazovém registru je uložena bazová adresa, v indexovém registru index prvku pole a v poli posunutí počáteční adresa pole vztažená k bazové adrese. Bazová adresa se vztahuje k počátku běžného segmentu dat.

Pro implementaci jazyků s blokovou strukturou a implementaci reentrantních procedur lze výhodně využít nepřímé adresovací metody pracující s obsahem bazového registru BP. Tento registr slouží jako ukazatel zásobníku běžného bloku. Tak lze adresovat jak jednoduché proměnné tak pole proměnných aktivního bloku. Při přechodu k jinému bloku lze měnit pouze obsah registru BP.

Přesunové instrukce

Přesunové instrukce mohou být rozděleny do čtyř kategorií: univerzální, střadačové, pro práci s adresami a pro práci s příznaky.

K univerzálním instrukcím patří instrukce MOVE, POP, PUSH, EXCHANGE, které mohou pracovat se všemi typy operandů. Z jejich mnemonických kódů je zřejmé, že předepisují přesuny mezi registry a operační paměti, práci se zásobníkem a výměny obsahů registrů a paměťových míst.

Střadačové instrukce zahrnují vstupně-výstupní instrukce a překládací instrukce. Vstupně-výstupní instrukce mohou přímo adresovat prvních 256 vstupně-výstupních zařízení, stejně jako instrukce mikropočítače MCS-80. Kromě toho lze však adresovat nepřímo pomocí obsahu registru DX. Adresový rozsah je v tomto případě 64 K. Navíc se mohou vstupně-výstupními instrukcemi předepisovat přesuny osmi i šestnáctibitových slov. Překládacími instrukcemi lze předepisovat konverzi bytů předepsanou tabulkou. Tyto instrukce se často používají ve spojení s instrukcemi pro práci s textovými řetězci při konverzi bloků dat.

Instrukce pro práci s adresami umožňují přístup k adresám operandů v celém adresovém rozsahu operační paměti. Jediná instrukce tohoto typu ukládá adresu buď do jednoho z ukazatelů nebo do jednoho ze segmentových registrů.

K instrukcím pro práci s příznaky patří instrukce PUSH, POP, LOAD, STORE.

Aritmeticko-logické instrukce

Na rozdíl od mikropočítače MCS-80, který je vybaven pouze dvěma základními operacemi s neznaménkovými čísly, předepisuje operační kód mikropočítače 8086 všechny čtyři základní operace se znaménkovými čísly v dvojkovém doplňku. Příznak OF detekuje přetečení při operacích se znaménkovými čísly. Desítkovou korekci při práci s binárně kódovanými desítkovými čísly lze předepsat jak při jejich zápisu ve zhuštěném tak i nezhuštěném formátu, např. v kódu ASCII. Operační kód obsahuje dvě instrukce příkazující korekci; jedna se používá po sčítání, druhá po odečítání. Při násobení je výsledkem slovo s dvojnásobnou délkou, t.j. při osmibito-

vých operandech šestnáctibitové slovo, při šestnáctibitových operandech třicetidvoubitové slovo. Při dělení dělence s dvojnásobnou délkou dělitelem s jednoduchou délkou je podíl a zbytek reprezentován slovy s jednoduchou délkou.

Kromě standardních logických instrukcí AND, OR, XOR a NOT obsahuje operační kód mikropočítače MCS-86 instrukci TEST, umožňující testovat stav zvolených bitů. Tato instrukce je shodná s instrukcí AND (včetně odpovídajícího nastavení příznaků) až na to, že její výsledek se neukládá, takže testování je nedestruktivní a nemění operand, t.j. obsah registru vstupujícího do operace.

Čtyři instrukce rotace mikropočítače MCS-80 jsou v operačním kódu mikropočítače MCS-86 doplněny čtyřmi instrukcemi posunu. U všech těchto instrukcí lze předepsat posun nebo rotaci o jedno nebo několik míst.

Instrukce pro práci s textovými řetězci

Tato skupina instrukcí pracuje s řetězcí bytů i s řetězcí slov. Základní operace se mohou provádět opakovaně, jestliže se před instrukcí zařadí speciální prefixový byte. Všechny základní operace používají obsah registru SI pro adresování zdrojových operandů, o kterých se implicitně předpokládá, že jsou umístěny v běžném datovém segmentu. Obsah registru DI slouží obecně pro adresování paměťových míst příjemce, o kterých se předpokládá, že jsou v běžném speciálním segmentu. Tito ukazatelé adres zdroje a příjemce se v závislosti na stavu příznaku DF automaticky buď inkrementují nebo dekrementují buď o jednu při práci s řetězcem bytů, nebo o dvě při práci s řetězcem slov.

Jestliže instrukci předchází prefix, pak se její provádění opakuje až do splnění předepsaného počtu opakování, určeného obsahem registru CX. Splnění této podmínky se testuje vždy před vykonáním instrukce. Prefixový byte rovněž určuje stav, který se srovnává se stavem příznaku ZF. Ovlivňuje-li základní operace stav tohoto příznaku, a je-li tento stav rozdílný oproti stavu předepsanému prefixem, pak se ukončí opakování základní operace. To umožňuje prohledávání textových řetězců.

Stav čítače CX a ukazatelů adres operandů DI a SI se během opakování instrukce aktualizuje. Obsah čítače instrukcí přitom

ukazuje na adresu paměťového místa s prefixovým bytem, což umožňuje snadno obnovit opakování instrukce po návratu z přerušení.

Operační kód mikropočítače 8086 obsahuje pět základních instrukcí pro práci s textovými řetězci. Instrukce MOVb a MOVW (MOVb je bytově orientovaná, MOVW slovně orientovaná) přesunují zdrojový operand do příjmového operandu. Při opakování se zajišťuje přesun řetězce z jedné zóny paměti do jiné zóny paměti.

Instrukce CMPb a CMPW odečítá příjmový operand od zdrojového operandu a ovlivňuje příznaky, aniž by však ukládala výsledek. Jako opakovaná instrukce slouží k porovnávání dvou textových řetězců. Pomocí prefixu lze předepsat srovnání dvou textových řetězců, při němž lze určit, po který prvek řetězců se řetězce shodují.

Instrukce SCAB popřípadě SCAW odečítá příjmový operand od obsahu registru AL (SCAB) nebo AX (SCAW) a ovlivňuje příznaky, aniž by ukládala výsledek. Jako opakovaná instrukce slouží k prohledávání textového řetězce k zjištění, zda obsahuje byte nebo slovo, shodné s obsahem registru AL resp. AX.

Instrukce LODB resp. LODW ukládá zdrojový operand do registru AL resp. AX. Tato operace se obvykle neopakuje. Instrukce STOB resp. STOW ukládá operand z registru AL resp. AX do příjmového operandu. Opakováním této instrukce lze naplnit prvky textového řetězce zadanou hodnotou.

Opakované provádění základních instrukcí předepsaných pomocí prefixu uskutečňují technické prostředky. Opakované provádění složitějších operací lze předepsat pomocí instrukcí pro řízení iteračních výpočtů. K těmto instrukcím patří čtyři instrukce:

Instrukce LOOP dekrementuje obsah čítače CX o jedničku a předepíše skok na začátek smyčky, není-li obsah čítače CX nulový.

Instrukce LOOPZ dekrementuje obsah čítače CX o jedničku a předepíše skok na začátek smyčky, není-li obsah čítače CX nulový a stav příznaku ZF jedničkový.

Instrukce LOOPNZ dekrementuje obsah čítače CX o jedničku a předepíše skok na začátek smyčky, není-li obsah čítače CX nulový a stav příznaku ZF nulový.

Instrukce JCXZ se užívá pro přeskok smyčky, jestliže je obsah

čítače CX nulový.

Kombinací základních instrukcí pro práci s textovými řetězci a dalších instrukcí operačního kódu s instrukcemi pro řízení iteračních výpočtů lze vytvořit efektivní programy pro práci s textovými řetězci. Velmi často se např. využívá překládacích instrukcí, dovolujících předepsaným způsobem zpracovat (např. dekodovat) operand převzatý ze zdrojového textového řetězce před jeho předáním do textového řetězce příjemce. Při překládací instrukci je např. obsah registru AL, do kterého byl instrukcí LODB přečten zdrojový byte, interpretován jako index překládací tabulky, na jejíž počátek ukazuje obsah registru BX. Prvkem této tabulky se nahradí obsah registru AL, další instrukcí CMPB se např. porovná s předepsaným znakem a instrukcí STOB se запиše do bytu příjemce.

Jako příklad využití může posloužit zápis informačního bloku v kódu ASCII, ukončeného znakem EOT. (přijímaného ze vstupního zařízení mapovaného do operační paměti) do předepsané zóny operační paměti v kódu EBCDIC. Vstupní zařízení přijímající informační blok je chápáno jako místo v operační paměti. Při inicializaci tohoto přenosu se např. do indexového registru SI запиše začátek zóny operační paměti (sloužící jako vyrovnávací paměť) informačního bloku v kódu EBCDIC. Do registru BX se запиše adresa začátku tabulky překladu z kódu ASCII do kódu EBCDIC a do registru CX maximální délka informačního bloku. Posloupností instrukcí, popsanou v předcházejícím odstavci, lze pak zapsat standardní podprogram příjmu informačního bloku.

Instrukce pro řízení programu

Kromě instrukcí pro řízení iteračních výpočtů patří do této skupiny další základní instrukce: skokové a pro práci s podprogramy. Dělíme je do dvou skupin: instrukce pro řízení běžného programového segmentu a pro řízení mezisegmentových přechodů. Instrukce obou skupin mohou používat přímé i nepřímé adresovací metody, které byly popsány v předchozích odstavcích.

Instrukce pro řízení běžného programového segmentu používají jako operand přímou adresou, vyjádřenou jako posunutí vztahené k počátku segmentu. Strojový kód je pak nezávislý na alokaci programu; je tedy dynamicky relokovatelný. K těmto instrukcím patří např. skoková instrukce s relativní adresou vztahenou k běžné adrese in-

strukce. Má zkrácené pole posunutí s délkou osm bitů, kterou lze obsáhnout adresový rozsah + 127 a - 128 vůči běžné adrese. Obohacení operačního kódu představuje též návratová instrukce, jejímž operandem lze předepsat počet pamětových míst zásobníkové paměti, které mají být při návratu z paměti vysunuty. Tak lze po návratu z podprogramu odstranit již nepotřebné skutečné parametry, které byly do zásobníku zapsány při vyvolání programu.

Podmíněné instrukce pro práci s podprogramy typu CZ, RZ a pod. používané v mikropočítači MCS-80 nejsou do operačního kódu mikropočítače 8086 zahrnuty, neboť se ukázalo, že se využívají velmi zřídka, takže jejich vyloučení nepředstavuje pro programátora podstatné omezení.

8.3.4 Styk mikropočítače MCS-86 s okolím

K nejzávažnějším aspektům styku mikropočítače s okolím patří vyřešení mechanismu přerušení. Mikroprocesor 8086 rozeznává dva základní typy: maskované a nemaskované.

Přijetí přerušení má za následek přechod do předem zadaného programového segmentu na určenou adresu. Tabulka přerušení vektorů na začátku operační paměti má 256 prvků. Každý její prvek obsahuje ukazatel adresy příslušného standardního podprogramu obsluhy přerušení. Ukazatel adresy obsahuje dva byty adresy posunutí. Každému zdroji přerušení, t.j. každému typu požadavku o přerušení odpovídá jeden prvek tabulky. Po přijetí požadavku na přerušení se automaticky uloží obsah registru příznaků do zásobníku a pak se vykoná vyvolávací instrukce s nepřímou adresou, převzatou z tabulky vektorů přerušení.

Operační kód zahrnuje dvě instrukce, kterými lze vyvolat programové přerušení. První z těchto instrukcí je dvoubytová; její druhý byte specifikuje typ žádosti o přerušení, takže může vyvolat libovolný standardní podprogram obsluhy. Druhá instrukce je pouze jednobytová a může vyvolat jen jeden obslužný podprogram. Tato instrukce se používá pro implementaci odlaďovacího programu, neboť s její pomocí lze zapsat do programu zastavovací body. Návraty z přerušení lze předepsat speciální návratovou instrukcí, která kromě běžné činnosti při návratu obnovuje ze zásobníku obsah registru příznaků.

Jestliže je nastaven příznak TF do stavu 1, dochází k přerušení po každé instrukci. Při přijetí požadavku na přerušení a vyvolání standardního podprogramu obsluhy se při ukládání obsahu registru příznaků příznak TF vždy nuluje. Stav tohoto příznaku nelze přímo měnit žádnou instrukcí ale lze jej modifikovat po uložení do zásobníku. Tuto modifikaci může např. zajistit diagnostický nebo odlaďovací program přerušením vyvolaný. Činnost tohoto programu probíhá normálně v souvislém režimu, avšak po návratu z tohoto programu do testovaného programu lze modifikací příznaku TF do stavu 1 předepsat opět vykonání jediné instrukce. Testovaný program se tak vykoná krok po kroku a po každém vykonání jedné instrukce se vyvolá diagnostický program, který však probíhá v souvislém režimu.

Speciální instrukce ESCAPE umožňuje na úrovni procesů synchronizovat činnost mikroprocesoru 8086 s činností jiného specializovaného procesoru, který uskutečňuje např. aritmetické operace. Požadovaná aritmetická operace je v tomto případě předepsána operandem instrukce ESCAPE v odpovídajícím místě programu mikropočítače 8086. Po převzetí operační části této instrukce nečte mikroprocesor 8086 její operand, ale čeká na potvrzení připravenosti specializovaného procesoru. Specializovaný procesor do této chvíle pouze monitoroval datovou sběrnici mikropočítače 8086. Jakmile však rozeznal instrukci ESCAPE a je připraven provádět aritmetickou operaci, potvrdí svou připravenost signálem, přiváděným na zvláštní vývod pouzdra mikroprocesoru 8086. Po potvrzení připravenosti uskuteční mikroprocesor 8086 čtení operandu instrukce ESCAPE. Při tomto čtení se předává adresa operandu sběrnici. Výsledek čtení není zpracován mikroprocesorem 8086, ale je určen specializovanému procesoru.

Formát instrukce ESCAPE je volen shodně s formátem instrukce pro vyvolání podprogramu obsluhy přerušení. Činnost specializovaného procesoru může být nahrazena odpovídajícím standardním podprogramem. Strojový kód generovaný překladačem je přitom nezávislý na prostředích, kterými se požadovaná činnost uskutečňuje. Tyto prostředky, t.j. buď standardní program nebo specializovaný procesor, určuje až zaváděcí program.

V multiprocessorových systémech, ve kterých musí jednotlivé

procesory sdílet některé ze systémových zdrojů, je nutné řídit jejich přístup k těmto zdrojům. Tímto úkolem je obvykle pověřen operační systém, který však musí mít k dispozici vhodné technické vybavení.

Mikroprocesor 8086 je vybaven mechanismem pro blokování přístupu k systémovým zdrojům, který splňuje požadavky na řízení činnosti multiprocesorového systému. Každé instrukci může být předřazen speciální prefixový byte. Tento prefix způsobí, že mikroprocesor 8086 uplatní blokovacím signálem nárok na použití sběrnice. Vnější technické prostředky zakáží na základě tohoto signálu přístup k sběrnici jiným systémovým prostředkům.

9. MIKROPROGRAMOVÁNÍ

9.1 Úvod

V uzavřených systémech určují jednotlivé funkční bloky a jejich propojení architekturu mikropočítače. K uzavřeným systémům patří univerzální počítače první a druhé generace a větší na minipočítačů a mikropočítačů, dále speciální počítače pracující s velkou výpočetní rychlostí. Architekturu uzavřených systémů lze změnit změnou konfigurace funkčních bloků a jejich vnitřní struktury, tedy změnou technických prostředků.

Z hlediska uživatele a výrobců počítačů je žádoucí taková flexibilita architektury, která nežaduje zásahy do technických prostředků, což umožňuje přizpůsobit architekturu počítače požadavkům uživatele, nebo naopak přizpůsobit technické prostředky požadované architektuře počítače. Flexibilita architektury je vlastní počítačům a mikropočítačům, které patří mezi tzv. otevřené systémy a je umožněna mikroprogramováním.

Přednost mikroprogramování spočívá jednak v možnosti vytvoření celé rodiny počítačů, které mají stejnou architekturu, ale liší se technickými prostředky, jednak možností vybavit jeden počítač různými architekturami. Míra univerzálnosti počítače, tj. jeho schopnost řešit úlohy různého charakteru předepsané různými algoritmy, je schématicky znázorněna na obr. 9.1. Ne-programovatelné systémy jsou určeny pro specializované úlohy. Druhým extrémem jsou plně programovatelné systémy, reprezentované abstraktním modelem - Turingovým počítačem - který obsahuje minimum technických prostředků, a který má nejuniverzálnější charakter. Většina prakticky realizovaných počítačů leží ve vyčárkované oblasti obr. 9.1, přičemž uzavřené systémy leží v levé části této oblasti, otevřené systémy v pravé.

9.2 Koncepce mikroprogramování

Procesor počítače se skládá ze dvou typů obvodů: výkonných a řídicích. Výkonné obvody obsahují kombinační a paměťové obvody, které jsou spolu propojeny a kterými probíhají signály reprezentující zpracovávaná data. K nejdůležitějším výkonným obvodům patří aritmetická jednotka, ve které dochází k vlastním zpracování dat. Řídicí obvody interpretují instrukce zapsané v programu a řídí tok dat mezi výkonnými obvody a předepisují způsob jejich zpracování.

Řadič interpretuje instrukci programu pomocí posloupnosti elementárních kroků, které nazýváme mikrooperacemi. Činnost předepsaná instrukcí je tedy rozložena na posloupnost činností, vykonávaných postupně v jednotlivých elementárních krocích.

Jedná se tedy o postup, který je zcela analogický postupu, jakým je řešena úloha na úrovni programu ve strojovém jazyku. Rozklad úlohy na posloupnost elementárních kroků neboli operací nazýváme algoritmizace a příkaz k vykonání elementárního kroku instrukce. Algoritmus zapsaný jako posloupnost instrukcí, které mohou být počítačem interpretovány, nazýváme program počítače. Analogicky můžeme označit elementární kroky na úrovni zpracování instrukce jako mikrooperace, příkazy k jejich vykonání jako mikroinstrukce a posloupnost mikroinstrukcí předepsanou algoritmem jako mikroprogram. Vztah mezi těmito úrovněmi programování je schématicky naznačen v obr. 9.2. Zdrojový program zapsaný v jazyku symbolických adres, nebo ve vyšším problémově orientovaném jazyku je assemblerem nebo kompilátorem přeložen do strojového jazyka. Každá strojová instrukce se provádí pomocí mikroprogramu, který uskutečňuje činnost předepsanou instrukcí vykonáním posloupnosti mikrooperací.

Z obrázku 9.1 je zřejmé, že představa elementárního kroku algoritmu je závislá na úrovni, pro kterou se algoritmus sestavuje. Na úrovni zdrojového programu obsahuje elementární krok podstatně složitější činnost, než na úrovni výsledného programu nebo mikroprogramu.

Analogie mezi programováním a mikroprogramováním má svá omezení. Při přechodu od zdrojového programu k výslednému pro-

gramu je každá instrukce zdrojového jazyka přeložena jako posloupnost instrukcí výsledného jazyka. Teprve po ukončení překladače se program provádí. Na úrovni mikroprogramování se instrukce strojového jazyka po jejím převzetí a dekodování bezprostředně provádí. Přechod od výsledného programu k mikroprogramu se uskutečňuje interpretacním způsobem, tedy analogicky jako při přechodu ze zdrojového programu k výslednému programu pomocí překladače interpretačního typu.

9.2.1 Pevně zapojený řadič

Posloupnost mikrooperací požadovaná pro vykonání strojové instrukce může být generována pevně zapojeným řadičem. Řadič je tvořen sekvenčním obvodem obvykle Morreova typu, ve kterém je posloupnost vnitřních stavů, generujících signály pro řízení činnosti výkonných obvodů předepsána konfigurací stavebních prvků, tj. kombinačních a paměťových obvodů. V představě, uvedené na obr. 9.1 patří řadič tohoto typu mezi neprogramované systémy tvořené pouze technickými prostředky.

Pro bližší představu uvažme hypotetický počítač, jehož struktura je znázorněna na obr. 9.3. Jeho funkční bloky jsou následující:

- R I - registr instrukcí obsahuje operační a adresovou část právě prováděné instrukce;
- Č I - čítač instrukcí obsahuje adresu následující instrukce, která má být vykonána;
- I - inkrementační obvod umožňuje přičtení jedničky k obsahu čítače instrukcí;
- A R - adresový registr operační adresy obsahuje adresu paměťového místa, jehož obsah se čte do vstupně-výstupního registru při čtení, a do něhož se z tohoto registru zapisuje při operaci zápis;
- O P - operační paměť;
- ROP - vstupně-výstupní registr operační paměti;
- A J - aritmetická jednotka uskutečňující sčítání;
- A C - střadač, t.j. aritmetický registr, v němž se tvoří výsledek aritmetické operace;
- D I - dekodér instrukce dekoduje obsah operační části instrukce a generuje signály na jejichž základě řadič instrukci

interpretuje.

Mikrooperace, které se mohou v daném uspořádání uskutečňovat, jsou uvedeny v tab. 9.1. Řídící signály jsou pouze přesunového typu. Informační cesty, po kterých se přesun uskutečňuje, jsou označeny souhlasně s obr. 9.3. Mikrooperace (ČI) $\rightarrow$ (AR) znamená přesun obsahu čítače instrukcí do adresového registru operační paměti. Předpokládáme, že při přesunu informace do inkrementačního obvodu dojde automaticky po přesunu zápisu jedničky k obsahu tohoto registru a dále, že při přesunu informace do aritmetické jednotky dojde automaticky k přičtení této informace k obsahu registru okamžité hodnoty, který je součástí této aritmetické jednotky a konečně, že po přesunu operačního znaku z registru RI do dekodéru operačního znaku DI přejde řadič automaticky na vykonávání posloupnosti mikrooperací, která interpretuje dekodovanou instrukci.

V tab. 9.2 až 9.5 jsou uvedeny posloupnosti mikrooperací, které implementují některé strojové instrukce hypotetického počítače a další činnosti, jako např. převzetí instrukce.

Při převzetí instrukce, popsané v tab. 9.2 se předá nejprve obsah čítače instrukcí ČI (ukazující na následující instrukci programu) do adresovacího registru operační paměti AR. Pak je obsah paměťového místa obsahující tuto instrukci převzat do vstupně-výstupního registru operační paměti ROP. Další mikrooperace přesune obsah tohoto registru do registru instrukcí RI, kde je připraven k dekodování. Další dvě mikrooperace aktualizují obsah čítače instrukcí ČI. Přesun do inkrementačního obvodu RI a zpět zvětší obsah čítače o jedničku, takže ukazuje opět na následující instrukci. Poslední mikrooperace přesune část obsahu registru instrukcí nesoucí kód operačního znaku do dekodéru instrukcí DI, čímž se zahájí provádění instrukce předepsané operačním znakem.

Pokud operační znak předepisuje provádění instrukce LOAD, pokračuje posloupnost mikrooperací podle tab. 9.3. Instrukce LOAD předepisuje přesun obsahu paměťového místa určeného adresovou částí instrukce do střadače. První mikrooperace provádějící tuto instrukci přesune obsah adresové části registru instrukcí IR do adresového registru operační paměti AR. Pak následuje čtení operační paměti, při kterém se přečte obsah adresovaného paměťového místa do vstupně-výstupního registru operační paměti ROP. Další mik-

rooperace přesune jeho obsah do střadače. Tím je instrukce LOAD provedena. Poslední mikrooperace zajišťuje přechod na další instrukci. Obdobně se provádí instrukce STORE, předepisující přesun obsahu střadače do paměťového místa adresovaného adresovou částí této instrukce. Její uskutečnění podle tab. 9.4 nebudeme již podrobně vysvětlovat, stejně jako provádění instrukce ADD, předepisující se čtení obsahu střadače s obsahem paměťového místa, jehož adresa je obsažena v adresové části instrukce (tab.9.5) a ponecháváme je čtenáři.

Úkolem řadiče je generovat řídicí signály, které řídí provádění mikrooperací v předepsané posloupnosti, jak jsme si je ukázali v předchozích příkladech. Řadič musí umožňovat modifikovat provádění posloupnosti mikrooperací v závislosti na splnění nebo nesplnění podmínek, např. v závislosti na výsledku aritmetické operace, např. nulového výsledku, záporného výsledku nebo přetečení.

Příklad blokového uspořádání pevně zapojeného řadiče je uveden v obr. 9.4. Generátor taktovacích impulsů zajišťuje časový rozklad operace zadané instrukcí na požadovanou posloupnost řídicích signálů. Změna sledu taktovacích impulsů na základě podmínky umožňuje podmíněné větvení. Výstupy dekodéru testovacích impulsů, tvoří jeden vstup paměťové matice řídicích signálů. Druhý vstup je tvořen výstupy dekodéru operačního znaku. Výstupem paměťové matice řídicích signálů (která je kombinačním obvodem) jsou signály ovládající tok informací mezi funkčními bloky počítače, popřípadě předepisující způsob jejich činnosti.

9.2.2 Mikroprogramovaný řadič

Jak jsme již uvedli v předchozích odstavcích, pracuje řadič počítače při vykonávání instrukcí obdobným způsobem, jako počítač při provádění programu. Prvním, kdo si povšiml této analogie a navrhl řešit řadič jako programovatelný systém, byl M.V. Wilkes. Činnost řadiče tohoto typu se předepisuje pomocí posloupnosti mikroinstrukcí, tzv. mikroprogramu, uloženého ve zvláštní paměti, nazývané řídicí pamětí. Původní Wilkesovo uspořádání je uvedeno na obr. 9.5. Skládá se z registru adresy obvodu adresového výběru a řídicí paměti, tvořené pamětovou maticí mikrooperace a pamětovou

maticí příští adresy. Obsah registru adres adresuje jedno slovo paměti mikroprogramů. Toto slovo, představující jednu mikroinstrukci se skládá ze dvou částí. První část složená z paměťové matice mikrooperace nese informaci o mikrooperaci, která má být vykonána. Jsou to řídicí signály, odpovídající řídicím signálům z předchozího obr. 9.4. Druhá část, obsažená v paměťové matici příští adresy, nese informaci o následující mikroinstrukci. Na rozdíl od strojových instrukcí, které většinou explicitně neobsahují adresu následující instrukce, neboť je obsažena v čítači instrukcí, je tedy v tomto uspořádání mikroprogramovaného řadiče adresa následující mikroinstrukce explicitně v mikroinstrukci obsažena. Modifikace této adresy umožňující větvení mikroprogramu na základě podmínky je zajištěna přepínačem volícím jeden ze dvou výběrových vodičů paměťové matice příští adresy. Na základě polohy přepínače se tedy vybírá obsah paměťové matice příští adresy odpovídající buď splněné nebo nesplněné podmínce. V uvedeném uspořádání jsou obě adresy vzájemně nezávislé. Paměťová matice příští adresy musí však mít ve srovnání s paměťovou maticí mikrooperace o tolik víc řádků, kolik je v mikroprogramu zapísáno podmíněných skoků. Kromě toho každému podmíněnému skoku odpovídá jeden elektronický přepínač. Uspořádání přepínačů je tedy závislé na zápisu mikroprogramu.

Tyto nevýhody byly odstraněny v uspořádáních, které byly vyvinuty později, a jejichž společným charakteristickým znakem je splynutí obou paměťových matic. Pro ilustraci uvádíme alespoň jedno z možných uspořádání na obr. 9.6. Adresy při splněné a nesplněné podmínce v tomto zapojení se mohou lišit v jediném bitu, např. v bitu 1. Všechny ostatní bity příští adresy jsou obsaženy ve výstupním slovu paměti mikroprogramu. Bit 1 příští adresy je určen výstupním signálem multiplexoru podmínek. Jeho vstup je adresován částí výstupního slova paměti mikroprogramu, tzv. polem výběru podmínky. V závislosti na adresovací podmínce je namísto bitu dosazován stav 0 nebo stav 1. Příští adresa je tedy modifikována na základě zvolené podmínky v jediném bitu. Pro případ prostého pokračování je v souboru podmínek obsažena podmínka trvale nesplněná (stav 0), a podmínka trvale splněná (stav 1).

9.3 Aplikace mikroprogramování

Mikroprogramované systémy naleznou uplatnění v následujících aplikačních oblastech:

- 1) univerzální počítače
- 2) periferní a přídavná zařízení počítačů
- 3) speciální počítače.

Ve všech těchto oblastech je koncepce mikroprogramovaného řadiče velmi podobná. Důsledky aplikace mikroprogramování na koncepci celého systému jsou však v těchto oblastech dosti odlišné.

9.3.1 Univerzální počítače

Univerzální počítače využívající mikroprogramování plní úlohu interpretů. Jejich procesory jsou navrženy tak, aby interpretovaly operační částí strojových instrukcí a potom operace prováděly s operandy určenými jejich adresovými částmi. Výhody takového přístupu spočívají v tom, že je možné vybavit několik počítačů stejným operačním kódem, nebo naopak jeden počítač několika operačními kódy, což umožňuje emulovat jeden počítač jiným počítačem.

Kompatibilní počítače. První uváděný případ znamená možnost vyvinout rodinu počítačů, které se liší výpočetní rychlostí, technickými prostředky a vnitřní organizací, které však mají stejný operační kód. Tyto počítače jsou pak programově kompatibilní. Lze tedy přenášet program z jednoho počítače na jiný počítač. Příkladem může být systém 370 firmy IBM. Do tohoto systému patří celkem devět modelů počítačů, které se vzájemně liší organizací i technickými prostředky. Jejich mikroprogram je však navržen tak, aby odpovídal architektuře systému 370, takže z hlediska programátora jsou všechny počítače systému totožné.

Emulace. Vybavení jediného počítače několika operačními kódy umožňuje na tomto počítači provádět programy napsané pro jiné počítače, aniž by bylo nutné provádět jejich změny. Předpokladem je ovšem přizpůsobit architekturu počítače architektuře, která odpovídá programu. Počítač, na kterém jsou programy prováděny, pak označujeme jako hostitelský počítač. Počítač, pro který jsou programy určeny, označujeme jako cílový počítač. Hostitelský počítač tedy simuluje cílový počítač. Pokud však je simulace dosažena mikroprogramováním, mluvíme o emulaci. Termín simulace je tedy

vyhrazen pro případy, kdy hostitelský počítač modeluje pomocí programového vybavení architekturu cílového počítače. O emulaci mluvíme tehdy, používáme-li k modelování architektury cílového počítače mikroprogramového vybavení hostitelského systému, nebo upravujeme-li nebo doplňujeme jeho technické prostředky.

Emulace je mnohdy podstatně jednodušším úkolem než simulace. Zjednodušení vyplývá z toho, že zatímco se různé počítačové systémy podstatně liší na úrovni programování ve strojovém jazyku, např. ve formátu instrukcí a reprezentaci dat, jsou si podstatně bližší na úrovni mikroprogramování. Úroveň mikroprogramování míníme úroveň spojenou s řízením elementárních funkčních bloků. Vzhledem k tomu, že mikroprogramování umožňuje tvořit flexibilní posloupnosti mikrooperací, využívá se podobnosti systémů na této základní úrovni k emulování systémů rozdílných na vyšší úrovni, úrovni strojového jazyka.

Mikroprogramovatelné počítače

Původně bylo mikroprogramování navrženo jako prostředek pro implementaci počítače se zadanou архитектурou a neuvažovalo se o obecnějších možnostech, vedoucích k modifikacím architektury, a o použití mikroprogramování k emulaci, i když na tyto možnosti poukázal již Wilkes, který uvažoval o výměnných maticích, které by interpretovaly různé typy operačních kódů. Tyto operační kódy byly voleny tak, aby měly optimální vlastnosti pro danou aplikaci.

V počítačích, ve kterých slouží pevná paměť jako řídicí se přechod od jedné architektury k jiné zajišťuje pomocí technických prostředků a vyžaduje obvykle zásah operátora. Pokud však funkci řídicí paměti částečně nebo zcela plní paměť s možností zápisu (writeable control store), pak přechod od jedné architektury k jiné se může uskutečnit dynamicky instrukcí programu bez zásahu operátora. Tuto vlastnost označujeme jako integrovanou emulaci.

Zavedení paměti s možností zápisu znamená vznik tak zvaných mikroprogramovatelných počítačů, ve kterých může tuto programovací úroveň použít nejen návrhář počítače, ale i systémový programátor uživatele počítače.

V prvních mikroprogramovatelných počítačích byla větší část řídicí paměti realizována pomocí pevné paměti základním mikropro-

gramem a menší část, určené pro rozšíření operačního kódu pomocí speciálních instrukcí, byla realizovaná jako pevná paměť s možností zápisu. Rozšíření operačního kódu o speciální instrukce umožňuje zvýhodnění počítače v určité aplikační oblasti, neboť dovoluje speciálními instrukcemi předepsat činnosti, které mají v dané aplikaci velkou opakovatelnost, nebo které jsou při použití standardního operačního kódu náročné na čas.

V poslední době se objevují na trhu počítače a mikropočítače, které jsou plně mikroprogramovatelné, jejichž řídicí paměť je převážně nebo výlučně realizována pomocí paměti s možností zápisu, a které jsou optimalizovány z hlediska programování na této nejnížší úrovni, na úrovni mikroprogramování. Znamená to, že mohou být mikroprogramovány stejně snadno, jako na vyšších úrovních programování tak, aby emulovaly různé cílové počítače, nebo plnily různé aplikační úlohy.

Mikroprogramovatelné počítače, u kterých je tato představa dovedena do důsledku, nemají strojový jazyk v konvenčním slova smyslu, ale jsou pouze mikroprogramovatelné. Při jejich návrhu nebyla uvažována specifická architektura na vyšší programovací úrovni, naopak bylo spíše spíše preferováno jiné hledisko. Tímto hlediskem je univerzálnost mikroprogramovatelného počítače ve funkci hostitelského systému. Touto univerzálností se rozumí možnost emulace jakéhokoliv cílového počítače pomocí mikroprogramu.

Univerzální hostitelský systém musí mít takovou organizaci a soubor mikroinstrukcí, které umožňují snadné přizpůsobení a optimalizaci vyhovující architektuře cílového počítače. Pod pojem cílový počítač se nemusí skrývat počítač programovatelný ve strojovém jazyku. Mikroprogramování může sloužit přímo k provádění instrukcí vyššího problémově-orientovaného jazyka, popřípadě může být přímo mikroprogramován operační systém, popřípadě uživatelský systém.

Mikroprogramovatelné počítače mohou používat na úrovni mikroprogramu stejnou metodu, jakou používají počítače na úrovni programu. Jedná se o výměnu celých stránek programu v operační paměti na úrovni programu, t.zv. techniky multiprogramování.

V případě nedostačující kapacity operační paměti se potřebné programy a data předávají do operační paměti z vnějších diskových

paměti a nepotřebné části programů a nepotřebná data se do nich ukládají. Pro úspornější využití se paměti rozdělují na bloky nebo stránky, což umožňuje dynamickou relokační programů a dat.

Využívá-li se tato technika na úrovni mikroprogramu, přesunují se analogicky mikroprogramy mezi řídicí paměti a vnější paměti. Mluvíme pak o dynamickém mikroprogramování. Vznikají tím ovšem stejné problémy jako na vyšších programovacích úrovních, především ochrana programů proti zásahu jiného programu. Na úrovni mikroprogramu je tento problém o to tíživější, že změna mikroprogramu může vyvolat změnu struktury počítače.

V dynamicky mikroprogramovatelných systémech se problém ochrany mikroprogramu řeší pomocí operačního systému, který používá k řízení styku s pamětí s možností zápisu metody privilegovaných mikroinstrukcí. Tato metoda je analogická metodě ochrany paměti na vyšších programovacích úrovních.

9.3.2 Periferní a přídavná zařízení počítačů

Některé funkční bloky počítačů a mikropočítačů, mezi něž patří především periferní a přídavná zařízení, vykonávají na základě instrukcí převzatých z procesoru poměrně složité činnosti, které jsou řízeny vnitřními řadiči těchto zařízení. Typickým příkladem je řadič multiplexního kanálu nebo řídicí jednotka pro připojení zobrazovacích jednotek k multiplexnímu kanálu. Tato zařízení můžeme pokládat za jednoúčelové procesory, jejichž struktura obsahuje registry, které jsou používány jako paměti řídicích, stavových a a datových slov. Obvykle však neobsahují operační paměti. V moderních počítačových a mikropočítačových systémech jsou tato přídavná a periferní zařízení realizována jako mikroprogramem řízené procesory, které mají jenom tuto jedinou programovací úroveň. Technika mikroprogramování přináší všechny výhody programovacích systémů i v této oblasti.

9.3.3 Speciální počítače

Pro řízení technologických, dopravních i jiných procesů se používá minipočítačů a mikropočítačů. Úloha těchto počítačů, které v těchto aplikacích nazýváme řídicími počítači, je prohlížení vstupních dat, jejich vyhodnocení a generování řídicích signálů. Převáž-

ná část řídicích počítačů se programuje na úrovni strojového jazyka. Programy jsou mnohdy poměrně složité, neboť instrukce strojového jazyka nejsou z hlediska řízeného procesu optimální. Některé mikropočítače a minipočítače, orientované na řízení procesů, mají proto ve svém operačním kódu zařazeny instrukce, které usnadňují algoritmizaci řídicích úloh, např. testování stavu jednotlivých bitů stavových slov a pod. Ani tento přístup není zcela univerzální. Optimalizaci operačního kódu pro danou aplikaci umožňuje mikroprogramovatelné mini- a mikropočítače.

9.4 Architektura a organizace počítače

V dosud uvedené představě mikroprogramem řízeného počítače jsme uvažovali dvě programovací úrovně. Vyšší programovací úroveň představuje program v operační paměti počítače s instrukcemi ve strojovém jazyku, které bývají někdy v literatuře pojednávající o emulaci nazývány makroinstrukce. Význam tohoto termínu je odlišný od jeho konvenčního významu. Označuje pouze, že se jedná o úroveň instrukcí, které je o jednu vyšší, než úroveň mikroprogramu, zapsaného pomocí mikroinstrukcí. Mikroprogram představuje druhou programovací úroveň a je zapsán v řídicí paměti.

Programovatelný řadič počítače nemusí mít pouze jednu programovací úroveň. V některých případech se ukázalo výhodné rozdělit řízení do dvou úrovní. Potom nazýváme program nejnižší úroveň nanoprogram. Je zapsán jako posloupnost nanoinstrukcí v paměti nanoprogramu. Teoreticky je možné rozdělit programové řízení do libovolného počtu úrovní. V dosud realizovaných počítačích a mikropočítačích se však používají dvě, popřípadě tři úrovně programového řízení.

V systémech s jedinou programovací úrovní jsou definovány pojmy architektura a organizace počítače. Pod pojem architektura se zahrnuje vybavení počítače, které charakterizuje počítač z hlediska programátora. Jedná se především o formát instrukcí, soubor instrukcí a soubor registrů přístupných programátoru. Pod pojem organizace počítače zahrnujeme charakteristiku struktury počítače, která je z hlediska programátora nepodstatná a popisuje počítač na nižší úrovni než je popsána jeho architekturou. Architektura popisuje tedy funkční vlastnosti počítače a organizace prostředky,

kterými jsou funkce počítače dosaženy.

Abychom si tento rozdíl ujasnili, uvažme velmi často používanou metodu překrytí fází převzetí a vykonání instrukce pro případ podmíněného skoku. V některých počítačích je v případě splnění podmínky automaticky nahrazena předem převzatá instrukce instrukcí, která má následovat po podmíněném skoku. Tato činnost sice vyvolá zpoždění při provádění instrukce oproti jejímu provádění při nesplněné podmínce, programátor se však o tuto činnost nemusí starat. Popis činnosti při převzetí instrukce patří v tomto případě k úrovni, kterou jsme označili jako organizace počítače. V jiných počítačích se musí o náhradu předem převzaté instrukce při splnění podmínky postarat programátor explicitně zápisem instrukcí v programu. V tomto případě patří tato otázka k úrovni, označované jako architektura počítače.

V systémech s několika programovacími úrovněmi je nutné vztahovat pojmy architektura a organizace vždy k některé z programovacích úrovní. Parametry charakterizující jisté vlastnosti počítače mohou patřit z hlediska vyšší programovací úrovně k jeho organizaci, zatímco z hlediska mikroprogramovací úrovně patří do architektury počítače chápáné na úrovni mikroprogramu. Tuto hierarchickou strukturu programovacích úrovní znázorňuje tab. 9.6.

Mikroprogram obsažený v řídicí paměti má mnoho vlastností, které by opravňovaly jeho zařazení do programového vybavení počítače, neboť jak již jeho jméno naznačuje, jde skutečně o program. Na druhé straně, z hlediska programátora sestavujícího uživatelský program na úrovni strojového jazyka je mikroprogram a řídicí paměť pouze rozšířením technických prostředků. Tato dvojí povaha mikroprogramu vedla k tomu, že mikroprogram se obvykle nezařazuje k programovému vybavení, ale tvoří zvláštní část vybavení systému, t.zv. mikroprogramové vybavení.

Návrháři technických a programových prostředků pracují s algoritmy a rozdíl spočívá ve způsobu implementace těchto algoritmů. Obecně jakýkoliv sekvenční logický obvod může používat paměť vnitřních stavů. Jestliže tato paměť je pevná, můžeme ji pokládat spíše za součást technických prostředků a její obsah, mikroprogram, pokládat za součást mikroprogramového vybavení systému. Jestliže jde však o paměť s možností zápisu u mikroprogramovatelných systémů,

patří obsah této paměti spíše k programovému vybavení.

9.5 Architektura mikroprogramovatelného počítače

Uvedli jsme si, že do architektury počítače zahrnujeme vybavení počítače, které charakterizuje počítač z hlediska programátora. V tomto odstavci se pokusíme popsat a analyzovat parametry architektury mikroprogramovatelného počítače poněkud podrobněji. Architekturu budeme posuzovat z hlediska její vhodnosti pro univerzální hostitelský systém. Hlediskem univerzálnosti není jen možnost emulace libovolného cílového počítače, ale též o možnost přímé interpretace instrukcí některého vyššího problémově orientovaného jazyka.

Požadavky na univerzálnost hostitelského systému a na účinnost emulace lze jen velmi těžko sloučit, neboť jsou v podstatě rozporné. Architektura na úrovni mikroprogramu, navržená jednoduše k implementaci cílového počítače, bude vykazovat vysokou účinnost emulace, neboť interpretace strojových instrukcí cílového počítače bude vyžadovat jen několik málo mikroinstrukcí. Pokusíme-li se však počítač s touto architekturou použít k emulaci zcela rozdílného cílového počítače, pak účinnost emulace bude podstatně nižší, neboť počet mikroinstrukcí, potřebných k interpretaci strojové instrukce bude pravděpodobně podstatně vyšší. Ideální univerzální hostitelský systém by vykazoval přibližně stejný počet mikroinstrukcí pro libovolnou cílovou zdrojovou instrukci.

Z technického vybavení mikroprogramovatelného počítače je z hlediska architektury na mikroprogramovací úrovni určující šíře toku dat, aritmetická jednotka a paměťové registry procesoru. Dalšími důležitými faktory je uspořádání řídicí paměti, operační paměti, systém přerušování a struktura vstupů a výstupů.

9.5.1 Šíře toku dat

Z hlediska účinnost emulace je nepochybně jedním z nejdůležitějších faktorů šíře toku dat, která charakterizuje architekturu hostitelského systému na mikroprogramovací úrovni. Jestliže se v tomto parametru hostitelský a cílový systém liší, je mikroprogramátor postaven před problém přizpůsobit tyto parametry. Architektura hostitelských systémů se s tímto problémem vyrovnává dvojím způsobem.

sobem:

K prvnímu z nich patří volba velké základní šíře toku, např. několik desítek bitů. Zúžení šíře toků dat se dosáhne pomocí maskovacích registrů, které dovolují přizpůsobení na požadovanou šíři odpovídající cílovému počítači. K výhodám této metody patří neměnná rychlost zpracování dat, nezávislá na šíři toku cílového počítače a flexibilita, neboť zúžení toku se dosahuje pouze programovými prostředky.

Druhou metodou, která je podstatně úspornější a uplatňuje se daleko častěji, je přizpůsobení malé základní šíře toku dat hostitelského systému požadované šíři toku dat cílového počítače iterační mikroprogramovací technikou, při níž je delší slovo cílového systému zpracováno vícenásobným zpracováním kratšího slova hostitelského systému.

Alternativou této metody je paralelní řazení stavebnicových bloků s malou základní šíří toku dat. Výhodou této metody oproti iterační metodě je vyšší rychlost zpracování dat. U minipočítačů užívá tuto metodu např. Borroughs (Borroughs Interpreter). Velmi často se též aplikuje u mikropočítačů. Mikroprogramovatelné mikropočítače patří téměř výlučně do skupiny mikropočítačů se stavebnicovou délkou slova.

Nejvýhodnější je kombinace metody stavebnicového uspořádání a maskovacích registrů, které dovoluje přizpůsobit architekturu cílovému počítači s největší délkou slova při největší rychlosti emulace, a tuto rychlost zachovat při emulaci jiných cílových počítačů s kratším slovem. Z tohoto hlediska splňují požadavky na ideální univerzální hostitelský systém např. mikropočítače INTEL 3000, AM 2900 atd.

9.5.2 Aritmetická jednotka

Vlastnosti aritmetické jednotky mají zásadní vliv na účinnost emulace. V zásadě musí tato jednotka plnit základní aritmetické funkce sčítání a odečítání a logické funkce součinu a součtu s binárními operandy. Násobení a dělení je obvykle zajišťováno mikroprogramem. Je výhodné, aby operandy aritmetických operací mohla být desítková čísla v úsporném formátu. K dalším režimům aritmetické

jednotky patří zpracování čísel s pohyblivou čárkou. Vzhledem k tomu, že zobrazení se může u cílových počítačů podstatně lišit, musí mít i aritmetická jednotka pracující v tomto režimu dostatečnou flexibilitu. K dalším operacím patří operace posuvu, které může vykonávat samotná aritmetická jednotka, nebo zvláštní funkční blok.

K důležitým operacím, požadovaným při univerzální emulaci, patří výběr libovolné části slova jak z hlediska libovolné délky, tak z hlediska polohy registru procesoru. Tato operace se uplatní při dekodování operačního znaku cílového počítače a dekodování dalších částí instrukce, např. pole adresy registru, pole modifikátoru, adresní části atd. Tyto výběrové operace mohou být předepsány buď instrukcemi posunu nebo speciálními mikroinstrukcemi, jejichž operandy jsou např. nejvyšší bit pole a délka pole, které má být vybráno.

Aritmetická jednotka univerzálního hostitelského systému by měla zpracovat operandy různého charakteru a proměnné délky i pracovat v různých režimech, aby se mikroprogramátor mohl přizpůsobit požadavkům cílového počítače. Pro zpracování operandů s vícenásobnou přesností musí dovolit předepsat a vyhodnotit přenos. Musí stejně výhodně pracovat s operandy v prvním i druhém doplňku.

Z hlediska účinnosti emulace je též důležitým parametrem možnost simultánní činnosti, která je daná počtem funkčních bloků aritmetické jednotky, nezávislým přístupem k nim a jejich vzájemně nezávislou činností. Jestliže na př. aritmetická jednotka obsahuje nezávislý posuvný registr, je možné jedinou mikroinstrukcí předepsat současně aritmetickou operaci a vícenásobný posuv. Jestliže je tento posuvný registr nezávisle přístupný, operand aritmetické instrukce a operand posuvu mohou být nezávislé.

9.5.3 Registry procesoru

Účinná emulace vyžaduje, aby hostitelský počítač měl přinejmenším stejný počet univerzálních a speciálních registrů jako cílový počítač. V opačném případě musí mikroprogramátor vytvořit virtuální paměť procesoru pomocí operační paměti. V případě, že tato paměť je pomalejší než procesor, zpomaluje se rychlost emulace. Velký počet univerzálních registrů zvyšuje univerzalitu hostitelské-

ho systému, neboť umožňuje přímé přiřazení registrů hostitelského a cílového počítače. Rovněž je výhodné, jestliže alespoň některé univerzální registry hostitelského systému mají navíc vlastnosti, umožňující jejich využití jako speciální registry cílového počítače, jako např. čítače instrukcí na úrovni programu. Emulaci usnadňuje možnost přímé i nepřímé adresace registrů. Tato možnost zbavuje mikroprogramátora nutnosti dekodování adresy registru cílového procesoru, popřípadě převádění adresového pole cílové instrukce do pole mikroinstrukce.

Zásobníková paměť hostitelského systému usnadňuje jednak emulaci cílového počítače s architekturou orientovanou na zásobníkovou paměť a za druhé vytváří potřebné technické prostředky pro mnohaúrovňovou práci s mikroprogramy.

9.5.4 Řídící paměť

Z předchozích odstavců vyplývá, že ve většině případů je žádoucí použít řídicí paměť, jejíž převážná část je tvořena pamětí a možností zápisu. Paměť tohoto typu je nevyhnutelná pro dynamicky mikroprogramované systémy. Vzhledem k tomu, že rychlost řídicí paměti bývá o řád vyšší než rychlost operační paměti, je z hlediska emulace výhodné ji využít k implementaci registrů cílového počítače, popřípadě jeho zásobníkové paměti, pokud je počet registrů procesoru hostitelského programu nedostatečný. Pro tyto účely je však nezbytné doplnit technické prostředky o registry umožňující nezávislé adresování řídicí paměti o další vstupně-výstupní registr, obdobný odpovídajícím technickým prostředkům paměti na úrovni programu. Analogie je patrná z obr. 9.7. Řízení mikroprogramu je uskutečňováno pomocí registru adresy mikroprogramu, který obsahuje adresu příští mikroinstrukce, a registru mikroinstrukcí obsahujícího mikroinstrukci právě prováděnou. Funkce těchto registrů je obdobná funkci čítače instrukcí a registru instrukcí na úrovni programu. Pro zápis a čtení dat slouží, podobně jako v operační paměti, adresový registr a vstupně-výstupní registr řídicí paměti.

Flexibility řídicí paměti se dosahuje, pokud se týče rozsahu, v zásadě dvěma metodami:

První z nich je obdobná metodě virtuální paměti, používané v architektuře na úrovni strojového programu. V architektuře na

úrovni mikroprogramu je zajištěna technickými prostředky, umožňujícími výměnu obsahu operační paměti a řídicí paměti. Virtuální řídicí paměť je vytvářena pomocí mikroprogramového operačního systému.

Druhou metodou, používanou zvláště u mikropočítačů, je adresovací systém stavebnicového typu, který umožňuje kaskádním řazením stavebnicových adresovacích bloků zvětšovat délku adresy řídicí paměti. Kapacita řídicí paměti se může zvětšovat úměrně adresovacím možностям.

9.5.5 Operační paměť

Podobně jako jsou přiřazovány registry procesoru hostitelského a cílového počítače, jsou přiřazována paměťová místa operačních pamětí. Rozhodujícími parametry je délka slova a kapacita operační paměti.

Pokud je doba cyklu operační paměti poměrně velká, je lépe volit velkou základní délku slova a přizpůsobit jí délce slova cílového počítače metodou maskování nebo zhuštěným zápisem, než volit základní délku slova malou, a přizpůsobení docílit iterační metodou, o které jsme se zmínili v odst. 9.5.1.

Kapacitu operační paměti hostitelského systému je třeba přizpůsobit cílovému počítači. Hlavním omezujícím činitelem je maximální počet přímo adresovatelných paměťových míst. Přímé adresování paměťových míst je předpokladem pro vyhovující účinnost emulace, neboť dovoluje mikroprogramátoru při emulaci zavést bez problémů jakoukoliv adresovací strukturu cílového počítače.

9.5.6 Struktura sběrnic

Struktura sběrnic představuje další ze závažných charakteristik architektury na úrovni mikroprogramu, neboť je spojena s problémem nejen šíře toku dat, diskutovaným v odst. 9.5.1, ale též s problémem flexibility zpracování informací.

Jestliže je hostitelský systém vybaven jen několika pevně danými sběrnicemi, pak taková struktura sběrnic vnáší omezení pro souběžné zpracování dat a představuje pro mikroprogramátora značné omezení. Východiskem je struktura s velkým počtem sběrnic, popří-

padě struktura s přepínatelnými sběrnicemi. Struktura sběrnic a jejich řízení se promítá zpět do struktury a formátu mikroinstrukcí. Jestliže např. aritmetická jednotka má pro každý vstup nezávislou sběrnici a další sběrnici pro výstup, pak určení zdrojů a příjemců informace vyžaduje mikroinstrukci tříadresového typu. Dvě adresy určí zdroj a jedna adresa příjemce výsledku aritmetické nebo logické operace.

9.5.7 Systém přerušení

Z hlediska emulace je výhodnější systém přerušení řešený pomocí programových a nikoliv technických prostředků. Nejlepším řešením je systém postupných dotazů na úrovni mikroprogramu, ve kterém jsou postupně mikroprogramem testovány stavy příznaků požadavků na přerušení. Tyto příznaky jsou při vyvolání přerušení uváděny do dohodnutého stavu technickými prostředky. Tento systém dovoluje mikroprogramátoru přizpůsobit systém přerušení hostitelského systému cílovému systému, neboť má vyšší flexibilitu než systém vektorového přerušení. Přerušení řízené výlučně technickými prostředky je vhodné použít jen pro ošetření závažných poruch, např. výpadku sítě a pod.

9.5.8 Struktura vstupů a výstupů

Struktura vstupů a výstupů hostitelského počítače je dalším ze závažných činitelů, které mohou do značné míry ovlivnit účinnost emulace, popřípadě se stát omezujícím činitelem, který rozhodne o možnosti emulace.

Podobně jako u ostatních prvků architektury, jako např. registrů procesoru, paměťových míst operační paměti atd., je nutné zvolit přiřazení mezi vstupy a výstupy hostitelského a cílového počítače. Kromě toho musí být zajištěny všechny činnosti, od vyvolání cílové vstupně-výstupní instrukce až do jejího vykonání ve vstupně-výstupním zařízení, ke kterým patří např. činnosti spojené s řízením kanálu a pod. Není samozřejmě nutné, aby existovalo přiřazení jednotlivých fází činnosti hostitelského a cílového počítače, výsledek všech těchto činností musí však být shodný.

Emulace počítačů druhé generace popřípadě minipočítačů je snadnější než emulace počítačů třetí generace, neboť struktura vstupů-

- výstupů je stále složitější a vstupně-výstupní kanál počítače třetí generace představuje specializovaný procesor, schopný plnit náročné funkce. Pro jediný mikroprogramovatelný hostitelský procesor se emulace základních cílových instrukcí a vstupně-výstupních instrukcí stává časově příliš náročným problémem. Nejlepším řešením se v mnoha případech ukázalo rozdělení úloh mezi dva hostitelské procesory; úkolem hostitelského systému vstupně-výstupního procesoru se může s úspěchem zhostit mikroprogramovatelný mikropočítač.

9.6 Struktura operačních jednotek mikroprogramovatelných mikropočítačů

V předchozí kapitole jsme se zabývali technickým vybavením mikroprogramovatelného počítače určujícím jeho architekturu na mikroprogramovací úrovni. Toto vybavení, kterým jsme se dosud zabývali, odpovídá naší představě výkonných obvodů, ve kterých probíhá zpracování dat, jak jsme si ji zavedli v odst. 9.2.

Seznámíme se nyní s operačními jednotkami některých mikropočítačů se stavebnicovou délkou slova, jejichž struktury obsahují nejdůležitější výkonné obvody (soubor registrů, aritmetickou jednotku a prostředky pro styk s operační pamětí) a uvážíme jejich vhodnost pro hostitelský systém na základě předchozích úvah, alespoň pokud se týká základních parametrů. Je třeba však podotknout, že všechny operační jednotky mají jednu společnou vlastnost. Základní šíře toku dat je poměrně malá, buď 2 nebo 4 bity (tab. 9.7). Požadovanou šíři toku dat (např. pro přizpůsobení cílovému počítači) lze dosahovat paralelním řazením operačních jednotek, které jsou vybaveny tak, aby toto řazení umožňovaly.

9.6.1 Operační jednotka Fairchild 9405

Operační jednotka Fairchild 9405 obr. 9.8 se skládá z aritmetické jednotky, souboru osmi pracovních registrů, vstupní sběrnice DI a výstupní sběrnice DO.

Aritmetické a logické funkce aritmetické jednotky jsou popsány v tab. 9.8.

V levé části tabulky jsou uvedeny dvojice informačních zdrojů.

Zdroj jednoho operandu je zapsán v levém okraji tabulky. Jedním operandem může být tedy buď obsah vstupní sběrnice DI nebo 0, druhým operandem obsah některého ze souboru registrů R, nebo 0 nebo F (hex). Operand 0 znamená, že na všech čtyřech vstupech a aritmetické jednotky je převeden stav 0, operand F (hex), že na všech čtyřech vstupech je stav 1, zajištěný vnitřním propojením operační jednotky. V tabulce jsou uvedeny soubory operací, které mohou být uskutečněny s operandy; jsou označeny písmeny A, B, C. Soubory operací jsou specifikovány v dolní části tabulky. Např. mezi dvojicí operandů, jejichž zdrojem je vstupní sběrnice DI a soubor registrů R, mohou být uskutečněny následující operace:

DI	$\wedge$	R
DI	$\vee$	R
DI	$\oplus$	R
DI	+	R

V pravé části tabulky jsou uvedeni příjemci informace. Výsledek aritmetické nebo logické operace může být předán na výstupní sběrnici DO nebo zapsán do některého ze souboru registrů R.

Struktura operační jednotky 9405 patří k nejjednodušším strukturám vytvořeným technikou velkoplošné integrace. Umožňuje provádění všech základních operací. Při emulaci složitějších systémů však je třeba počítat s nízkou účinností emulace.

9.6.2 Operační jednotka Intel 3002

Operační jednotka Intel 3002 má oproti jednotce Fairchild 9405 složitější strukturu podle obr. 9.9, která obsahuje navíc multiplexory v obou vstupech aritmetické jednotky, které dovolují volit mezi dalšími informačními zdroji, tvořenými vstupními sběrnici M, I a K, souborem registrů R a střadačem. Obsah směrnice K umožňuje maskovat ostatní vstupy multiplexoru, t. j. obsah sběrnice I a obsah registru AR, což představuje účinný prostředek pro testování obsahu jednotlivých bitů nebo polí, popřípadě pro zúžení šíře toku při emulaci. Aritmetické a logické operace mezi dvojicemi operandů jsou popsány tab. 9.9. Registr AC navíc umožňuje posuv obsahu vpravo.

Operační jednotka Intel 3002 poskytuje, podstatně větší možnos-

ti při volbě operandů díky několika vstupním sběrnicím; jejím nedostatkem je poměrně omezený soubor aritmetických operací, které může provádět.

9.6.3 Operační jednotka Advanced Micro Devices 2901

Ve struktuře této operační jednotky nalezneme pouze sedm registrů. Šest z těchto registrů tvoří soubor registrů, který má dvě nezávislá adresovací pole, takže může být současně obsah jednoho ze souboru registrů čten na výstup R a druhého na výstup W, jak znázorňuje obr. 9.10. Zapisovat je však možné pouze do jednoho z nich; data mohou být před zápisem posunuta vpravo nebo vlevo. Výstupní multiplexor umožňuje přivést na výstupní sběrnici DO buď výstup aritmetické jednotky, nebo obsah zvoleného registru ze souboru R. Nezávislá adresovací pole souboru registrů R dovoluje flexibilní výběr operandů. Bohatý repertoár aritmetických a logických operací vytváří z této operační jednotky velmi účinný nástroj na zpracování dat přesto, že nemá přímý maskovací vstup. Funkční možnosti jsou popsány v tab. 9.10.

Velmi podobnou strukturu má operační jednotka Monolithic Memories 6701. Srovnání parametrů operačních jednotek mikropočítačů se stavebnicovou délkou slova je uvedeno v tab. 9.7.

9.7. Struktura mikroinstrukcí

Nejdílnou součástí architektury mikroprogramovatelného počítače je mikropočítače tvoří soubor mikroinstrukcí, prostřednictvím kterých předepisuje mikroprogramátor tok informací a jejich zpracování. Se vzrůstající univerzálností procesoru mikroprogramovatelného počítače vzrůstají i nároky na strukturu jeho řídicí jednotky, neboť vzrůstá počet řídicích proměnných, jejichž stav se musí určit.

K nejdůležitějším parametrům mikroinstrukce patří délka mikroinstrukčního slova, mikroinstrukční cyklus, tj. doba vykonání jedné mikroinstrukce; její informační obsah, tj. množství řídicích informací a stupeň jejich zakódování a formát mikroinstrukce.

9.7.1 Formát mikroinstrukce

Mikroinstrukce dělíme podle formátu do dvou skupin. Do první

skupiny patří mikroinstrukce s pevným formátem, ve kterých je význam jednotlivých polí a bitů neměnný. Do druhé skupiny patří mikroinstrukce s proměnným formátem, ve kterých je interpretace jednotlivých polí a bitů závislá na obsahu druhých polí, a mění se podle typu instrukce. Mnohdy se mění i délka mikroinstrukce. Proměnný formát mikroinstrukce přináší úspory kapacity řídicí paměti, neboť vede ke kratším mikroinstrukcím; na druhé straně vyžaduje složitější dokódovací obvody.

9.7.2 Stupeň zakódování řídicí informace

Nejjednodušší metodou je metoda prostého přiřazení, při níž je každému řídicímu signálu přiřazen jeden bit mikroinstrukce. Tato metoda vede u složitějších systémů k neúměrně dlouhým mikroinstrukcím, proto se používá velmi zřídka. Obvykle jsou řídicí signály kódovány do nezávisle nebo současně kódovaných polí. Kódování vyžaduje však nejen dekodovací obvody, ale přináší také omezení z hlediska flexibility mikroinstrukce, neboť v jedné mikroinstrukci může být aktivován jen jediný řídicí signál ze skupiny signálů kódovaných v jediném poli. Zmenšuje se tak možnost předepsat v jedné mikroinstrukci několik paralelně probíhajících činností. Tato nevýhoda se však neprojeví, jestliže jsou v jednom poli kódovány vzájemně se vylučující signály, t.j. signály, které se nemohou vyskytovat současně. Příkladem vzájemně se vylučujících signálů jsou řídicí signály aritmetické jednotky, z nichž každý předepisuje jednu funkci této jednotky. Takové signály se nemohou vyskytovat současně, neboť aritmetické jednotce lze v jediné instrukci předepsat pouze jedinou činnost. Podobně řídicí pole obvodu posuvu na výstupu aritmetické jednotky představuje pole vzájemně se vylučujících signálů, neboť v jediné mikroinstrukci lze předepsat jediný posuv, např. aritmetický vpravo, logický vlevo a pod. Jestliže aritmetická jednotka představuje s obvodem posuvu celek, který může vykonávat buď operace posuvu nebo aritmetické operace, lze obě řídicí pole sloučit a docílit vyššího stupně kódování řídicích signálů. Jestliže však aritmetická jednotka a obvod posuvu jsou navrženy tak, aby obě operace mohly probíhat paralelně, pak by sloučení obou řídicích polí vneslo omezení, neboť by došlo ke sloučení signálů, které nejsou vzájemně se vylučující, a nebylo by možné využít možností paralelních činností, které technické pro-

středky poskytují.

9.7.3 Horizontální a vertikální mikroinstrukce

Maximální flexibilitu dovolující využít všech možností paralelních činností poskytuje formát mikroinstrukce obsahující oddělená pole vzájemně se vylučujících signálů. Každé pole je určeno pro řízení samostatné funkční jednotky nebo sběrnice. Využití možností paralelních činností, které tento formát mikroinstrukce poskytuje, vyžaduje značnou programátorskou erudici. V některých případech ani značná zkušenost nepomáhá, přesto se mikroinstrukce s tímto formátem (označované vzhledem k výrazně horizontálnímu členění mikroprogramu jako horizontální mikroinstrukce) velmi často používají. V důsledku nižšího stupně zakódování řídicích signálů a vzhledem k tomu, že pole příští adresy je explicitně vyjádřeno, má mikroinstrukční slovo obvykle větší délku, asi od 32 do 128 bitů. Formát horizontálních mikroinstrukcí je obvykle pevný.

Druhou velkou skupinu tvoří mikroinstrukce s proměnným formátem s vyšším stupněm zakódování řídicích signálů. Mikroinstrukce tohoto typu obvykle označujeme vzhledem k vertikálnímu členění jako vertikální. Adresa příští mikroinstrukce není obvykle v mikroinstrukci obsažena a posloupnost provádění mikroinstrukcí se určuje pomocí čítače mikroinstrukcí.

Proměnný formát mikroinstrukce a vysoký stupeň zakódování řídicích signálů vede k mikroinstrukčnímu slovu a poměrně malou délkou, 8 až 32 bitů, které je pro vertikálně mikroprogramované systémy charakteristická. Jedinou mikroinstrukcí lze však předepsat jedinou elementární mikrooperaci, takže nelze tímto způsobem využít takovou organizaci procesoru, která by paralelní činností dovozovala. Ve vertikálně programovatelném systému je tedy posloupnost mikrooperací interpretujících strojovou instrukcí delší než v horizontálně mikroprogramovaných systémech.

Hranice mezi vertikálně a horizontálně mikroprogramovanými systémy není sazřejmě zcela jednoznačná, a některé systémy leží svými vlastnostmi na hranici obou kategorií. Byly navrženy a realizovány systémy, které spojují výhody obou kategorií zavedením další programovací úrovně, tzv. nanoprogramu.

9.7.4 Generování adresy příští mikroinstrukce

V odst. 9.2.2 jsme se seznámili s původní koncepcí mikroprogramovaného řadiče. V tomto řadiči byla použita jedna ze základních technik pro získání adresy příští mikroinstrukce. Tato adresa se generuje na základě informace explicitně obsažené v samotné mikroinstrukci, což je charakteristické pro mikroinstrukci horizontálního typu. Větvení mikroprogramu na základě podmínek se dosahuje pomocí modifikátorů příští adresy. Nejjednodušší modifikátor pro větvení na základě jedné podmínky byl již uveden na obr. 9.6. Mikroinstrukce se složitějším formátem umožňují současné větvení podle několika podmínek a vícenásobné větvení mikroprogramu. Formát horizontálně členěné mikroinstrukce mnohdy obsahuje několik možných pokračovacích adres a jedno nebo několik polí, které předepisují podmínky, podle nichž se volí pokračovací adresa.

Druhá základní technika generace adresy příští mikroinstrukce je analogická technice uživené na úrovni instrukcí. Je založena na dohodě, podle které pořadí provádění jednotlivých mikroinstrukcí je předepsáno pořadím jejich zápisu na bezprostředně následující paměťová místa řídicí paměti. Mluvíme pak o krokovém adresování.

Mechanismus krokového adresování je řízen čítačem mikroinstrukcí, tj. adresovým registrem mikrořadiče, vybaveným prostředky pro zvýšení obsahu o jeden adresovací krok. Po ukončení mikroinstrukčního cyklu dojde automaticky ke zvýšení obsahu čítače mikroinstrukcí, takže ukazuje na adresu příští mikroinstrukce. Podobně jako na úrovni instrukcí umožňuje speciální mikroinstrukce-mikroinstrukce nepodmíněného skoku přechod na mikroinstrukci v jiném paměťovém místě než bezprostředně následujícím. Mikroinstrukce nepodmíněného skoku se vykoná zápisem adresy tvořícím její operand do čítače mikroinstrukcí.

Technika krokového adresování je charakteristická pro vertikální mikroinstrukce.

Nezávisle na typu mikroinstrukce je žádoucí, aby technické prostředky mikrořadiče dovoľovaly snadno vytvářet základní prvky struktury mikroprogramu, vyznačené na obr. 9.11. Tyto základní prvky umožňují vytvářet hierarchickou strukturu mikroprogramů (mikroprogramové moduly, víceúrovňové podprogramy, víceúrovňové podpro-

gramy obsluhy přerušení), analogické struktury programů na vyšších programovacích úrovních.

Tyto požadavky jsou v mikrořadičích respektovány např. multiplexorem na vstupu adresového registru, umožňujícím převzít příští adresu z vnějšího zdroje, zásobníkovou pamětí nebo jejím ukazatelem. V druhém případě je zásobníková paměť realizována v části řídicí paměti vhodného typu. Technické prostředky jsou tedy analogické prostředkům umožňujícím hierarchickou strukturu vyšších programovacích úrovní.

9.7.5 Mikroinstrukční cyklus

Mikroinstrukční cyklus sestává z převzetí a vykonání mikroinstrukce. Tyto dvě činnosti se mohou uskutečňovat buď postupně, takže k převzetí další mikroinstrukce dojde až tehdy, je-li vykonána běžná mikroinstrukce, nebo současně, takže k převzetí příští mikroinstrukce dochází ještě v době, kdy je vykonávána běžná mikroinstrukce. Druhá metoda, označená jako metoda překrytí fáze převzetí a vykonání mikroinstrukce, přináší časové úspory. Její nevýhodou jsou těžkosti vznikající při mikroinstrukcích podmíněných skoků, neboť podmínky pro stanovení příští adresy mohou být závislé na výsledku běžné mikroinstrukce. Tyto těžkosti se řeší obvykle tak, že se předpokládá příští adresa, na kterou se mikroprogram větví s největší pravděpodobností. Jestliže tento předpoklad se ukáže jako správný, mikroprogram pokračuje bez časových ztrát mikroinstrukcí na této adrese. Jestliže po ukončení běžné mikroinstrukce podmínky vedou na jinou pokračovací adresu, vloží se technickými prostředky mikrořadiče nevýkonný pracovní takt, v němž dojde k převzetí mikroinstrukce z adresy odpovídající podmínkám, a v následujícím taktu pokračuje vykonání mikroprogramu normálním způsobem.

Zápis mikroprogramu je většinou závislý na tom, jakou metodou se uskutečňuje převzetí a vykonání mikroinstrukcí; tato metoda patří tedy ke znakům architektury na úrovni mikroprogramu.

9.8 Struktura řídících jednotek mikroprogramovatelných mikropočítačů

Struktura mikroinstrukcí i struktura mikroprogramu je do znač-

né míry ovlivněna technickými prostředky a strukturou mikrořadiče, na kterém je mikroprogram implementován. Návrháři zůstává samozřejmě ještě značná volnost při definici formátu mikroinstrukce, např. při definici uživatelských řídicích polí. Rovněž se může vhodnou volbou technických a mikroprogramovacích prostředků vyrovnat s problémem víceúrovňových podprogramů a přerušení, i když mikrořadič neobsahuje zásobníkovou paměť. Doplněním technických prostředků je možné navrhnout systém, pracující v režimu překrývání fází převzetí a vykonání mikroinstrukce, i když je vnitřní technické prostředky mikrořadiče neobsahují. Některé parametry technických prostředků jsou však pro návrh systému určující, např. generování příští adresy, modifikace při splnění podmínky a pod.

V dalších odstavcích si popíšeme vnitřní strukturu řídicích jednotek mikroprogramovatelných mikropočítačů, které obsahují technické prostředky, které jsme souhrnně v předchozích odstavcích označovali jako mikrořadič. Základní parametry řídicích jednotek jsou uvedeny v tab. 9.11. Posoudíme jejich strukturu z hlediska možnosti implementace základních prvků struktury mikroprogramu, uvedené na obr. 9.11.

9.8.1 Řídicí jednotka Fairchild 9408

Struktura této řídicí jednotky je uvedena na obr. 9.12. Sestává z obvodů příští adresy, registru příznaků zásobníkové paměti se čtyřmi registry, registru příští adresy s inkrementačním obvodem a výstupního multiplexoru.

Řídicí jednotka může uskutečňovat podmíněné skoky na základě stavu čtyř podmínkových vstupů B uložených v registru příznaků. Jednoduché i vícenásobné podmínkové skoky se mohou uskutečňovat též na základě stavu adresových vstupů C a D.

Inkrementační obvod dovoluje implementaci mikroinstrukcí vertikálního typu. Zásobníková paměť umožňuje čtyřúrovňové vkládání podprogramů a vytváří předpoklady pro strukturní mikroprogramování.

Výstupní multiplexor přivádí na výstup adresy A buď přímo obsah obvodů příští adresy nebo obsah registrů příští adresy. V prvním případě pracuje řídicí jednotka v režimu překrývání fází převzetí a vykonání mikroinstrukce, v druhém případě v režimu bez pře-

kryvání fází.

Řídicí jednotka Fairchild 9408 je díky svému uspořádání schopna vytvořit všechny prvky struktury mikroprogramu, uvedené v obr. 9.11. Neumožňuje však stavebnicovým způsobem rozšiřovat adresovací rozsah. Rozšíření adresovacího rozsahu je možné pouze dodatečnými technickými prostředky při použití metody stránkového adresování řídicí paměti.

9.8.2 Řídicí jednotka Intel 3001

Řídicí jednotka obsahuje dva vstupy PX a SX umožňující vícenásobné větvení mikroprogramu a třetí vstup umožňující zápis podmínek do registru příznaků. Na základě jeho obsahu se mikroprogram může jednoduše větvit. Příští adresa je vždy funkcí běžné adresy mikroinstrukce. Tento přístup minimalizuje paměť mikroprogramů. Vzhledem k tomu, že neobsahuje inkrementační obvod a vzhledem k systému adresování příští mikroinstrukce je tato řídicí jednotka předurčena k implementaci mikroinstrukce s horizontálním členěním.

Řídicí jednotka Intel 3001 může přímo generovat všechny prvky struktury mikroprogramu uvedené v obr. 9.13., a výjimkou přechodu do podprogramu, pro který nemá technické prostředky, takže hierarchickou strukturu mikroprogramu lze konstruovat jen s obtížemi. Rozšíření adresovacího rozsahu je možné pouze dodatečnými technickými prostředky.

9.8.3 Řídicí jednotka Advanced Micro Devices 2909

Struktura této řídicí jednotky je uvedena na obr. 9.14. Obsahuje kromě obvodů příští adresy zásobníkovou paměť se čtyřmi registry, adresovací registr a čítač mikroinstrukcí, tvořený registrem příští adresy a inkrementačním obvodem. Obvody příští adresy obsahují vstupní multiplexor, který umožňuje volit příští adresu na základě čtyř zdrojů: přímého vstupu D, obsahu adresového registru se vstupem R, obsahu vrcholového registru zásobníkové paměti a obsahu čítače mikroinstrukcí. Inkrementační obvod má vstup a výstup přenosu, vytvářející předpoklad pro paralelní řazení řídicích jednotek. Tímto řazením lze stavebnicově rozšiřovat adresovací rozsah. Stavebnicovost mikropočítače Advanced Macro Devices řady 2900 se tedy netýká pouze šíře toku dat, ale též řídicí paměti a mikrořa-

diče.

Struktura řídicí jednotky se čtyřmi možnými zdroji příští adresy umožňuje vytvářet všechny základní prvky struktury podle obr. 9.11 pomocí řídicích signálů, které nejsou v obr. 9.14 vyznačeny. Tyto řídicí signály umožňují zápis a čtení ze zásobníkové paměti, zápis do adresovacího registru, opakování mikroinstrukce až do splnění podmínky, zápis nuly do čítače mikroinstrukcí a maskování příští adresy, přičemž jako maskovací vstup slouží přímý vstup D.

9.9 Mikroprogramovací jazyky

Vyšší programovací úrovně používají již do jisté míry ustálené systémy automatického programování. K těmto systémům patří zdrojové jazyky různých úrovní, překladače z těchto jazyků do cílového jazyka překládajícího počítače a cílový počítač vybavený operačním systémem. Systémy automatického programování na úrovni mikroprogramu se obvykle omezují na jazyk symbolických mikrokódů, překladač, nazývaný mikroassembler, který překládá symbolické instrukce zdrojového jazyka do strojového mikrokódu a odlaďovací program, usnadňující zkoušení a ladění programu.

Jazyky symbolických mikrokódů a jejich překladače určené pro vertikálně mikroprogramovatelné systémy jsou velmi podobné jazykům symbolických adres a assemblerům na úrovni strojového jazyka, neboť vertikálně kódovaný formát mikroinstrukce byl inspirován formátem instrukcí strojového kódu počítačů, takže příbuznost prostředků automatického programování je pochopitelná.

Systémy s horizontálním formátem mikroinstrukcí přinášejí však mnoho těžkostí. Překlad zdrojového jazyka do cílového jazyka sám o sobě nepředstavuje zásadní problém. Potíž spočívá ve volbě vhodné metody zápisu, která by vedla mikroprogramátora k využívání všech možností paralelního zpracování informací, které mikroinstrukce nabízejí, aniž by jej zatěžovala neúměrnými nároky na podrobnou znalost technických prostředků.

U mikropočítačů se stavebnicovou délkou slova a flexibilním uspořádáním vznikají ještě další problémy související s přizpůsobením systému automatizovaného programování konkrétním uspořádáním mikropočítače. Systém CROMIS vyvinutý firmou Intel pro mikropočí-

tače řady 3000 se s tímto problémem vyrovnává tak, že každému poli mikroinstrukce přiřazuje zvláštní mnemonický kód. Není to však jediné možné řešení. Např. firma Scientific Micro Systems vyvinula univerzální mikroassembler RAPID, který umožňuje programátoru nejdříve definovat všechny důležité parametry překladu pomocí pseudoinstrukcí. Tyto pseudoinstrukce se dělí do tří skupin:

První skupina definuje délku řídicího slova v cílovém jazyku, typy požadovaných programových výstupů, např. výpis protokolu o překladu na řádkové rychlotiskárně, děrování matričních pásek pro zápis cílového jazyka do reprogramovatelných pamětí, a rovněž je možné pseudoinstrukcemi této skupiny předepsat způsob zaplnění programem neobsazených paměťových míst řídicí paměti, popřípadě generování paritního bitu ve slovech výsledného jazyka.

Druhá skupina pseudoinstrukcí definuje formát mikroinstrukce, tj. jednotlivá řídicí pole a jejich interpretaci. V jediném programu je možné definovat několik různých formátů řídicích slov.

Třetí skupina pseudoinstrukcí definuje mnemonické kódy a jejich překlad do výsledného jazyka.

Mikroprogramátor zapíše pak svůj mikroprogram pomocí mnemonických kódů a ve formátech, které si předtím specifikoval. Mikroassembler přeloží na základě pseudoinstrukcí tento zdrojový program a pořídí všechny požadované výstupní informace.

V dynamicky mikroprogramovaných systémech není nejnižší úroveň programovacího jazyka, jazyka symbolických mikrokódů, vyhovující. Mikroprogramování složitějších úloh vyžaduje vytvořit i na této programovací úrovni jazyky obdobné jazykům makroinstrukcí, popřípadě algoritmickým jazykům jako FORTRAN a PL/1. Tyto jazyky by měly být strojově nezávislé a měly by mít srovnatelné vlastnosti, mezi něž patří především dobře definovaná syntaxe i sémantika a srozumitelnost. Problémem však zůstává návrh překladačů, které by uskutečnily dostatečně efektivní překlad, a to především pro mikroprogramovatelné systémy s horizontálním formátem mikroinstrukcí.

10. MIKROPOČÍTAČ INTEL 3000

Mikropočítač Intel 3000 je jedním z představitelů mikroprogramovatelných mikropočítačů se stavebnicovou délkou slova. Jeho základní stavebnicové prvky, se kterými se v dalších odstavcích seznámíme poněkud podrobněji, jsou uvedeny v tab. 10.1.

10.1 Řídící jednotka 3001

Řídící jednotka 3001 plní následující funkce: zajišťuje správné pořadí provádění mikroinstrukcí včetně větvení mikroprogramu na základě podmínek a zajišťuje řízení a zpracování vstupu a výstupu přenosu a posuvu operační jednotky.

Podrobné schéma je uvedeno na obr. 10.1. Věnujme se nejdříve základní funkci řídící jednotky, řízení posloupnosti mikroinstrukcí. Adresu příští mikroinstrukce vytvářejí obvody příští adresy na základě tří informačních zdrojů: obsahu sedmibitového pole příští adresy, příznaků (zapisovaných na základě výsledků aritmeticko-logických operací operační jednotky) a obsahu instrukční sběrnice (reprezentující např. operační znak strojové instrukce cílového počítače, nebo některá další pole této strojové instrukce). Pole příští adresy obsahuje zakódovanou informaci o metodě formulace příští adresy a část příští adresy. Interpretace pole příští adresy je uvedena v tab. 10.2. Z ní vyplývá, že kromě uvedených informačních zdrojů se k formulaci příští adresy používá i běžné adresy mikroinstrukce. Tato metoda přináší sice zjednodušení obvodů a zkrácení pole příští adresy, přináší však i omezení při volbě příští adresy. Příští adresu k běžné mikroinstrukci můžeme zvolit jen z předem určené podmnožiny příštích adres, dané typem skoku. Přejít na příští adresu je předepsán buď jako nepodmíněný nebo podmíněný skok, přičemž podle počtu podmínek může dojít k větvení mikroprogramu buď na dvě, čtyři nebo šestnáct příštích adres.

Vzhledem k tomu, že pravidla pro stanovení příští adresy (uvedené v tab. 10.2) jsou poměrně komplikovaná, používá se často jejich grafického znázornění. Paměťová místa řídicí paměti můžeme uspořádat do matice o 16 sloupcích a 32 řádcích. Sloupce matice jsou adresovány pomocí nižších čtyřech bitů adresy mikroinstrukce. Každý prvek matice představuje jedno paměťové místo řídicí paměti s kapacitou 512 paměťových míst. Každý typ nepodmíněného nebo podmíněného skoku můžeme znázornit v grafickém znázornění tak, že prvku matice představujícímu běžnou mikroinstrukci přiřadíme prvky, ve kterých může být uložena příští mikroinstrukce. Tak např. nepodmíněné skoky JCC, JCR mohou být uskutečněny ve stejném sloupci nebo řádku, ve kterém se nalézá běžná mikroinstrukce. (obr. 10.2) Prvek matice, reprezentující běžnou mikroinstrukci, je znázorněn v pátém sloupci a dvacátém prvním řádku. Její adresa je 155 (hex). Cílové adresy paměťových míst, na kterých může být uložena příští mikroinstrukce, leží pro nepodmíněný skok JCC v pátém, tedy ve stejném sloupci, v jakém je znázorněna běžná mikroinstrukce. Analogicky si můžeme odvodit pravidla pro znázornění nepodmíněného skoku JCR. Nepodmíněný skok JZR zvýhodňuje mikroinstrukce, uložené na adresách, ležících v nulovém řádku. Na tyto adresy lze přejít z kteréhokoliv paměťového místa. Využívají se jako počáteční adresy často se opakujících rutin. Nepodmíněný skok JCC má 32 cílových adres, skoky JCR a JZR 16 cílových adres. Nepodmíněný skok JCE má pouze osm cílových adres, které leží ve stejném sloupci a ve stejném odstavci (tab. 10.3) jako běžná instrukce. Příkaz k uskutečnění toho nepodmíněného skoku však současně aktivuje výstupy programových příznaků PRO a PR2. Toho je možno využít k modifikaci mikroinstrukce na výstupu řídicí paměti, nebo ke generování řídicích signálů. Nepodmíněný skok na adresu předepsanou obsahem instrukční sběrnice může být vyvolán aktivací signálu řízení zápisu příští adresy LD. Do devátého bitu příští adresy se v tomto případě zapisuje vždy stav 0.

Podmíněné skoky můžeme rozdělit na dvě skupiny. Do první z nich patří skoky JPR, JLL, JRL, JPX, znázorněné v obr. 10.3 a 10.4, které větvi mikroprogram na základě obsahu instrukční sběrnice a to buď přímo (JPX) na základě vyšších čtyř bitů (primární instrukční sběrnice), nebo nepřímo na základě obsahu registru programových příznaků do něhož se ukládají nižší čtyři bity instruk-

ční sběrnice (sekundární instrukční sběrnice).

Podmíněný skok JPX má 64 cílových adres. Z nich je obsahem pole příští adresy předepsána jedna ze čtyř skupin cílových adres, nalézajících se v jednom řádku. Na základě obsahu primární instrukční sběrnice je pak vybrána jedna ze šestnácti cílových adres, na které se mikroprogram může větvit. JPX současně předepíše zapis obsahu sekundární instrukční sběrnice do registru programových příznaků. Na základě obsahu tohoto registru se může mikroprogram dále větvit příkazem podmíněného skoku JPR. Podmíněný skok JPR má 128 cílových adres nalézajících se v odstavci běžné mikroinstrukce. Podobně jako v předchozím případě je obsahem pole příští adresy předepsána jedna z osmi skupin cílových adres nalézajících se v jednom řádku. Na základě obsahu registru programových příznaků se pak vybere jedna ze šestnácti cílových adres, na které se mikroprogram může větvit.

Obsah registru programových příznaků můžeme testovat též pomocí podmíněných skoků JRL a JLL. Této možnosti využíváme tehdy, je-li třeba mikroprogram větvit jen podle vyšších nebo nižších dvou bitů tohoto registru. Podmíněný skok JLL má 32 cílových adres nalézajících se v odstavci běžné mikroinstrukce; z nich je obsahem pole příští adresy předepsána jedna z osmi skupin cílových adres nalézajících se v jednom řádku. Na základě obsahu vyšších dvou bitů registru programových příznaků se pak vybere jedna ze čtyř cílových adres, na které se mikroprogram může větvit. Podmíněný skok JRL má pouze šestnáct cílových adres. Skupinu čtyř cílových adres, ve kterých se může mikroprogram větvit na základě obsahu nižších dvou bitů registru programových příznaků, tvoří opět adresy ve stejném řádku. K nepříjemným omezením při volbě cílových adres podmíněných skoků JLL a JRL patří nutnost umístit cílové adresy skoku JLL do sloupce 4 až 7, a skoku JRL do sloupce 12 až 15.

Do druhé skupiny podmíněných skoků patří JFL, JCF, JZF, které uskutečňují skok na základě podmínek, a to buď na základě stavu klopného obvodu F, nebo stavu příznaků C a Z. Podmíněný skok JFL má 32 cílových adres. Obsahem pole příští adresy je vybrána jedna dvojice adres, která se musí nalézat v jednom řádku stejných dvou odstavců (1 a 2 nebo 3 a 4) jako adresa běžné mikroinstrukce. Tato dvojice adres musí ležet ve sloupcích 2 a 3, pro adresy jejichž

nejnižší řád obsahuje šestnáctkovou číslici 0 až 7, nebo ve sloupcích 10 a 11 pro ostatní adresy. Na základě obsahu klopného obvodu F se vybere z dvojice adres, na kterou se může mikroprogram větvit. Při nesplnění podmínce, t.j. při stavu 0 klopného obvodu F, leží tato adresa ve sloupcích 2 nebo 10, při splnění podmínce ve sloupcích 3 nebo 11.

Podmíněné skoky JCF a JZF mají pouze šestnáct cílových adres. Kromě toho, že musí ležet ve stejném odstavci jako běžná adresa mikroinstrukce, platí pro ně stejná pravidla jako pro podmíněný skok JFL. Omezení volby cílových adres na sloupce 2 a 3, popřípadě 10 a 11, vyvolává rovněž potíže při přiřazování absolutních adres řídicí paměti symbolickým adresám mikroprogramu. Tyto potíže lze alespoň částečně vyloučit vhodným postupem, který si popíšeme v následující kapitole.

Dalším úkolem řídicí jednotky je zpracování vstupu příznaku FI a aktivování výstupu příznaku FO. Vstup příznaku FI je v obvyklých sestavách mikropočítače Intel 3000 připojen na spojené výstupy CO a RO operační jednotky 3001, z nichž jen jeden je aktivní. Podobně je výstup FO připojen na spojené vstupy CI a LI operační jednotky, z nichž může být aktivní pouze jeden. Během každého mikroinstrukčního cyklu se stav vstupu FI automaticky zaznamenává do klopného obvodu F. Stav tohoto klopného obvodu může být na základě obsahu pole řízení příznaků zapsán do některého z příznaků C, Z nebo do obou. Interpretace pole řízení příznaků je uvedena v tab. 10.4. Obsah tohoto pole rovněž předepisuje stav výstupu FO, který může být buď 0 nebo 1, nebo je určen stavem některého z příznaků C a Z.

Obvody zpracování příznaků umožňují větvení mikroprogramu v závislosti na výsledku aritmetických nebo logických operací uskutečněných v předcházejících mikroinstrukčních cyklech, nebo cyklů, ve kterém je větvení předepsáno.

K dalším funkcím řídicí jednotky patří zpracování přerušení. Vzorkovací signál přerušení je aktivován tehdy, předepisuje-li obsah pole příští adresy podmíněný skok JZR s cílovou adresou ve sloupci 15 (řádek 0, t.j. na adresu OOF (hex)). V případě, že řadič prioritního přerušení má nezpracovaný požadavek na přerušení, generuje na základě vzorkovacího signálu přerušení výběrový signál

adresy řádku ERA. Tímto signálem se odpojí výstupní obvod adresy řádku od adresovací sběrnice, a tuto adresu nahradí adresa mikroprogramu obsluhy přerušení. Obsah registru však se tím nezmění, takže je nutné počítat s tím, že adresa příští mikroinstrukce bude určena na základě obsahu registru adres a pole příští adresy podle tab. 10.2.

Adresovací rozsah řídicí jednotky 3001 může být rozšířen pomocí zvláštního pole mikroinstrukce a registru adresy, který slouží jako paměť rozšířené části adresy.

Z hlediska snadnější orientace je výhodné si řídicí paměť představit jako vícerozměrnou matici. Část adresy obsažená ve zvláštním registru reprezentuje adresu dvourozměrné matice, z níž každá obsahuje 512 prvků.

10.2 Operační jednotka 3002

Blokové schéma operační jednotky 3002 je uvedeno na obr. 10. 5. Základní šíře toku je 2 bity. Kaskádním řazením operačních jednotek je možné dosáhnout požadované šíře toku. Operační jednotka má šest sběrnic, z nichž čtyři jsou určeny pro komunikaci s operační pamětí s periferními zařízeními, zbývající jsou určeny jako řídicí a maskovací sběrnice.

Obsah řídicí sběrnice F určuje mikrooperaci, kterou operační jednotka provádí. Předepisuje aritmetickou nebo logickou operaci, adresu jednoho z registrů R0 až R9 a T a adresy multiplexorů A a B.

Pole řízení operační jednotky je sedmibitové a je rozděleno do dvou skupin. Tři bity skupiny F určují aritmetickou nebo logickou operaci, čtyři bity skupiny R určují zdrojové a příjmové registry. Struktura a způsob adresování dovoluje provádět střadačové operace s obsahem kteréhokoli z registrů. V tab. 10.5 je uvedena interpretace pole řízení operační jednotky. Tato tabulka je doplněna tab. 10.6, v níž je vysvětlen význam použitých symbolů. Pravidlovostní tabulky přiřazují mikrokódu zdrojový nebo příjmový registr. Kódové kombinace pole R jsou rozděleny do tří skupin. V první skupině lze adresovat libovolný z registrů operační jednotky. V druhé a třetí skupině lze adresovat pouze jeden ze střadačů, buď AC nebo T. Kódové kombinace druhé a třetí skupiny modifikují význam

pole F, jak je patrné z tab. 10.5.

Všechny další sběrnice operační jednotky jsou dvoubitové. Při kaskádním řazení operačních jednotek se odpovídající jednotlivé vstupy sběrnice F propojují. Ostatní sběrnice se podílejí dvěma bity na vytváření výsledných sběrnic. Sběrnice M se využívá pro vstup dat z operační paměti, sběrnice I pro vstup dat z periferních zařízení, sběrnice D slouží pro výstup dat. Sběrnice D spolu se sběrnicí A (na kterou je přiveden výstup adresového registru MAR) slouží především k zápisu dat do operační paměti. Obsah sběrnice A slouží k adresování paměťového místa, popřípadě k adresování periferního zařízení. Obsah sběrnice D, který je totožný s obsahem registru AC, reprezentuje informaci, která má být na tuto adresu zapsána.

Na sběrnicí K se přivádí obsah zvoleného pole řídicí paměti. Tímto způsobem se mohou předávat operační jednotce konstanty, které mohou být interpretovány jako výpočetní konstanty nebo masky. Maskování se uskutečňuje jako logický součin obsahu kteréhokoliv ze dvou vstupů multiplexoru B s obsahem sběrnice K. Výsledek logického součinu se přivede na jeden ze vstupů aritmetické jednotky. Maskováním se jednak rozšiřuje mikrokód operační jednotky, za druhé řeší problém testování jednotlivých bitů zpracovávaných slov, a za třetí řídí šíře toku dat aritmetické jednotky (délka zpracovávaných slov). Rozšíření mikrokódu operační jednotky je zřejmé z tab. 10.5. Stejnému obsahu pole řízení operační jednotky odpovídá např. pro nulový obsah sběrnice K mikrooperace ILR, a pro jedničkový obsah mikrooperace ALR. Je-li třeba přičíst jen část obsahu registru AC k obsahu registru R, pak je třeba vhodně zvolit obsah sběrnice K. K testování jednotlivých bitů se využívá obvodů přenosu. Při nearitmetických operacích, kdy přenos nemá význam, se tyto obvody využívají k vytváření logického součinu všech bitů slova, které nejsou maskovány obsahem sběrnice K, tedy k detekci nulového obsahu nemaskované části slova.

Výstupy posunu vpravo RO a výstupu přenosu CO jsou vybaveny třístavovými výstupními obvody. Při mikrooperaci posun vpravo je výstup přenosu v nevodivém stavu a výstup posunu vpravo ve vodičném stavu. Výstupní stav RO odpovídá nejnižšímu bitu střadače před uskutečněním posuvu. Při ostatních mikrooperacích je výstup RO

v nevodivém stavu a stav výstupu CO odpovídá přenosu, vzniklému při aritmetických operacích, popřípadě detekci nulového obsahu při logických operacích. Výstupy třístavových budičů ovládaných tímto způsobem je možné spolu spojit a přivést na vstup F řídicí jednotky. Na základě stavu na tomto vstupu lze pak větvit mikroprogram.

Pomocí vnějších obvodů lze podmíněně, např. na základě stavu zvláštního jednobitového pole řídicí paměti řídit vstup hodinových impulsů operační jednotky. Tímto způsobem lze předepsat mikrooperaci a tím, že její výsledek nebude zapsán do registru příjemce, ale bude pouze testován na výstupu CO nebo RO. Podmíněné řízení hodinových impulsů umožňuje nedestruktivní testování obsahů registrů a sběrnice.

10.3 Generátor zrychleného přenosu 3003

Při kaskádním řazení operačních jednotek 3002 se spojuje výstup přenosu CO operační jednotky pro zpracování nižších dvou bitů se vstupem přenosu CI operační jednotky pro zpracování vyšších dvou bitů. Výstup přenosu CO operační jednotky pro zpracování nejvyšších dvou bitů se spojí s výstupem posunu vpravo RO a přivádí se na vstup F řídicí jednotky. Vstup přenosu CI operační jednotky pro zpracování nejnižších dvou bitů se spojí se vstupem posunu vpravo LI a s výstupem FO řídicí jednotky. Toto zapojení je uvedeno v obr. 10.6 znázorňujícím jednoduchou sestavu mikropočítače Intel 3000. Přenos se v tomto zapojení zpracovává metodou postupného přenosu. Doba, potřebná ke zpracování přenosu při provádění aritmetických operací s operandy s délkou 2 n bitů, je n-násobkem doby, potřebné ke zpracování přenosu v jedné operační jednotce. Pro zpracování operandů s délkou 16 bitů je třeba kaskádně řadit 8 operačních jednotek; jestliže jedna vyžaduje ke zpracování přenosu 25 ns, vyžaduje zpracování přenosu celého slova 200 ns. K tomu je třeba připočíst časové zpoždění z datových vstupů na výstup přenosu CO jednotky zpracovávající nejnižší dva bity, což představuje dalších 35 ns.

Podstatné zkrácení doby zpracování slov s větší délkou dosáhne metodou zrychleného přenosu, spočívající v použití speciálního kombinačního obvodu, t.zv. generátoru zrychleného přenosu. Tento generátor patří k zúžené třídě obvodů typu inverze-součin-sou-

čet, jejichž logická síť je třístupňová. Obvody pro zpracování přenosu metodou postupného přenosu patří k obecné třídě obvodů a vícestupňovou logickou sítí. Každý stupeň logické sítě znamená zpoždění. Obvody patřící k zúžené třídě se vyznačují větší rychlostí zpracování informace, ale též větší složitostí. Mikropočítač Intel 3000 je vybaven generátorem předvídaného přenosu 3003, který může zpracovat přenos slova s délkou 2 n bitů, kde n může nabývat hodnot 1 až 8 - tedy s délkou 16 bitů. Pro větší délky slova je možno využít buď metody postupného přenosu pro přenos mezi skupinami šestnácti bitů, nebo víceúrovňového řazení generátorů zrychleného přenosu. Booleovy rovnice popisující generátor zrychleného přenosu jsou uvedeny v tab. 10.7, jeho zapojení je znázorněno v obr. 10.7. Výstup $C_n + 8$ je opatřen třístavovým budičem, takže je možno jej paralelně spojit s výstupem posuvu vpravo RO jako v předcházejícím případě, je však nutné sestavu mikropočítače doplnit obvody pro dekodování mikrooperace posunu vpravo v poli F řízení operační jednotky. Výstup tohoto dekodéru řídí třístavový budič signálu $C_n + 8$ tak, aby byl v době mikrooperace posunu vpravo v nevodivém stavu. Pokud nejsou časové nároky na zpracování přenosu tak kritické, postačí ke zpracování přenosu nejvyšších dvou bitů využít postupného přenosu vznikajícího v operační jednotce pro zpracování těchto dvou bitů. Pak se sice zpracování přenosu prodlouží, ale není nutné použít dekodér.

Tato metoda je znázorněna v obr. 10.7. Doba zpracování přenosu generátorem zrychleného přenosu je 30 ns a celková doba zpracování přenosu je asi 55 ns, což je doba podstatně kratší než při použití metody postupného přenosu. Použitím speciálního dekodéru je možné ji snížit na 30 ns.

10.4 Řadič prioritního přerušení 3214

Řadič prioritního přerušení předává na základě vzorkovacího signálu ISE z řídicí jednotky potvrzení požadavku na přerušení a vektor požadované úrovně přerušení. (Obr. 10.8) Je řešen tak, aby zpracoval požadavky osmi zdrojů přerušení a aby jej bylo možno kaskádním způsobem řadit pro zpracování požadavků i většího počtu zdrojů přerušení.

Požadavek na přerušení na některém ze vstupů RO až R7 se za-

píše do paměti požadavku na přerušení, není-li přerušení zakázáno stavem odpovídajícího klopného obvodu. Existuje-li několik současných požadavků na přerušení vyhodnotí dekodér priorit požadavek s nejvyšší úrovní a srovná ji s úrovní v paměti běžné úrovně přerušení. Jestliže jsou všechny podmínky pro přijetí požadavku splněny, přechází klopný obvod požadavku na přerušení do aktivního stavu a současně vyvolá zákaz přerušení jiným zdrojem nastavením klopného obvodu zákazu přerušení. K podmínkám přijetí patří především: vzorkovací signál ISE a signál povolení přerušení od řadiče s vyšší prioritou ETLG. Druhá podmínka přichází v úvahu pouze při kaskádním řazení více řadičů. Signál ENLG se aktivuje při přijetí požadavku na přerušení a přivádí na vstup řadiče ETLG s nižší prioritou od řadiče s vyšší prioritou.

Na výstupu řadiče přerušení se při přijetí požadavku na přerušení generují ještě signály IA. Současně může mikroprogram generovat řídicí signál ELR, a tím aktivovat výstupní budiče AO - A2 připojené k některé sběrnici mikropočítače. Řídicí jednotka tak může převzít informaci o zdroji a požadované úrovni přerušení. Mikroprogram na základě převzaté úrovně přerušení zapíše tuto úroveň zpět do paměti běžné úrovně přerušení pomocí signálu ECS. Zápisem běžné úrovně přerušení se ruší zákaz přerušení vzniklý nastavením klopného obvodu přerušení a blokují se požadavky na přerušení s nižší úrovní priorit než běžná úroveň.

Pro zpracování většího počtu požadavků na přerušení je možné řadit obvody 3214 do kaskády. Budiče signálů AO až A2 jsou řešeny jako budiče s otevřeným kolektorem, takže se jejich odpovídající výstupy mohou spolu spojit. Po takto vzniklé sběrnici předává nižší 3 bity vektoru přerušení ten řadič, který přerušení vyžaduje. Zapojení se doplňuje dalším řadičem přerušení, na jehož vstup se přivedou signály IA jednotlivých řadičů řazených do kaskády. Jeho výstupní signály AO až A2 představují vyšší 3 bity vektoru přerušení. Priorita jednotlivých řadičů je dána jejich zapojením v kaskádě a zajišťuje se signály ENLG a ETLG, jejichž význam byl vysvětlen v předchozím odstavci. Paměti běžné úrovně přerušení jednotlivých řadičů se doplňují vnějšími obvody, které prostřednictvím výběrových vstupů SGS jednotlivých řadičů zajišťují volbu běžné úrovně přerušení celého systému.

Vstup do mikroprogramu obsluhy přerušení může být řešen přivedením signálu IA řadiče prioritního přerušení na vstup výběrového signálu adresy řádku (ERA) řídící jednotky 3001. Signál IA převéde výstupní třístavové budiče adresy řídící paměti MA8 - MA4 do nevodivého stavu. Tímto způsobem se modifikuje adresa příští mikroinstrukce a na ní je pak umístěn mikroprogram obsluhy přerušení.

Mikroprogram obsluhy přerušení může být zapsán tak, že analyzuje obsah sběrnice A0 až A2 a podle výsledku programové analýzy zahajuje činnost podprogramů obsluhujících jednotlivé zdroje přerušení. V tomto případě je modifikátor poměrně jednoduchý - modifikuje adresu řádku na jedinou hodnotu, např. IF hex, což znamená převést signály MA8 až MA4 do stavu 1. Druhou možností je zavést mikroprogram přímo do podprogramu obsluhy zdroje přerušení bez programové analýzy. V tomto případě musí být technické prostředky řešeny tak, aby signály MA8 až MA4 byly modifikovány v závislosti na obsahu sběrnice A řadiče prioritního přerušení. V obou případech je však třeba pamatovat na to, že k modifikaci dochází jen na výstupu adresy. Obsah registru adresy se však nemění, takže formulace příští adresy podléhá pravidlům v tab. 10.2.

Další možností je použít sběrnici A k modifikaci pole příští adresy AC řídící jednotky. V tomto případě musí však dojít ke generování vzorkovacího signálu přerušení o jeden mikroinstrukční cyklus dříve. Nelze tedy využít signál ISE od řídící jednotky, ale příkaz ke vzorkování musí být obsažen v mikroinstrukci.

Zbývá podotknout, že analýzu sběrnice A mikroprogramovými prostředky je možné uskutečnit po zápisu jejího obsahu do některého z registrů operační jednotky prostřednictvím jejich sběrnic M, I, nebo K. Rovněž formulaci běžné úrovně přerušení může uskutečnit operační jednotka. Tato úroveň se pak předá na sběrnici B řadiče přerušení prostřednictvím sběrnice D operační jednotky.

10.5 Sestavy mikropočítače Intel 3000

Stavebnicové prvky mikropočítačového systému Intel 3000 dovolují vytvořit velmi rozmanité sestavy mikropočítačových systémů. Může je rozdělit podle způsobu zpracování mikroinstrukcí do dvou kategorií: Systémy bez překrývání fází a systémy s překrýváním

fází.

Seznámíme se nejprve s jednoduchou sestavou mikropočítače Intel 3000 prvně jmenované kategorie pracující v režimu bez překrývání fází převzetí a vykonání mikroinstrukce (obr. 10.6). Sestává z řídicí jednotky 3001, řídicí paměti a bloku čtyř operačních jednotek 3002, které umožňují zpracování osmibitových informačních slov. Kromě toho patří k sestavě mikropočítače operační paměť a přídatné zařízení s vazebními obvody, které nejsou v obrázku vyznačeny. Blok operačních jednotek zpracovává přenos metodou postupného přenosu. Výstup přenosu CO a posunu vpravo RO jsou spojeny se vstupem příznaku FI řídicí jednotky. Výstup příznaku FO řídicí jednotky je spojen se spojenými vstupy CI přenosu a LI posunu vpravo. Na začátku každého pracovního taktu, vymezeného hodinami systému (zdroj hodinových impulsů není též v obrázku vyznačen), se na výstupu MA řídicí jednotky zapíše adresa mikroinstrukce. Vybavovací doba řídicí paměti představuje fázi převzetí mikroinstrukce. Po jejím ukončení se na výstupu řídicí paměti objeví mikroinstrukce, jejíž formát je znázorněn v tab. 10.8. K jejím základním polím patří pole řízení operační jednotky, jež předepisuje mikrooperaci, která má být v daném taktu vykonána. Další základní pole, pole řízení příznaků, umožňuje předepsat stav vstupů LI resp. CI operační jednotky. Druhá jeho část řídí zpracování výsledku mikrooperace. Mikrooperace je určena též obsahem pole K, které je definováno uživatelem. Jeho délka může být menší, než délka slov zpracovávaných mikropočítačem. Vystačí-li např. uživatel s mikrooperacemi, pro které je v tab. 10.5 předepsán mнемonický kód, postačí jednobitová sběrnice K, jejíž obsah je přivezen na vstupy K všech operačních jednotek. Třetí základní pole mikroinstrukce je pole příští adresy, které řídí posloupnost provádění mikroinstrukcí. Další uživatelem definovaná pole jsou závislá na sestavě mikropočítače a řídí činnost operační paměti, přídatných zařízení a všech dalších technických prostředků, zařazených uživatelem do sestavy mikropočítače. V uspořádání mikropočítače podle obr. 10.6 může být v jediné mikroinstrukci předepsána aritmetická nebo logická operace a podmíněný skok, jehož podmínka je určena výsledkem této operace. V uspořádání překrývání fází odpovídá stav klopného obvodu F výsledku aritmetické nebo logické operace, která je předepsána běžnou mikroinstrukcí a v této mikro-

instrukci může být zapsán podmíněný skok JFL, který se provede až po skončení této operace. Podmíněné skoky JCF a JZF mohou předepisovat větvení na základě podmínky zapsané do paměti příznaků C a Z. Zápis do této paměti se předepisuje některou z předcházejících instrukcí, předepisujících aritmetickou operaci v poli řízení operační jednotky a uchování výsledku této operace v paměti příznaku C nebo Z v poli řízení příznaků.

V uspořádání mikropočítače pracujícího v režimu překrývání fáze převzetí a vykonání instrukce (obr. 10.7) je mezi výstup řídicí paměti a řízené vstupy jednotlivých funkčních bloků zařazen vyrovnávací registr mikrokódu, což je pro tyto systémy charakteristické. S výjimkou pole příští adresy, které je přivedeno přímo na vstup řídicí jednotky, je obsah všech ostatních polí mikroinstrukce uložen na začátku pracovního taktu do tohoto vyrovnávacího registru a řídí po celý pracovní takt vykonání mikroinstrukce. Do adresového registru řídicí jednotky se zapíše adresa následující mikroinstrukce a současně s prováděním běžné mikroinstrukce probíhá čtení řídicí paměti, tedy fáze převzetí příští mikroinstrukce. Výsledek aritmetické nebo logické operace je tedy zpožděn o jeden pracovní takt vůči podmíněnému skoku předepisujícímu větvení mikroprogramu. Podmíněný skok JFL se tedy vztahuje k výsledku aritmetické nebo logické operace, předepsané v předcházející mikroinstrukci. Podobně podmíněné skoky JCF a JZF se mohou vztahovat k výsledkům operací předepsaných v mikroinstrukcích, které předcházejí mikroinstrukci podmíněných skoků JCF nebo JZF alespoň o dva pracovní takty.

Konfigurace mikropočítače patří ke znakům architektury na úrovni mikroprogramu, neboť jej musí mikroprogramátor při zápisu mikroprogramu respektovat. Pro zvýšení rychlosti zpracování informací používá mikropočítač (obr. 10.7) metodu zrychleného přenosu, realizovanou pomocí generátoru zrychleného přenosu 3003. Zapojení řadiče prioritního přerušování 3214 není v sestavě blíže znázorněno vzhledem k množství variantních řešení, které může návrhář zvolit.

10.5.1 Časování mikropočítače

Při návrhu mikropočítačového systému musí být věnována značná pozornost správnému časování systému. Je třeba provést analýzu

všech přenosových zpoždění datových a řídicích signálů, a na základě této analýzy volit kmitočet hodinových impulsů. Uvážíme např. zpoždění, ke kterým dochází při provádění aritmetické operace v uspořádání mikropočítače podle obr. 10.7.

Pracovní takt začíná nástupní hranou hodinového impulsu (obr. 10.9). Zápis do vyrovnávacího registru mikrokódu se uskuteční se zpožděním t_1 . Dekódování obsahu pole F operační jednotku zavádí další zpoždění t_2 . Po této době jsou stavy výstupů X a Y ustálené. Zpracování přenosu generátorem předvídaného přenosu vyžaduje další dobu t_3 a zpracování přenosu operační jednotkou zpracovávající nejvyšší dva bity slova vyžaduje dobu t_4 . Kromě toho je třeba počítat s dobou předstihu t_5 pro zápis výsledku aritmetické operace do klopného obvodu F řídicí jednotky. Dále musíme uvážit dobu potřebnou pro zápis výsledku do registru operační jednotky, určenou dobou trvání hodinového impulsu t_6 . V tomto případě je minimální pracovní takt t_h určen vztahem:

$$t_h = t_1 + t_2 + t_3 + t_4 + t_5 + t_6 \quad (1)$$

Kdybychom nepoužili generátoru předvídaného přenosu a blok n operačních jednotek by využíval metody postupného přenosu, pak by vztah pro minimální pracovní tak byl změněn takto:

$$t_h = t_1 + t_2 + nt_4 + t_5 + t_6 \quad (2)$$

Jestliže by mikropočítač byl zapojen podle obr. 10.6 a pracoval bez překrývání fází, pak bychom museli ještě uvážit dobu t_7 potřebnou pro formulaci adresy řídicí paměti na výstupu MA řídicí jednotky na základě signálu AC na vstupu a vybavovací dobu paměti t_8 , takže bychom došli k výrazu:

$$t_h = t_1 + t_2 + nt_4 + t_5 + t_6 + t_7 + t_8 \quad (3)$$

platnému pro minimální pracovní takt sestavy mikropočítače z obr. 10.6 s délkou slova $2n$ bitů.

V případě, že zárojem dat jsou sběrnice M nebo sběrnice I, musíme ještě uvážit dobu potřebnou pro předání dat z přídatných zařízení nebo z operační paměti. Tuto dobu musíme připočítat

k předcházejícímu vztahu odpovídajícímu používané konfiguraci mikropočítače.

Tímto způsobem jsme určili minimální pracovní takt z hlediska zpracování dat. Zbývá však ještě porovnat tento výsledek s výsledkem analýzy zpoždění řídicích signálů. Pro určení pracovního taktu je rozhodující ten výsledek, který představuje větší omezení.

Při analýze zpoždění řídicích signálů zapojení z obr. 10.7 musíme podle obr. 10.10 uvážit dobu t_7 zpoždění od nástupní hrany hodinového impulsu na výstupy MA, vybavovací dobu t_8 řídicí paměti, dobu předstihu na vstupech příští adresy AC a dobu trvání hodinového impulsu t_6 . Nalezneme pak výraz:

$$t_h = t_6 + t_7 + t_8 + t_9 \quad (4)$$

Porovnáme pak hodnotu získanou z výrazů (1) a (4) a pracovní takt volíme na základě větší z nich.

Podle zapojení řadiče prioritního přerušení a systému přerušení je třeba časově analyzovat i pracovní takt, ve kterém je přerušení přijato, a na základě analýzy odpovídajícím způsobem prodloužit pracovní takt.

Hodnoty časových zpoždění potřebné pro výpočet předcházejících vztahů jsou uvedeny v katalogových listech jednotlivých stavebních prvků mikropočítače.

11. MIKROPROGRAMOVACÍ SYSTÉM CROMIS

Systém CROMIS je systém pro automatizaci mikroprogramování mikropočítače Intel 3000. Tvoří jej dva programy, napsané ve zdrojovém jazyku FORTRAN IV, a sice překladač jazyka symbolických mikrokódů XMAS (nevlastní mikroassembler) a zaváděcí program XMAP, generující matriční pásku pro programování pevných pamětí. Systém CROMIS je podrobně popsán v lit./1/.

Jazyk symbolických mikrokódů XMAS je strojově orientovaný zdrojový jazyk stejnojmenného assembleru. Obsahuje mnemonické kódy reprezentující funkce operační a řídicí jednotky (tab. 10.2, 10.4, 10.5) a umožňuje symbolické adresování a zavedení mnemonických kódů uživatelsky definovaných polí.

Rozdělení systému CROMIS na dva subsystémy odpovídá dvěma fázím jeho činnosti při převodu zdrojového jazyka do mikrokódu s formátem vhodným pro programování paměťových prvků řídicí paměti. První fází je překlad zdrojového jazyka do mikrokódu, druhou fází je převedení mikrokódu do vhodného formátu určeného použitými paměťovými prvky. Rozdělení systému CROMIS na uvedené subsystémy bylo motivováno předpoklady, že první fáze - překlad - bude probíhat několikrát, než mikroprogramátor odstraní chyby, a teprve pak proběhne druhá fáze - konverze výsledku do vhodného formátu.

Výsledkem překladu je protokol o překladu, obsahující zdrojové instrukce, výsledný strojový mikrokód, zprávy o chybách a grafické zobrazení řídicí paměti.

Pseudoinstrukce zaváděcího programu XMAP dovolují přiřadit mikrokód generovaný překládacím programem XMAS paměťovým místům pevných pamětí. Protokol zaváděcího programu může obsahovat zdrojové instrukce, výpis obsahu pevných pamětí a zprávy o chybách.

11.1 Jazyk symbolických mikrokódů XMAS

11.1.1. Základní prvky a objekty jazyka XMAS. Deklarace a příkazy

Jazyk symbolických mikrokódů je jazyk s volným formátem. Syntaktické prvky jazyka se mohou objevit kdekoliv ve zdrojovém zápisu a jejich pořadí je ve většině případů libovolné. Čárky a mezery mohou být používány zcela volně a jejich zařazením může mikroprogramátor zlepšit čitelnost zápisu.

Objekty jazyka symbolických mikrokódů mohou být označeny symbolickými jmény neboli identifikátory. Jako identifikátory mohou být použity jednak uživatelsky definovaná symbolická jména nebo standardní slova jazyka, která jsou uvedena v tab. 11.1. Tato standardní slova tvoří neměnnou část slovníku jazyka XMAS. Všechny identifikátory mohou být definovány jen jednou. Tentýž identifikátor nelze použít jako identifikátor adresy a jako název pole.

Zdrojový program obsahuje dva typy instrukcí: deklarace a příkazy. Deklarace jsou pseudoinstrukce určující uživatelsky definovaná pole mikroinstrukce a jim přiřazené identifikátory, mikrokód polí v příkazu explicitně neuvedených, pole sběrnice K, pole rozšířené adresy a identifikátory textových řetězců a konstant. Deklarace jsou označovány standardními slovy FIELD, IMPLY, KBUS, ADDRESS, VALUE a STRING. Příkazy jsou aktivní prvky jazyka XMAS, které se překládají ze zdrojového do výsledného jazyka, t.j. strojového mikrokódu. Každému příkazu odpovídá jedno mikroinstrukční slovo.

K základním polím mikroinstrukce (tab. 10.8) patří pole příští adresy, pole řízení příznaků a pole řízení operační jednotky. Tato pole jsou definována implicitně a jsou jim přiřazena standardní slova JUMP, FI, FO a CPE. I když mezi jednotlivými poli mikroinstrukce a to jak základními, tak uživatelsky definovanými, je funkční vztah, pokládáme je z hlediska formálního zápisu za nezávislé prvky jazyka. Pojem funkční vztah lze ilustrovat na příkladu polí CPE, FO a KBUS. Mikrooperace prováděná operační jednotkou je plně určena jen tehdy, je-li obsah všech těchto polí definován. Formálně jsou příkazy pro jednotlivá pole zcela nezávislé; mikroprogramátor musí však funkční vztah mezi jednotlivými poli při zápisu mikroinstrukce respektovat.

K definici uživatelských polí mikroinstrukce řídících např.

operační paměť, vnější zařízení nebo generující obsah sběrnice K, slouží deklarace FIELD. Například deklarace:

```
KB      FIELD      LENGTH = 8 ;
MEM     FIELD      LENGHT = 3 ;
```

určují dvě uživatelská pole s identifikátory KB a MEM. Standardní slovo LENGTH, za nímž následuje operátor přiřazení (=) a celočíselná hodnota definuje délku pole, t.j. počet bitů vyhrazených v mikroinstrukci pro toto pole. Vlevo od standardního slova FIELD je umístěn identifikátor pole. Syntaktická pravidla pro zápis zdrojového programu v modifikované Backus-Naurově notaci jsou uvedeny v tab. 11.2, použitá symbolika je vysvětlena v odstavci 11.5.

Jestliže mikroprogramátor definoval mikroinstrukci pomocí deklarací FIELD, pak každý příkaz musí obsahovat dostatečné informace, umožňující mikroassembleru předepsat mikrokód všech polí mikroinstrukce. Příkaz může být např. následující:

```
70: CPE = 23 FI = 3 FO = 0 JUMP = 37 KB = 0 MEM = 0;
```

Celočíselná hodnota oddělená dvojtečkou určuje adresu paměťového místa, na které má být uložena mikroinstrukce odpovídající příkazu. Tuto část příkazu nazýváme adresovou částí mikroinstrukce. Může obsahovat též identifikátor adresy, neboli návěští. Celočíselná hodnota uvedená za operátorem přiřazení definují mikrokód, který má být generován v poli určeném identifikátorem pole. Pořadí, ve kterém jsou zapsány příkazovací příkazy jednotlivých polí je libovolné. Volný formát jazyka XMAS připoští i následující zápis, který je zcela ekvivalentní předchozímu zápisu:

```
70: CPE = 23, KB = 0, FO = 0,
    FI = 3
    MEM = 0
    JUMP = 37;
```

K oddělení jednotlivých příkazů a deklarací slouží středník (;), takže příkaz i deklarace mohou být zapsány v několika řádcích. Hodnoty v uvedených příkazech jsou zapsány v desítkové soustavě. Jazyk XMAS připouští i zápis hodnot v šestnáctkové (H), osmičkové (Q nebo O) nebo dvojkové soustavě (B). Předchozí příkaz můžeme zapsat i takto:

```
41H: JUMP 0101101B KB = 0
      MEM = 0 CPE = 27 Q FI = 3 FO = 0 ;
```

Hodnoty s výjimkou přiřazení adresy paměťového místa, můžeme určovat též pomocí výrazu, např.:

```
JUMP = 5 + (010B SHL 4) ;
```

Závorky jsou v tomto výrazu nadbytečné, neboť operátor posuvu vlevo o čtyři místa má vyšší prioritu, než operátor sečtení, tab. 11.1. Syntaktická pravidla pro zápis výrazů jsou uvedena v tab. 11.2.

Zápis obsahu polí mikroinstrukce pomocí hodnoty předepsané přímo nebo algebraickým výrazem není nejvhodnější, neboť je zaměřena na samotný binární obsah pole, t.j. jeho mikrokód bez zřetele na jeho funkční význam. Takovýto zápis snižuje čitelnost zdrojového programu. Vhodnější formou je zápis mnemonického kódu neboli identifikátoru mikrooperace. Tento zápis spíše odpovídá zápisu jazyka symbolických adres na úrovni strojového jazyka. Jazyk symbolických kódů obsahuje standardně definované identifikátory mikrooperací pro základní pole mikroinstrukce, které jsou uvedeny v tab. 11.1. S použitím těchto identifikátorů mikrooperací namísto některých přiřazovacích příkazů základních polí, je zápis příkazu následující:

```
46 H: JMP = 25 H MEM = 0
      HCOZ FFO KB = 0 LMI (R7);
```

Samotný identifikátor mikrooperace pole CPE ještě plně nedefinuje obsah tohoto pole. Je třeba ještě definovat operand této mikrooperace, t.j. zdrojový nebo příjmový registr. Identifikátor registru (tab. 10.6) se uvádí v závorce za identifikátorem mikrooperace (tab. 10.5), takže předpis pro kód pole CPE má např. tvar LMI(R7).

Identifikátor mikrooperace v polí příští adresy JUMP specifikují některou z metod formulace adresy příští mikroinstrukce podle tab. 10.2. V příkazu musí být tento identifikátor, podobně jako identifikátor mikrooperací pole CPE, blíže specifikován. Jestliže identifikátor předepisuje nepodmíněný skok, pak musí být specifikována jedna příští adresa. Identifikátor předepisující mikrooperaci nepodmíněného skoku je tedy doplněn operandem uvedeným v závorce, specifikujícím cílovou adresu. Jestliže však identifikátor

předepisuje podmíněný skok, pak podle jeho typu musí být specifikována skupina dvou, čtyř nebo šestnácti cílových adres. Operandy předepisující cílovou adresu mohou být vyjádřeny jako výraz nebo jako návěští jiného příkazu. Jestliže není obsah pole JUMP v příkazu mikroinstrukce explicitně uveden, mikroassembler předepisuje v tomto poli nepodmíněný skok na mikroinstrukci, jejíž příkaz ve zdrojovém jazyku bezprostředně následuje.

Mnemonické kódy neboli identifikátory mikrooperací v základních polích CPE, FI, FO, JUMP jsou součástí jazyka XMAS. Mikroprogramátor však může deklarovat mnemonické kódy v uživatelských polích, a to pomocí standardního slova MICROPS, použitého při deklaraci uživatelského pole. Např. deklarace

```
MEM FIELD LENGTH = 3  MICROPS (READ = 2, WRITE = 3)
```

definuje uživatelské pole, identifikovatelné jménem MEM s délkou 3 bity. Mnemonický kód READ předepisuje pro toto pole mikrokód 010 (dvojkově) a mnemonický kód WRITE mikrokód 011 (dvojkově). V příkazu

```
24 H:  FETCH :  JMP (ACCESS)  READ  LMI (R7)  H CZ
      FFO KB = 0  CONT = 2 ;
```

nahrazuje mnemonický kód READ přiřazovací příkaz MEM = 2. V tomto příkazu byla využita možnost přiřazení několika návěstí, která jsou ekvivalentní. Toto několikanásobné přiřazení, které syntaktická pravidla jazyka XMAS připouštějí, nahrazuje pseudoinstrukci ekvivalence např. EQU, používané v jiných zdrojových jazycích. Symbolická adresa FETCH odpovídá absolutní adrese 24 (šestnáctkově). Pro pole JUMP je předepsán mnemonický kód JMP, který předepisuje nepodmíněný přechod na symbolickou adresu ACCESS, která musí být uvedena jako návěští jiného příkazu. Mnemonický kód JMP mikroassembler přeloží podle cílové adresy jako mikrokód některého nepodmíněného skoku JCC, JZR nebo JCR.

V uváděných příkladech příkazů byl explicitně buď pomocí přiřazení a nebo pomocí mnemonického kódu definován obsah všech polí mikroinstrukce. Jazyk XMAS obsahuje několik metod pro implicitní definici obsahů některých polí.

Jedna z metod spočívá v definici obsahu v příkazu explicitně neuvedených polí. Jestliže není v příkazu explicitně uvedeno někte-

ré ze základních polí, mikroassembler při překladu generuje určitý mikrokód, jehož definice je pro základní pole součástí jazyka XMAS. Tak např. není-li uvedeno pole FI, generuje mikroassembler při překladu mikrokód 11 (dvojkově). Pro uživatelsky definovaná pole lze mikrokód explicitně neuvedených polí deklarovat pomocí standardního slova DEFAULT v rámci deklarace FIELD, např.:

```
MEM FIELD LENGTH = 3 MICROPS (READ = 2, WRITE = 3)
    DEFAULT = 0;
```

Jestliže mikroprogramátor v příkazu explicitně neuvede pole MEM, buď přiřazením (např. MEM = 3), nebo mnemonickým kódem (např. READ) obsahuje mikrokód tohoto pole po překladu 000 (dvojkově), jako kdyby byl v příkazu definován.

Zvláštním způsobem je definován mikrokód explicitně neuvedeného pole sběrnice K. Toto pole spoluurčuje mikrooperace vykonávané operační jednotkou (tab. 10.5), které jsou předepisovány mnemonickými kódy mikrooperace v poli CPE. První skupina těchto mnemonických kódů implicitně definuje obsah pole sběrnice K samé nuly, druhá samé jedničky. Mikrokód explicitně neuvedeného pole sběrnice K je generován v závislosti na mnemonickém kódu pole CPE. Pole sběrnice K však musí být předem deklarováno pomocí deklarace pole sběrnice KBUS. Např. deklarace

```
KB FIELD LENGTH = 8;
KB KBUS ;
```

deklaruje pole identifikované symbolickým jménem KB s délkou 8 bitů jako pole sběrnice K. V příkazu:

```
84 H : CLR (R2) JZR (FETCH) FFO HZC READ CONT = 2;
```

je mikrokód neuvedeného pole KB implicitně definován identifikátorem mikrooperace CLR (R2) pole CPE. Mikrokód pole KB je v tomto případě 0. Jestliže však toto pole v příkazu je explicitně definováno, např. :

```
84 H : CLR (R2) JZR (FETCH) FFO HZC READ CONT = 2
    KB = OFH ;
```

pak tato definice má prioritu před implicitní definicí v poli CPE, a mikrokód pole KB je OF (šestnáctkově).

Další metodou implicitní definice obsahu polí mikroinstrukce

poskytuje deklarace IMPLY. Podobně jako základní pole CPE implicitně definuje obsah pole sběrnice K, může i uživatelské pole implicitně definovat obsah jiného pole. Deklarace IMPLY zavádí tuto implicitní definici, např.

```
CONT FIELD LENGTH = 4  MICROPS (LOAD = 7  GATE = 17 Q
      STOP = 2  RUN = 3);
      RUN IMPLY FI = 2  MEM = 2  KB = OFH;
      LOAD IMPLY MEM 4;
```

Deklarace FIELD deklaruje pole s identifikátorem CONT s délkou 4 bity a mnemonické kódy LOAD, GATE, STOP, RUN. První deklarace IMPLY definuje obsahy polí FI, MEM, KB, zavedené při použití identifikátoru mikrooperace RUN v příkazu, jestliže tato pole nejsou explicitně definována. Podobně druhá deklarace IMPLY definuje obsah pole MEM při použití identifikátoru mikrooperace LOAD. Např. v příkazu:

```
81 H : CLR (R4)  JZR (FETCH)  FFO RUN WRITE;
```

by se měl zavést pro neuvedené pole KB na základě identifikátoru mikrooperace CLR pole CPE mikrokód 0, a současně na základě identifikátoru mikrooperace RUN pole CONT mikrokód OFH. Podle prioritních pravidel zavede mikroassembler mikrokód OFH. Identifikátor mikrooperace WRITE pole MEM je explicitní definicí obsahu pole, které má přednost před identifikátorem mikrooperace RUN pole CONT, který implicitně definuje obsah pole MEM, takže mikrokód pole MEM, odpovídající příkazu, je dán kódem WRITE.

Definice obsahu pole má následující pořadí priorit:

- 1) Explicitní definice v příkazu pomocí přiřazení nebo identifikátoru mikrooperace
- 2) Použití identifikátoru mikrooperace, k němuž se vztahuje implicitní definice pomocí deklarace IMPLY
- 3) Implicitní definice obsahu pole sběrnice K, určená identifikátorem mikrooperace pole CPE
- 4) Implicitní definice v rámci deklarace FIELD pomocí standardního slova DEFAULT.

Definice 1 má nejvyšší prioritu.

Mikropočítačový systém Intel 3000 umožňuje rozšířit adresovací rozsah řídicí paměti pomocí speciálních technických prostředků,

kteřé byly probrány v kap. 10, a zvláštěního pole mikroinstrukce. Mikroprogramátor může deklarovat toto pole mikroinstrukce pomocí deklarace pole adresy ADDRESS, např.:

```
XA FIELD LENGTH = 2
XA ADDRESS ;
```

První deklarace deklaruje pole XA s délkou 2. Druhá deklarace deklaruje toto pole jako pole rozšíření adresy. Obsah tohoto pole se nemusí v příkazech definovat, ale určen absolutní nebo symbolickou adresou zapsanou jako operand pole příští adresy JUMP.

Deklarovat identifikátory číselných hodnot a konstant umožňuje deklarace hodnoty VALUE. Např.:

```
.MASK 1 VALUE OFH ;
MASK 2 VALUE NOT MASK 1 AND OFFH ;
```

deklaruje identifikátory hodnot OFH a OFFH. Identifikátor hodnoty může být užit všude, kde je povoleno zapsat výraz, např. v přiřazení pole KB v příkazu:

```
84 H : TZR (R2) KB = MASK 2 READ RUN JZR (FETCH) ;
```

ve kterém identifikátor MASK 2 určuje mikrokód pole KB.

K zjednodušení zápisu zdrojového programu je možné přiřadit řetězci symbolů identifikátor, který může zastoupit tyto symboly v příkazu, a sice pomocí deklarace řetězce STRING:

```
DONE STRING READ RUN JZR (FETCH);
```

která přiřazuje identifikátor DONE řetězci symbolů uvedených mezi uvozovkami.

Předcházející příkaz je pak možno zapsat ekvivalentním způsobem

```
84 H : TZR (R2) KB = MASK 2 DONE ;
```

Jako komentář interpretuje jazyk XMAS jakýkoliv řetězec symbolů uzavřených mezi znaky /* a */.

11.1.2 Výstup mikroassembleru XMAS

Výstupem mikroassembleru XMAS je jednak soubor dat, který je zdrojovým souborem programu XMAP a obsahuje strojový mikrokód, jednak protokol o práci mikroassembleru, který obsahuje instrukce

zdrojového jazyka, instrukce výsledného jazyka, t. j. generovaný mikrokód, seznam odvolávek na symbolické adresy a grafické zobrazení obsahu řídicí paměti. Položky protokolu jsou volitelné a jejich uvedení v protokolu si uživatel předepisuje pomocí příkazů řídicího jazyka systému CROMIS. Pomocí těchto příkazů může předepsat též stránkování, počet řádek na stránce a nadpis stránky protokolu.

Každé instrukci zdrojového jazyka je v protokolu bezprostředně přiřazena strojová mikroinstrukce, Před mikroinstrukcí je v závorkách uvedena adresa řídicí paměti (v šestnáctkové soustavě) a dvojkové zobrazení mikrokódu jednotlivých polí, počínaje polem CPE, za kterým následuje FI, FO, JUMP a uživatelské pole v pořadí, jak byla deklarována. Protokol obsahuje též výpis chyb.

Seznam odvolávek na symbolické adresy je vhodnou pomůckou při odlaďování mikroprogramu a provádění úprav a změn. Obsahuje v jednom sloupci abecedně seřazený seznam symbolických adres a v druhém sloupci pořadová čísla řádků protokolu s instrukcemi, které se na tuto symbolickou adresu odvolávají. Číslo řádku, ve kterém je symbolická adresa (návěští) definována, je uvedeno v závorce.

Grafické zobrazování obsahu řídicí paměti odpovídá představě, kterou jsme zavedli v odst. 10.1. Podle ní jsou paměťová místa reprezentována prvky matice o 16 sloupcích a 32 řádcích. Grafické znázornění v protokolu má takový počet řádek, které umožňuje zobrazit všechny adresy použité v protokolu. V každém prvku matice odpovídajícím jednomu paměťovému místu jsou zapsány následující údaje: identifikátor mikrooperace pole JUMP, šestnáctkově zapsaná cílová adresa, pořadové číslo řádku protokolu určující obsah tohoto paměťového místa a počet paměťových míst, jejichž obsah předepisuje adresu tohoto paměťového místa jako cílovou adresu. Jestliže identifikátor mikrooperace v poli JUMP předepsaný obsahem paměťového místa specifikuje podmíněný skok, pak cílová adresa zapsaná v paměťovém místě odpovídá nejnižší adrese z přípustného souboru cílových adres.

Informace uvedené v každém prvku grafického zobrazení jsou orientovány na posloupnost provádění mikroinstrukcí, neboť diagram je určen jako pomůcka pro přemísťování mikroinstrukcí při úpravách a odlaďování mikroprogramu, podobně jako seznam odvolávek na symbo-

lické adresy. Tyto pomůcky jsou velmi užitečné, neboť složitý mechanismus stanovení příští adresy, který jsme poznali v odst. 10.1, nedovoluje účinně zautomatizovat přiřazení absolutních adres symbolickým adresám. Přiřazování musí zajišťovat mikroprogramátor.

11.2 Zaváděcí program XMAP

Zdrojovým souborem programu XMAP je soubor dat generovaný mikroassemblerem XMAS. Úkolem zaváděcího programu XMAP je převést tento mikrokód do formátu odpovídajícího konkrétnímu členění řídicí paměti a pro každý paměťový prvek (integrováný obvod) generovat matriční pásku umožňující jeho programování.

Instrukce jazyka XMAP rozdělujeme na dva typy. Instrukce prvního typu popisují organizaci paměťových prvků. Instrukce druhého typu přiřazují adresy mikroprogramu adresám paměťových prvků a jednotlivé bity řídicích polí bitům výstupního slova paměťových prvků.

Syntaktická pravidla jazyka XMAP jsou uvedena v tab. 11.3. Instrukce prvního typu specifikující paměťový prvek s kapacitou 512 osmibitových slov se zapisuje pomocí standardního slova ROM:

ROM 512 by 8

Specifikace paměťového prvku je platná až do následující specifikace. Instrukce druhého typu přiřazuje adresy mikroinstrukcí adresám paměťových prvků pomocí standardního slova WORDS. Instrukce

WORDS 0 TO 513

přiřazuje adresy mikroinstrukcí 0 až 511 adresám 0 až 511 paměťového prvku. Adresy paměťových míst paměťového prvku začínají vždy adresou 0. Této adrese odpovídá vždy adresa mikroinstrukce, následující po standardním slovu WORDS. Dalšímu paměťovému místu s adresou o jednu vyšší je přiřazena adresa mikroinstrukce, která odpovídá dalšímu členu posloupnosti adres definované standardním slovem WORDS. Tato posloupnost může být i sestupná, např.

WORDS 511 TO 0

V tomto případě je adrese 0 paměťového prvku přiřazena adresa 511 mikroinstrukce, adrese 1 paměťového prvku adresa 510 mikroinstrukce atd.

Adresový rozsah udaný standardním slovem WORDS musí odpovídat

platné specifikaci paměťového prvku. Bity řídicích polí se přiřazují bitům výstupního slova paměťového prvku pomocí standardního slova BITS, u kterého se závorkách uvádí specifikace bitů. Tato specifikace má tolik položek, kolik má podle platné specifikace paměťový prvek výstupních bitů. První bit specifikovaný v seznamu se přiřadí nejvyššímu bitu výstupního slova paměťového prvku, poslední nejnižšímu. Každý bit řídicího pole mikroinstrukce se určuje pomocí identifikátoru pole (odpovídajícího identifikátoru v jazyku XMAS) a číslem bitu. Nepovinným standardním slovem INVERT se předepisuje zápis inverzního obsahu řídicích polí. Pokud je třeba zapsat inverzní pouze jediný bit, označuje se znaménkem ('). Např. instrukce

```
ROM 256 BY 4 ;
WORDS 256 TO 511 ;
BITS (0, 1, JUMP /4/, KB /2/')
```

předepisuje zapsat do paměťového prvku pole mikroinstrukcí s adresami 256 až 511 stav 0 do nejvyššího bitu, do dalšího bitu stav 1, bit 4 pole JUMP do dalšího bitu a inverzi bitu 2 pole KB do nejnižšího bitu.

Instrukce

```
WORDS 256 TO 511
```

```
INVERT BITS (0, 1, JUMP (4) KB (2)')
```

předepisuje zápis téhož obsahu, ale v jedničkovém doplňku. Výstupem programu XMAP je matriční páska buď v šestnáctkovém kódu, nebo ve formátu BNPF (v němž se nuly zobrazují znakem N, jedničky znakem P, začátek slova znakem B a konec slova znakem F) a protokol o práci programu XMAP. Formát zápisu matriční pásky i obsah protokolu se předepisuje instrukcemi jazyka.

Protokol může obsahovat následující informace: zdrojové instrukce programu XMAP; výpis binárního obsahu paměťových prvků a seznam identifikátorů bitů řídicích polí a jím odpovídající výstupní bity paměťových prvků.

11.3 Řídicí jazyk systému CROMIS

Řídicí jazyk systému CROMIS je určen pro specifikaci řídicích proměnných, které řídí provádění programů XMAS a XMAP. Tyto proměnné řídí formát výstupních pásek, formát protokolu a jeho obsah.

Význam řídicích proměnných je uveden v tab. 11.4. Příkazy řídicího jazyka jsou dvojího typu: zobrazovací příkazy a přiřazovací příkazy. Syntaktická pravidla pro jejich zápis jsou uvedena v tab. 11.5.

Zobrazovací příkazy předepisují uvedení hodnot řídicích proměnných v protokolech programů XMAS a XMAP. Příkaz k zobrazení hodnot řídicích proměnných LEFT a RIGHT má tvar:

```
% DISPLAY (LEFT, RIGHT)
```

Mají-li být zobrazeny všechny proměnné, má příkaz tvar:

```
% DISPLAY (ALL)
```

Přiřazovací příkaz přiřazuje řídicím proměnným hodnoty, např.:

```
% LEFT = 4 RIGHT = 72
```

přiřazuje řídicí proměnné LEFT hodnotu 4 a proměnné RIGHT hodnotu 72. Chceme-li přiřadit hodnotu 1 řídicím proměnným, které mohou nabývat jen dvou hodnot, postačí uvést v příkazu jen jejich jména. Např. příkaz:

```
% PRINT LEFT = 4 CODE
```

přiřazuje proměnným PRINT a CODE hodnotu jedna, proměnné LEFT hodnotu 4.

Proměnné TITLE přiřazujeme textový řetězec, např.:

```
% TITLE = 'MIKROPROGRAM TISKU'
```

Na základě tohoto příkazu je na začátku každé stránky v prvním řádku protokolu vytištěn tento textový řetězec.

Všechny řídicí proměnné musí být specifikovány před prvním zdrojovým příkazem jazyka XMAS nebo XMAP.

Proměnná MIKROMEMORY umožňuje programům XMAP a XMAS kontrolovat adresy mikroprogramu. Pokud tyto programy zjistí ve zdrojových příkazech vyšší nebo stejnou adresu, přiřazenou proměnné MICROMEMORY, vypíší v protokolu záznam o chybě.

Programy systému CROMIS kontrolují syntaktickou správnost zdrojových příkazů, a při zjištění nedostatků zaznamenávají počet chyb a druh chyby v protokolu.

11.4 Mikroprogramovací techniky

11.4.1 Deklarace STRING, IMPLY, KBUS

Vhodným využitím příkazů jazyka XMAS a vlastností řídicí a operační jednotky lze docílit efektivní zápis mikroprogramu a minimalizovat počet slov řídicí paměti, potřebný pro řešení úlohy.

Uvedeme několik příkladů mikroprogramovacích technik, které mohou nalézt širší uplatnění.

Zápis mikroprogramu usnadňují příkazy STRING, IMPLY a KBUS. Chceme-li např. zvýšit obsah registru Rn operační jednotky o jedničku a uložit výsledek do registru Rn a AC zapisujeme tuto mikrooperaci v příkazu následujícím způsobem:

```
ILR (Rn) FF1;
```

Vyskytne-li se tato mikrooperace v mikroprogramu častěji, zkrátíme zápis zavedením nového mnemonického kódu, který identifikuje tuto mikrooperaci deklarací STRING

```
INCE STRING 'FF1 ILR';
```

Pak můžeme namísto předchozího zápisu v příkazu zapsat

```
INCE (Rn);
```

Tento způsob zápisu předepisuje současně mikrooperace pole CPE, pole FO a definuje tak nové výsledné mikrooperace. Používá se v mikroprogramátorské praxi velmi často.

Podobně zápis samých jedniček do registru Rn umožňuje mikrooperace předepsaná v příkazu pomocí

```
CSR (Rn) FFO ;
```

kterou můžeme zkrátit pomocí deklarace

```
-SREG STRING 'FFO CSR';
```

Při mikroprogramování řízení procesů je velmi často nutné větvit mikroprogram podle stavu některého bitu slova v registru Rn, sběrnici I nebo sběrnici M. K tomuto testování je vhodné deklarovat následující mikrooperace:

```
TSTR STRING 'FFO TZR';
```

jejíž funkční popis je následující:

$Rn \wedge K \rightarrow Rn$ $Rn \wedge K \rightarrow CO$,
 TSTR STRING 'FFO LTM';

její funkční popis je následující:

$M \wedge K \rightarrow AT$ $M \wedge K \rightarrow CO$

popřípadě

TSTI STRING 'FFO ORI'

jejíž funkční popis je následující:

$I \wedge K \quad AT \rightarrow AT$ $I \wedge K \rightarrow CO$

Ve všech těchto mikrooperacích může být předepsán deklarací KBUS takový obsah pole sběrnice K, aby maskoval s výjimkou zvolených bitů obsah slova v registru Rn, sběrnici I nebo sběrnici M. Nulový obsah slova pak může být testován pomocí stavu klopného obvodu F, který zaznamenává stav výstupu FO.

Deklarace IMPLY umožňuje implicitní deklarace obsahu polí, takže v příkazu nemusí být všechna pole obsažena.

11.4.2 Přiřazování absolutních adres symbolickým adresám mikroinstrukcí

Složitý mechanismus odvození příští adresy nedovoluje účinně automatizovat proces přiřazení absolutních adres řídicí paměti symbolickým adresám mikroprogramu, takže tuto činnost musí zajišťovat mikroprogramátor.

Zpočátku zapisujeme mikroprogram jako posloupnost mikroinstrukcí v logickém sledu, ve kterém budou vykonávány při provádění mikroprogramu bez ohledu na pozdější zavedení do řídicí paměti. Jestliže se má posloupnost mikroinstrukcí vykonat ve sledu, ve kterém je zapísána ve zdrojovém jazyku, nemusíme obsah pole JUMP explicitně definovat a místo toho můžeme použít obecné mikroinstrukce nepodmíněného skoku JMP; program XMAS přidělí tomuto poli takový mikrokód, který umožní přejít na následující mikroinstrukci. Jestliže se mikroprogram větví, pak musíme předepsat identifikátor podmíněného skoku (JFL, JCF, JZR, JPR, JLL, JRL nebo JPX). Přitom musíme vzít v úvahu počet možných cílových adres každého typu podmíněného skoku. Cílové adresy označíme identifikátory, které rovněž zapíšeme jako návěští mikroinstrukce, na kterou může mikroprogram přejít.

Po zápisu mikroprogramu v symbolické formě musíme doplnit zdrojové příkazy jazyka XMAS absolutními adresami mikroinstrukcí. Jako pomůcka je použit seznam odvolávek na symbolické adresy a stavový diagram mikroprogramu. Seznam odvolávek lze pořídit při prvním překladu mikroprogramu na základě řídicího příkazu systému CROMIS. Stavový diagram mikroprogramu je orientovaný graf, jehož uzly odpovídají mikroinstrukcím a orientované hrany přechodům mezi mikroinstrukcemi. Ve stavovém diagramu se uvádí ke každému uzlu symbolická adresa a každému větvení typ podmíněného skoku, jak znázorňuje obr. 11.1. Další pomůckou je diagram s políčky uspořádanými do 16 sloupců a 32 polí, do něhož můžeme zaznamenávat zaplnění řídicí paměti.

Při přiřazování absolutních adres se doporučuje následující postup: Nejprve se přiřadí absolutní adresy, které jsou dány požadavky technického vybavení (např. první mikroinstrukce inicializačního mikroprogramu musí být umístěna na adrese 0, nebo první mikroinstrukce mikroprogramu obsluhy přerušování na adrese 15).

Dále se přiřadí absolutní adresy symbolickým adresám, které představují cílové adresy podmíněných skoků a pak symbolickým adresám mikroinstrukcí, které se na tyto adresy odvolávají. Je třeba vzít v úvahu omezení, zřejmě z obr. 10.2 a 10.4, která platí pro cílové adresy a jejich počet pro každý skok. V řídicí paměti s kapacitou 512 slov jsou jen 64 cílové adresy pro skoky JCF, JZF, JFL. Každý podmíněný skok JPX nebo JPR vyžaduje při každém použití jeden řádek možných cílových adres. Nakonec se přiřadí adresy mikroinstrukcí, které v mikroprogramu tvoří posloupnosti bez podmíněného větvení. Adresy v řádku 0 musí být přiřazovány s rozmyslem, neboť jsou dosažitelné z kteréhokoliv místa paměti. Skok na řádek 0 (JZR) se používá např. tehdy, jestliže několik větví mikroprogramu má společné pokračování. První mikroinstrukce tohoto pokračování je umístěna v řádku 0.

11.4.3 Pseudopodprogramy

Základní technické prostředky mikropočítače Intel 3000 nedovolují vytvářet hierarchické struktury mikroprogramů. S tímto nedostatkem se můžeme v některých případech alespoň částečně vyrovnat buď přechodem na společné pokračování nebo vytvářením pseudo-

podprogramů.

Problém přechodu na společné pokračování se může řešit pomocí nepodmíněného skoku JZR. Pokud však tento řádek nemá volné adresy, je možné předepsat skok JCF nebo JZF a předem nastavit příznak C nebo F do dohodnutého stavu. Cílové adresy skoků JCF a JZF jsou dosažitelné ze 64 paměťových míst, které jsou zřejmé z obr. 10.4 nebo tab. 10.2.

Pseudopodprogramy lze vytvářet tak, že při vstupu do pseudopodprogramu se nastaví příznak Z nebo C do stavu, který odpovídá požadovanému logickému pokračování. Pokud pseudopodprogram tyto příznaky nepoužívá, je možné po návratu z pseudopodprogramu předepsat čtyři pokračování. Je samozřejmé, že tato metoda má celou řadu omezení, ale mnohdy je její aplikace užitečná. Podobným způsobem lze použít i obsah programových příznaků.

11.4.4. Příklad mikroprogramu

V tab. 11.6 je uveden příklad mikroprogramu pro násobení dvou šestnáctibitových čísel. Předpokládáme, že jde o systém s překrýváním fází. Násobenec je obsažen v registru, jehož výstup je přivezen na sběrnici M, násobitel je obsažen v registru T.

Deklarační část mikroprogramu je na řádcích 1 až 12. Deklaruje se pole sběrnice K s identifikátorem KB a identifikátory mikrooperací tohoto pole. Tyto mikrooperace se používají při deklaracích mikrooperací operační jednotky. Zahrnují mikrooperace polí CPE, FO a KB, které předepisují souhrnně činnost operační jednotky.

Inicializační část mikroprogramu je uvedena na řádcích 13 až 15, vlastní smyčka násobení na řádcích 16 až 21. Násobitel je uložen v registru T, a v tomto registru se též postupně ukládá výsledek. Algoritmus násobení je zřejmý z komentářů uvedených v tab. 11.6. Stavový diagram a diagram obsazení řídicí paměti mikroprogramu násobení je uveden v obr. 11.1.

11.5 Modifikovaná Backus-Naurova notace

Backus-Naurova notace patří k formálním systémům používaným k definici syntaxe programovacích jazyků. Jakýkoliv jazyk je možné chápat jako množinu vět nebo řetězců symbolů s dobře definovanou

strukturou a jistým významem. Pravidla, která definují platné konstrukce jazyka, t.j. platné symboly a jejich řetězení, nazýváme syntaktickými. Interpretaci symbolů a jejich řetězců určují sémantická pravidla.

Nejjednodušším prvkem jazyka je symbol. Symboly se spojují do řetězců symbolů nebo vět, které mohou nebo nemusí patřit do jazyka. Platnými větami jazyka jsou jen určité řetězce symbolů. Backus-Naurova notace formalizuje proces generování platných vět jazyka; lze ji chápat jako soustavu pravidel, podle kterých lze vytvářet platné věty jazyka. Patří mezi t.zv. syntaktické meta-jazyky, t.j. jazyky používané k popisu jiného jazyka. Popisovaný jazyk se nazývá objektovým jazykem. Jeho symboly se nazývají terminální. Symboly meta-jazyka, které označují řetězce v objektovém jazyku a reprezentují mezistavy v procesu generování se nazývají neterminální. Terminální symboly jsou v Backus-Naurově notaci totožné se symboly objektového jazyka, neterminální symboly se uvádějí v závorkách $\langle \rangle$. Znak $::=$ má význam „je možné nahradit“. Alternativní možnosti daných symbolů se oddělují vertikálními čarami.

Tak např. pravidlo:

$\langle \text{příkaz} \rangle ::= \langle \text{adresová část} \rangle \langle \text{operační část} \rangle$

se čte „symbol příkaz lze nahradit symbolem adresová část, na níž následuje symbol operační část“.

Pravidlo:

$\langle \text{desítkové číslo} \rangle ::= 0|1|2|3|4|5|6|7|8|9$

se čte „symbol desítkové číslo lze nahradit kterýmkoliv z terminálních symbolů 0, 1, 2 až 9“.

Modifikovaná Backus-Naurova notace používá další znaky, kterými eliminuje používání rekursivního zápisu. Znaky reprezentují libovolný nenulový počet opakování předcházející symbolu. Tak např.:

$\langle \text{desítková celočíselná hodnota} \rangle ::= \langle \text{desítkové číslo} \rangle \dots$

se čte „symbol desítková celočíselná hodnota lze nahradit libovolně dlouhou posloupností symbolů desítkové číslo“.

Dále lze uzavírat symboly do závorek. Složené závorky uzavírají symboly, na něž se vztahují předchozí pravidla tak, jako kdyby se jednalo o jediný symbol. Např. pravidlo:

$\langle \text{seznam implicitních definic} \rangle ::= \{ \langle \text{identifikátor pole} \rangle = \langle \text{výraz} \rangle \}$

se čte „symbol seznam implicitních definic lze nahradit libovolně dlouhou posloupností : symbol identifikátor pole = symbol výraz“.

Jestliže jsou symboly zapsány ve složených závorkách vertikálně nad sebou, pak lze z takto uvedených symbolů využít jen jediný, např.:

$$\langle \text{výraz} \rangle ::= \langle \text{term} \rangle \left\{ \begin{array}{c} + \\ \text{OR} \\ \text{XOR} \end{array} \right\} \langle \text{term} \rangle$$

lze ekvivalentně zapsat takto:

$$\langle \text{výraz} \rangle ::= \langle \text{term} \rangle + \langle \text{term} \rangle | \langle \text{term} \rangle \text{ OR } \langle \text{term} \rangle | \langle \text{term} \rangle \text{ XOR } \langle \text{term} \rangle$$

Hranaté závorky v modifikované Backus-Naurově notaci uzavírají volitelné symboly. Jsou-li zapsány v závorkách nad sebou, lze pro výslednou konstrukci využít jen jediný. Např. pravidlo

$$\langle \text{identifikátor} \rangle ::= \langle \text{písmeno} \rangle \left[\begin{array}{c} \langle \text{písmeno} \rangle \\ \langle \text{číslice} \rangle \end{array} \right] \dots$$

říká: vezmeme-li v úvahu definici neterminálních symbolů písmeno a číslice v tab. 11.2, k platným identifikátorům patří např.:

A

BODE

LOOP3

Nepatří k nim např. identifikátory:

3

3AB

ABO

které uvedeným definicím odporují.

12. EMULACE

V této kapitole se budeme zabývat využitím mikropočítače 3000 jako hostitelského systému a uvedeme dva příklady. V prvním z nich je cílovým systémem hypotetický počítač, jehož architektura je záměrně volena tak, aby byla blízká hostitelskému. Při emulaci lze proto v podstatě vystačit se základními technickými prostředky hostitelského systému. Podrobněji je tento příklad emulace uveden v /2/. V druhém uváděném příkladu jsou architektury hostitelského a cílového systému mikropočítače 8080 zcela rozdílné, což vede k jinému pojetí emulace. Uspořádání hostitelského systému je v takovýchto případech nutno přizpůsobit uspořádání cílového počítače, neboť jinak lze těžko dosáhnout přijatelných parametrů emulace. Z druhého příkladu je zřejmé, že mikropočítač 3000 splňuje požadavky na univerzální hostitelský systém jen částečně.

Uváděné příklady emulace slouží jen k bližšímu vysvětlení pojmů z předcházejících kapitol. Proto je zadání architektury hypotetického počítače neúplné a pokud se týká souboru instrukcí, omezuje se např. jen na skupiny instrukcí s přímým operandem a s pamětovou referencí, pro které je uveden mikroprogram emulace. Rovněž popis emulace mikropočítače 8080 se podrobněji zabývá emulací jen jediné instrukce.

12.1 Emulace hypotetického cílového počítače

12.1.1 Architektura cílového počítače

Cílový počítač má základní šíři toku dat 16 bitů. Formát instrukce je uveden v obr. 12.1. Procesor obsahuje následující registry: čítač instrukcí P, ukazatel zásobníku S, střadač A, indexový registr X, bazový registr dat B, bazový registr programu E a registr stavového slova W s příznakem přenosu v nejvyšším bitu.

Operační kód cílového počítače obsahuje instrukce s pamětovou referencí, jednoslovné nebo dvouslovné instrukce s přímým operandem (tab. 12.1 až 12.3), skokové instrukce, vyvolávací instrukce, přesunové instrukce, instrukce pro uložení a převzetí bytu a vstupně výstupní instrukce. V instrukcích mohou být použity různé adresovací metody: přímé adresování, nepřímé adresování, indexové adresování, relativní adresování a jejich kombinace.

Instrukce s pamětovou referencí a instrukce s přímým operandem jsou uvedeny v tab. 12.1. Kromě identifikátoru operačního znaku a významu instrukce je v tabulce uveden i kód pole operačního znaku, který je volen tak, aby byl optimální s hlediska implementace v hostitelském systému. V případě, že by byl tento kód předem zadán, znamenalo by to pravděpodobně zařazení dalších mikroinstrukcí do mikroprogramu a snížení účinnosti emulace. Adresovací metody, které lze v těchto dvou skupinách použít, jsou uvedeny v tab. 12.2. Instrukce s přímým operandem jsou jednoslovné tehdy, jestliže pole posunutí má nenulový obsah. V případě, že obsah pole posunutí je nulový, je instrukce interpretována jako dvouslovná a přímý operand je obsažen ve druhém slovu.

Adresovací metoda použitá v instrukci je určena obsahem pole modifikátoru, jehož interpretace je uvedena v tab. 12.3. V této tabulce je rovněž uveden způsob vykonání instrukce, který spočívá ve výpočtu skutečné adresy a ve vykonání instrukce předepsané polem modifikátoru a polem operačního znaku.

12.1.2 Mikroprogram emulace

Architektura cílového počítače je emulována hostitelským systémem mikropočítače Intel 3000 na úrovni mikroprogramu. Mikroprogram implementující cílový počítač obsahuje dvě části - deklarační a příkazovou. V deklarační části (tab. 12.4) se deklarují pole KB sběrnice K a uživatelské pole MCF a identifikátory mikrooperací těchto polí. Sestava mikropočítače Intel 3000 musí být vybavena odpovídajícími technickými prostředky, umožňujícími uskutečnění těchto mikrooperací. Pro další popis mikroprogramu nás především zajímají mikrooperace RRM a RWM pole MCF pro řízení štyku s operační pamětí. Snadnější zápis příkazové části mikroprogramu dovolující přímé odvolávky na identifikátory registrů cílového počítače umožňuje deklarace textových řetězců. Implicitní deklarace ob-

sahu pole FFO při mikrooperaci SDR dovoluje tuto mikrooperaci používat při zápisu mikroprogramu jako mikrooperací uložení obsahu střadače AC do zvoleného registru Rn, který v mikroprogramu reprezentuje většinou některý z registrů cílového počítače.

Vývojový diagram mikroprogramu je uveden v obr. 12.1 a 12.2. Každý příkazový blok je označen symbolickou adresou mikroinstrukce, kterou se zahajuje jeho činnost. První příkazový blok obsahuje inicializační mikroprogram, ve kterém jsou uskutečněny všechny činnosti počítače po připojení napájecích zdrojů. Další blok příkazu je převzetí cílové instrukce. Příkazové bloky, na které se blok převzetí cílové instrukce větví obsahují výpočet skutečné adresy operandu, popřípadě výpočet nebo převzetí přímého operandu. Tyto údaje se ukládají do střadače AC. Skutečná adresa se pak využívá následujícím způsobem:

Před vykonáním instrukce podskupiny 1 nebo podskupiny 2 se z paměťového místa operační paměti určeného skutečnou adresou převezme operand této instrukce. Před vykonáním instrukce podskupiny 4 se z paměťového místa operační paměti určeného skutečnou adresou převezme adresa, která slouží jako adresa paměťového místa ML používaného při vykonání této instrukce. Tato adresa se proto ukládá do adresového registru MAR. Pro vykonání instrukcí podskupiny 3 se použije přímo skutečná adresa.

Obr. 12.2 obsahuje příkazové bloky cílových instrukcí. Po vykonání předepsaných činností se vrací zpět na začátek bloku převzetí cílové instrukce.

Mikroprogram inicializace je uveden v tab. 12.5. Realizuje následující činnosti, předepsané při inicializaci cílového počítače: vynulování registrů W, A, X, uložení obsahu paměťového místa s adresou do registru S, obsahu paměťového místa s adresou 1 do registru B, obsahu paměťového místa s adresou 2 do registru E a konečně obsahu paměťového místa s adresou určenou obsahem registru E do registru P. Při zápisu mikroprogramu se předpokládá, že tato poslední činnost bude společná s úsekem mikroprogramu implementujícím skokové instrukce cílového počítače začínajícím návěštím XRTN, který není dále rozpracován.

Další bloky vývojového diagramu z obr. 12.1 jsou realizovány mikroprogramem uvedeným v tab. 12.6. Mikroinstrukce s návěštím

FETCH v řádce 1 a další mikroinstrukce na řádcích 2 a 3 odpovídají bloku převzetí cílové instrukce. Podle obsahu pole modifikátoru se mikroprogram větví na šestnáct možných pokračování, z nichž podrobněji vysvětlíme pokračování s návěštími IMMD, DMRF, IMRF, IXMD, IRBM, jež odpovídají cílovým instrukcím s pamětovou referencí a přímým operandem (tab. 12.3). Obsah pole operačního znaku se automaticky ukládá do paměti programových příznaků a obsah pole posunutí D do registru R9.

Mikroinstrukce na řádcích 6 až 9 zpracovávají přímý operand. Obsah pole posunutí byl testován mikroinstrukcí na řádce 2, v případě nulového obsahu pole posunutí pokračuje mikroprogram mikroinstrukcí s návěštím IMMS na řádce 9, v opačném případě mikroinstrukcí s návěštím IMML na řádce 7. Rozhodování se uskutečňuje na základě stavu příznaku Z, do kterého byl výsledek testu zaznamenán.

V instrukcích na řádcích 8 a 9 po převzetí přímého operandu mikroprogram větví podle nižších dvou bitů pole operačního znaku na podskupinu cílových instrukcí 1 (návěští ILGA) nebo na podskupinu cílových instrukcí 2 (návěští ILPX). Uskutečnění instrukce ADA je uvedeno na řádcích 14 až 17. Mikroinstrukce s návěštím ADA 1 na řádce 14 obsahuje podmíněný skok na základě přenosu vzniklého v předchozí operaci. Jestliže přenos vznikl, pokračuje mikroprogram mikroinstrukcí NCY; obě mikroinstrukce pak mají společné pokračování mikroinstrukcí s návěštím FETCH, kterým začíná blok převzetí další cílové instrukce. Podobně se provádějí další cílové instrukce podskupiny 1 nebo 2.

Na řádce 27 je uvedena mikroinstrukce s návěštím DMRF, kterou se zahajuje zpracování cílových instrukcí s pamětovou referencí s přímým adresováním. Mikroinstrukce na řádce 27 a 28 vypočítávají skutečnou adresu a větví mikroprogram podle nižších dvou bitů operačních znaků na podskupiny 1 až 4 cílových instrukcí. Před jejich vykonáním se uskuteční činnosti předepsané bloky s návěštími MRV1, popřípadě MRV2, MRAD, STRG. Např. mikroinstrukce na řádce 44 s návěštím MRV1 a následující mikroinstrukce připravují data pro cílové instrukce podskupiny 1 tak, že přečtou obsah pamětového místa adresovaného registrem AC (t.j. skutečnou adresou) a uloží jej zpět do registru AC. Pak pokračuje mikroprogram vykonáním některé z cílových instrukcí poskupiny 1 v závislosti na obsahu vyšších dvou bitů

operačního znaku.

Výpočet skutečné adresy pro další adresovací metody je obdobný. Za zmínku stojí pseudopodprogram tvořený mikroinstrukcemi na řádcích 30, 31. Při nastavení příznaku C pokračuje jeho vykonávání mikroinstrukcemi na řádce 33, v opačném případě na řádce 32. Podle toho se liší výpočet skutečné adresy při nepřímém adresování a ostatních adresovacích metodách.

Mikroprogram emulace dalších cílových instrukcí lze sledovat s pomocí podrobného komentáře v tab. 12.6 a vývojového diagramu v obr. 12.1 a 12.2.

12.2 Emulace mikropočítače Intel 8080

Dalším příkladem emulace je emulátor mikropočítače Intel 8080, který uvedla na trh firma Signetics. Hostitelský systém je vytvořen pomocí stavebnicových prvků mikropočítače Intel 3000.

12.2.1 Architektura cílového počítače

Cílový počítač je vytvořen technickými prostředky, které jsou uvedeny na obr. 12.3. Kromě samotného mikroprocesoru 8080 obsahuje zdroj hodinových impulsů 8224, systémový řadič 8228, dva obousměrné vazební obvody 8226 a dva budiče sběrnice 8212. Spolu s operační pamětí, která není v obr. vyznačena, tvoří tyto technické prostředky cílový mikropočítač s instrukčním cyklem 2 μ sec. Předpokládáme, že s architekturou tohoto cílového mikropočítače je čtenář v dostatečném rozsahu seznámen; podrobnosti může nalézt v předcházejících kapitolách.

12.2.2 Uspořádání hostitelského systému

Technické prostředky hostitelského systému emulátoru Signetics jsou uspořádány podle obr. 12.4 a sestávají ze tří sekcí: řídicí, aritmetické a vazební.

Hlavním úkolem aritmetické sekce je vykonávat aritmetické a logické operace se slovy s jednoduchou a dvojnásobnou délkou. Hostitelský systém je tak přizpůsoben cílovému mikropočítači, jehož většina instrukcí je sice orientována na osmibitová slova, avšak celá skupina instrukcí, které se označují jako instrukce pro práci s

dvojicemi registrů, je orientována na slova šestnáctibitová. Rovněž některé víceslovné instrukce, v jejichž operační části vystupují operandy adresy vyžadují při vykonání provádět aritmetické operace s touto šestnáctibitovou adresou.

Aritmetická sekce hostitelského systému obsahuje dva v podstatě nezávislé bloky (tvořené operačními jednotkami 3002), z nichž každý má šíři toku osm bitů. Zpracování osmibitových slov uskutečňuje vždy první blok operačních jednotek. Při zpracování šestnáctibitových slov uskutečňovaném např. při aktualizaci obsahu čítače instrukcí, ukazatele zásobníku a ukazatelů adresy operační paměti cílového mikropočítače, pracují oba dva bloky společně, takže výsledná šíře toku je šestnáct bitů. Jednobytové registry operační jednotky hostitelského systému jsou přiřazeny registrům cílového mikropočítače podle tab. 12.7. V tabulce jsou použity běžné identifikátory registrů jak hostitelského, tak i cílového mikropočítače. Čtyři z registrů hostitelského mikropočítače zůstaly nevyužity, takže jsou k dispozici pro rozšíření emulátoru. K zvýšení účinnosti emulace je střadač A cílového mikropočítače emulován dvěma registry, a to registrem T 1. bloku a registrem T 2. bloku. Pro zkrácení pole masky, které slouží především pro maskování polí cílových instrukcí s proměnným formátem slouží dekodér masky. Úlohu dekodéru generujícího 42 potřebných maskovacích osmibitových slov plní pevná paměť, jejíž výstup je přiveden na vstupy dat I operačních jednotek 3002. Styk mezi oběma bloky usnadňuje vstupní multiplexor operačních jednotek, který dovolu je volit jeden ze tří informačních zdrojů sběrnice K, a to buď sběrnici D jednoho z bloků nebo výstup instrukčního registru nebo výstup obousměrného budiče datové sběrnice. Kromě toho lze obsahem jednoho z polí mikroinstrukčního slova předepsat buď invertování obsahu adresovaného vstupu multiplexoru nebo vnutit na výstup multiplexoru stav 00 (hex) nebo stav FF (hex). Na sběrnici M prvního bloku operačních jednotek je připojen registr stavového slova PSW, který odpovídá registru PSW cílového mikropočítače. Nese informaci o všech příznacích C, Z, P a AC. Stav těchto příznaků se předepisuje na základě vyhodnocení výsledků aritmetických nebo logických operací vnitřními technickými prostředky operačních jednotek s výjimkou příznaků Z a P, pro jejich vyhodnocení je aritmetická sekce vybavena vnějšími prostředky (bloky Z a P v obr. 12.4).

Řídicí sekce generuje posloupnosti mikroinstrukcí emulující cílové instrukce, doplňkové instrukce a činnosti spojené se základním instrukčním cyklem cílového mikropočítače.

Vzhledem k proměnnému formátu cílové instrukce by bylo časově náročné a neekonomické dekodovat cílovou instrukci pomocí základních technických prostředků hostitelského mikropočítače Intel 3000. Tento postup vedl k cíli v předcházejícím příkladu emulace uvedeném v odst. 12.1 a 12.2, ve kterém pevný formát zvolené instrukce cílového počítače zcela vyhovoval pro přímé dekodování základními technickými prostředky hostitelského systému. Pro dekodování cílových instrukcí mikropočítače Intel 8080 je však výhodnější doplnit základní technické prostředky o dekodér cílových instrukcí. Tento dekodér zvyšující podstatně účinnost emulace je tvořen pevnou pamětí s kapacitou 512 osmibitových slov. Její obsah je rozdělen do dvou částí. V první části na adresách 0 až 255 je obsažena tabulka pro primární dekodování instrukce. Pomocí této tabulky se dekodují přímo všechny cílové instrukce i doplňkové instrukce násobení a dělení. Jestliže však skupina cílových instrukcí sdílí společné úseky mikroprogramu, pak při návratu do vykonávání jednotlivých mikroinstrukcí se používá druhé části paměti s adresami 256 až 511, ve kterých se přitom znovu dekoduje operační kód zapsaný v instrukčním registru. Výstupní slovo dekodéru cílových instrukcí je přivedeno na primární a sekundární instrukční sběrnici PX a SX řídicí jednotky 3001. Pomocí mechanismu vícenásobného větvení, popřípadě přímo zápisem obsahu těchto sběrnic pomocí signálu LD lze přejít na první adresu mikroinstrukce mikroprogramu emulujícího dekodovanou cílovou instrukci. Mikroprogram emulace všech cílových mikroinstrukcí a doplňkových mikroinstrukcí násobení a dělení je zapsán v řídicí paměti s kapacitou 512 x 48 bitů. Z toho je skutečně obsazeno jen asi 350 adres; zbylé adresy se mohou využít k dalšímu rozšíření.

Generace příští adresy mikroinstrukce se uskutečňuje převážně technickými prostředky řídicí jednotky 3001. K větvení na základě vnější stavové informace jsou tyto prostředky doplněny vnějšími odvody větvení, které dovolují výhodnější zpracování cílové instrukce RESTART a dalších. Registr mikrokódu svědčí o tom, že hostitelský systém pracuje na úrovni mikroprogramu v režimu překrývání fáze výběru a převzetí mikroinstrukce.

Vazební sekce obsahuje prostředky pro komunikaci se všemi sběrnicemi. Styk s adresovou sběrnicí zprostředkuje svými výstupy adresový registr operační paměti, který je svými vstupy připojen na sběrnice A operačních jednotek. Jeho obsah je tedy určen obsahem vnitřního adresového registru MAR operačních jednotek. Styk s datovou sběrnicí zajišťuje obousměrný budič datové sběrnice, který má dostatečně velký logický zisk, aby dovolil rozšiřování systému. Pokud obsah datové sběrnice představuje instrukční slovo, ukládá se do instrukčního registru a dekoduje se buď dekodérem cílových instrukcí, nebo dekodérem adres registrů. Obsah instrukčního registru se v některých případech zpracovává aritmetickou sekcí.

Vstupní signály řídící sběrnice se zpracovává vstupní registr řídící sběrnice, který plní úlohu střadačového registru. Výstupní signály generuje pevná paměť, která tvoří nejdůležitější součást výstupních obvodů řídící sběrnice, jejichž podrobnější schéma je uvedeno na obr. 12.5.

12.2.3 Struktura mikroinstrukčního slova

Mikroinstrukční slovo má délku 48 bitů. Jeho formát je pevný a určuje rozdělení do jednotlivých polí různých délek. Mikroinstrukční slovo obsahuje celkem 25 polí, která jsou přehledně uvedena v tab. 12.8. Význam polí je uveden v tabulce a vysvětlíme jej v dalším textu podrobněji.

Obsah pole LD je přiveden přímo na stejnojmenný vstup řízení zápisu příští adresy (obr. 10.1). Je-li toto pole aktivní, pak se obsah instrukční sběrnice zapisuje do registru adresy. Vzhledem k tomu, že je sběrnice osmibitová, zapisuje se do devátého bitu příští adresy bit 0, takže lze tímto způsobem předepsat adresu paměťového místa v rozsahu 0 - 255. Další pole LD2 a doplňkové technické prostředky rozšiřují adresovací rozsah tohoto zápisu na celý adresový prostor řídící paměti. Na základě obsahu instrukčního registru lze pak vyvolat přechod na první adresu mikroprogramu interpretace cílové instrukce, jež může být zapsána v kterémkoliv paměťovém místě řídící paměti.

Obsah pole CS řídí vstupní podmínky při aritmetických operacích s jednoduchou i dvojnásobnou délkou slova, zpracování výstupu přenosu při posuvech a zápis přenosu a pomocného přenosu do regist-

ru stavového slova.

Obsah pole DBY předepisuje buď nezávislé nebo současné zpracování přenosů jednotlivými bloky. Současné zpracování odpovídá činnosti při operacích s dvojnásobnou délkou slova.

Obsah pole IST řídí obvody přerušení. Je-li ve stavu 0, mohou signály INT a HOLD přerušit vykonávání programu. Tento stav se předepisuje vždy ve fázi převzetí cílové instrukce s výjimkou cílové instrukce EI, Di a převzetí přerušovacího vektoru.

Obsah pole RRE předepisuje, zda skupina F pole CPE je určena výlučně obsahem pole RGP nebo též obsahem instrukčního registru, t.j. operandem cílové instrukce.

Stav pole EXT předepisuje testování signálu READY. Tak se emuluje čekání na vybavení informace z operační paměti cílového mikropočítače.

Obsah pole CCR umožňuje zastavit hodiny obou bloků operačních jednotek. Obsah jejich registrů a jejich stav se tak nemění, což umožňuje nedestructivní testování jednotlivých bitů, popřípadě předepsání nevýkonné operace v čekacích smyčkách. Používá se např. při čekání na signál READY, kdy se též aktivuje pole EXT.

Pole NC1 a NC2 mají stejný význam; řízení je však omezeno pouze na jeden z bloků operačních jednotek.

Pole SWPA umožňuje předepsat generaci jednoho ze signálů řídicí sběrnice RTRAP, HLTA, INTA, IOW, IOR, MEMW a MEMR. Pole dekodují výstupní obvody řídicí sběrnice v obr. 12.5. Je-li splněna podmínka EXT . READY, signály se negenerují.

Obsah pole IER řídí přepis obsahu signálu FO do klopného obvodu INTE.

Obsah pole D určuje, který z výstupů sběrnice D bloků operačních jednotek je připojen na společnou sběrnici.

Obsah pole IRW řídí zápis dat z obousměrné datové sběrnice do instrukčního registru ve třech případech: při čtení obsahu operační paměti vyvolané signálem MEMR, při čtení z vnějšího zařízení vyvolaném signálem IOR, nebo při čtení přerušovacího vektoru vyvolaného signálem INTA.

Obsah pole DOE řídí předání dat z výstupu aritmetické sekce

na datovou sběrnici během strojových cyklů zápisu do paměti. (signál MEMW) a zápisu do vnějšího zařízení (signál IOW).

Obsah čtyřbitového pole FGP určuje skupinu F pole CPE. Tato skupina má tři bity (tab. 10.5). Jeden z bitů skupiny, bit F5, je zapsán v poli FGP nezávisle pro oba bloky operačních jednotek. Tak lze v témže mikroinstrukčním cyklu vykonávat nezávisle dvě aritmetické nebo logické operace ze skupiny 0 a II, I a III, IV a VI, V a VII.

12.2.4 Příklad emulace cílové instrukce

Jako příklad uvedeme emulaci cílové instrukce MOV A,H. Tato instrukce cílového mikropočítače přesunuje obsah registru H do registru A, t.j. střadače. Při této instrukci se nemění obsah registrů stavového slova PSW.

Vývojový diagram převzetí a dekódování cílových instrukcí a vykonání instrukce MOV A,M je uveden v obr. 12.6. Poslední mikroinstrukce každého mikroprogramu vykonání cílové instrukce předepíše příští adresu mikroinstrukce 006H. Jak je zřejmé z obrázku mikroprogram však na tuto adresu nepřechází, ale vrací se po vykonání cílové instrukce na adresu 007H, kde začíná mikroprogram převzetí a dekódování cílových instrukcí. Pokud totiž není signál READY v aktivním stavu (signalizujícím, že data na výstupu operační paměti jsou platná) vykonává se namísto mikroinstrukce na adrese 006H instrukce na adrese 007H, neboť signál READY (zpracovaný v obvodech větvení) řídí adresový signál MAO a modifikuje obsah adresového registru řídicí jednotky 3001.

Obsah mikroinstrukčních slov mikroprogramu převzetí a dekódování cílových instrukcí a provedení cílové instrukce MOV A,M je zapsán v tab. 12.9. Předpokládá se, že v neuvedené deklarační části mikroprogramu jsou předepsány obsahy polí explicitně v mikroinstrukci neuvedených tak, aby tato pole předepisovala nevýkonné mikrooperace, a dále, že jsou deklarovány identifikátory mikrooperací, jejichž význam je vysvětlen v dalším textu.

První mikroinstrukce na adrese 007H vykonává převzetí cílové instrukce a vzorkování signálů INT a READY. V poli AC se předepíše mikrooperace JCR = 006H. Pokud však není signál READY ve stavu 0, modifikuje se předepsaná adresa na adresu 007H, a mikroprogram zús-

stává v čekací smyčce tvořené jedinou mikroinstrukcí na adrese 007H. Tato mikroinstrukce předepisuje následující činnosti: Mikro-kód pole IRW předepisuje zápis obsahu obousměrné datové sběrnice do instrukčního registru, EXT předepisuje modifikaci adresového signálu MAO stavem signálu READY a INTE připouští přijetí požadavku na přerušeni.

Mikroinstrukce na adrese 006H zajišťuje dekodování cílové instrukce a aktualizaci obsahu čítače instrukcí cílového mikropočítače PC a adresového registru MAR a adresového registru operační paměti EMAR hostitelského systému. Jejich obsah před převzetím instrukce a po aktualizaci je uveden v tab. 12.10. Cílový čítač instrukcí PC je podle tab. 12.7 reprezentován registrem R4 hostitelského systému. Aktualizace obsahu všech adresových registrů je řízena mikrokódem DBY, převádějícím aritmetickou sekci do režimu zpracování slov s dvojnásobnou délkou; mikrokódem FFL, předepisujícím stav 1 na výstupu FO řídící jednotky 3001, tedy na vstup CI bloku 1 operačních jednotek; mikrokódem R44, předepisujícím vstupní signály dekodéru adres registrů (po dekodování skupiny R sběrnice F (FO:F3) obou bloků operačních jednotek předepisuje kód registru R4). Mikro-kód KL předepisuje samé jedničky na vstupy sběrnice K operačních jednotek a mikrooperace LMI předepisuje mikrooperaci v poli CPE. Podle tab. 10.5 se uskuteční mikrooperace:

$R4 \rightarrow MAR$

$R4 + CI \rightarrow R4$

Dekodování cílové instrukce, uložené předchozí mikroinstrukcí do instrukčního registru je řízeno obsahem polí LD a SJM. Obsah instrukčního registru odpovídající instrukci MOV A,M je 7CH a dekodér cílových instrukcí ji při tomto stavu pole SJM překóduje na vstup instrukční sběrnice řídící jednotky. Do adresového registru se zapíše adresa 03A, na které je zapsána první mikroinstrukce úseku mikroprogramu cílové instrukce MOV A,M.

Tato mikroinstrukce předepisuje přesun obsahu registru R2 do pracovního střadače (střadač AC hostitelského systému) a přechod na následující mikroinstrukci.

Přesun se uskutečňuje následujícími mikrooperacemi, předepsanými mikroinstrukcí. V poli CPE je předepsán mikrokód ILR. Za předpokladu, že v poli sběrnice K jsou samé jedničky (K'1) na vstup CI

bloku operačních jednotek CI se přivede negovaný výstup FO řídící jednotky (předepisovaný identifikátorem mikrooperací FFI a NAN), a na vstup sběrnice F operačního bloku 2 se přivede kód registru R2 (vzniklý dekodováním obsahu instrukčního registru na základě aktivního stavu pole RRE a obsahu pole RGP) můžeme výslednou mikrooperaci, uskutečňovanou v bloku 2 operačních jednotek, popsat ve shodě s tab. 10.5 takto:

$R2 \rightarrow R2, AC.$

Na přesunu, probíhající v bloku 1 nezáleží, jak je zřejmé z následující činnosti. Přechod na následující mikroinstrukci je uskutečněn na základě stejného obsahu instrukčního registru, avšak tentokrát není v mikroinstrukci předepsáno pole SJM, takže dekodér cílových instrukcí generuje adresu 07FH.

Mikroinstrukce na této adrese dokončuje přesun obsahu registru RZ do registru T. Uskutečňuje na základě mikrooperace LDI v poli CPE jeho druhou fázi, t.j. přesun obsahu registru AC do registru T. Za předpokladu, že mikrooperace RBE předepisuje spolu s obsahem pole RGP a obsahem instrukčního registru kód registru T na sběrnici F operačních jednotek a mikrokód v poli CS a poli FO předepisují převést na vstup CI operačních jednotek stav 1, a mikrokód pole KS a pole ED1 předepisuje jako zdroj informace sběrnice K obsah sběrnice D druhého bloku operačních jednotek, pak můžeme výslednou mikrooperaci popsat vztahem:

$(IAK) - 1 + CI \rightarrow T.$

Vzhledem k tomu, že $CI = 1$, $I = R2(2)$, a pole IMB, které není uvedeno, předepisuje samé jedničky na sběrnici I, platí:

$AC(2) \rightarrow T(1)$

$AC(2) \rightarrow T(2)$

Mikroinstrukce tedy předepisuje přepis obsahu střadače bloku 2 operačních jednotek hostitelského mikropočítače do střadače cílového mikropočítače, který je reprezentován registrem T v bloku 1 i v bloku 2.

Mikroinstrukce na adrese 07GH zajišťuje též přechod do úseku převzetí a dekodování cílové instrukce. Vzhledem k tomu, že předepisuje mikrokódy polí EXT, IRW a IST, stejně jako mikroinstrukce s adresou 007H, uskutečňuje již činnosti, spojené s převzetím cílové

instrukce, popřípadě povoluje přijetí požadavku na přerušení.

12.3 Zhodnocení příkladů emulace

V předchozích odstavcích jsme uvedli dva dosti rozdílné příklady emulace. V prvním příkladu je architektura cílového počítače poměrně blízká architektuře hostitelského systému. Hostitelský systém je sice v tomto případě nutné vybavit kromě základních technických prostředků, t.j. řídicí jednotky 3001 a bloku operačních jednotek 3002, ještě dalšími technickými prostředky pro styk s operační pamětí a s výstupními a vstupními zařízeními, ale mnohé základní činnosti, např. dekodování cílové instrukce, adresování registrů cílového počítače, emulace instrukčního registru atd., se uskutečňuje výlučně základními technickými prostředky hostitelského systému.

Druhý příklad je odlišný ve dvou aspektech. Prvním z nich jsou značně rozdílné architektury cílového a hostitelského mikropočítače. Druhým je snaha docílit co nejvyšší účinnost emulace, neboť emulátor má komerční charakter a zvýšené náklady na technické prostředky hostitelského systému se odráží ve vyšších parametrech emulátoru ve srovnání s cílovým mikropočítačem. Oba tyto aspekty vedly k tomu, že základní technické prostředky hostitelského systému byly rozšířeny především o pevné paměti, které jsou použity převážně ve funkci dekodérů. Tyto pevné paměti pak zrychlí emulaci, neboť přejímají funkce vykonávané v předcházejícím příkladu základními prostředky, např. dekodování instrukce, přiřazení registrů cílového a hostitelského systému atd. K optimalizaci styku s operační pamětí a vstupními a výstupními zařízeními je hostitelský systém doplněn adresovým registrem operační paměti EMAR, který spolu s cílovým registrem PC a registrem MAR operační jednotky vytváří adresovací strukturu s třemi úrovněmi. Rovněž architektura cílového počítače, orientovaná na zpracování slov jednoduché i dvonásobné délky byla v hostitelském systému respektována uspořádáním technických prostředků.

Výslednou účinnost emulace charakterizuje tab. 12.11. Bylo by samozřejmě možné technické prostředky hostitelského systému zjednodušit a emulovat cílový mikropočítač převážně mikroprogramovým vybavením. Tato cesta by vedla ke snížení účinnosti emulace, takže porovnání rychlostí cílového mikropočítače a emulátoru by již neby-

lo tak příznivé jako v tabulce. Význam emulace by pak spočíval pouze v možnosti rozšíření operačního kódu cílového mikropočítače, ale nebylo by možné využívat emulátor v časově náročných aplikacích, ve kterých jsou parametry cílového mikropočítače nedostatečné. Návrh emulátoru firmy Signetics však byl podřízen požadavku možnosti jeho aplikace právě v těchto případech.