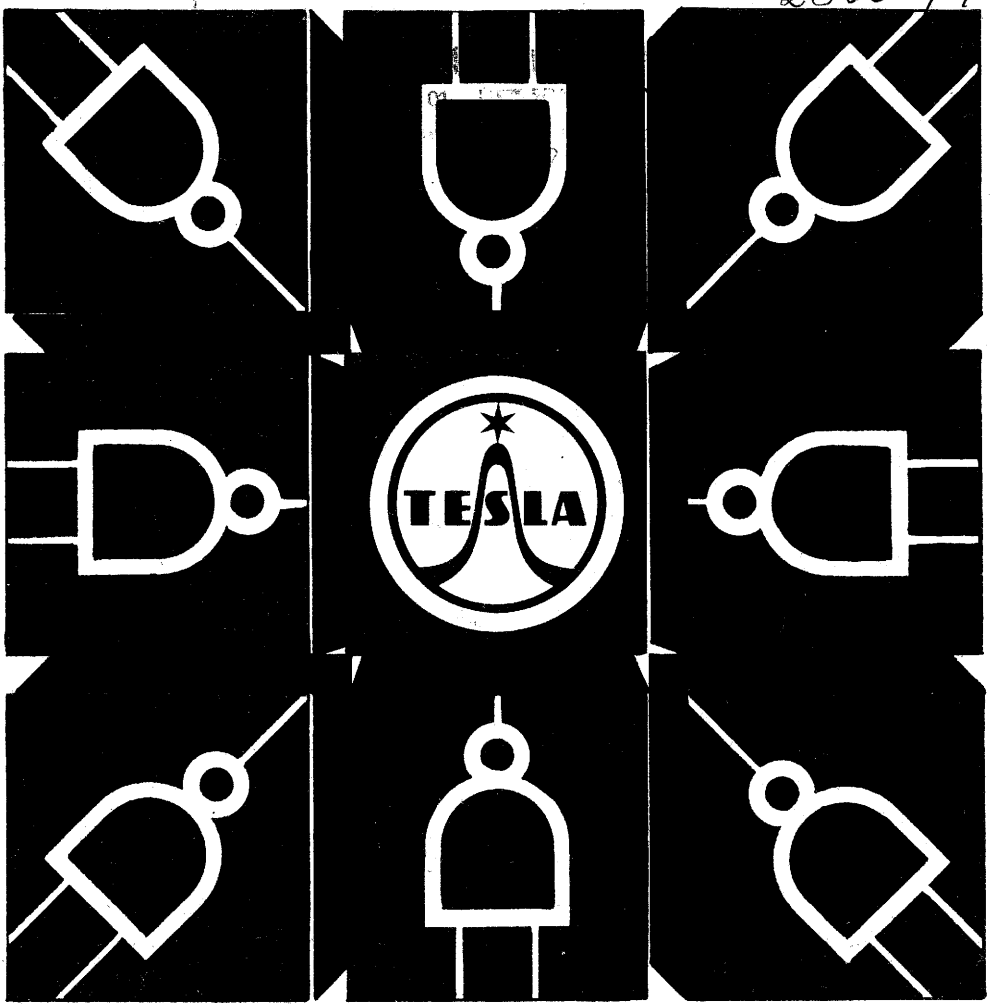
2300 /1



TESLA VÚST ČSVTS TESLA VÚST ČSVTS TESLA VÚST ČSVTS

MIKROPOČÍTAČE

ING. ZDENĚK SOBOTKA CSc.
ING. JAROSLAV STARÝ

TEXTOVÁ ČÁST

2300

Pobočka Československé vědeckotechnické společnosti při
TESLA Výzkumném ústavu pro sdělovací techniku A.S.Popova

TECHNICKÁ ÚSTŘEDNA SPOJŮ PRAHA
závod 01 - ÚSTŘEDNÍ DÍLNY PRAHA
PRAHA 7, Bubenečská třída 15

Ing. Zdeněk SOBOTKA[2] CSc.

Ing. Jaroslav STARÝ

MIKROPOČÍTAČE

Textová část

PRAHA 1981

2300

TECHNICKÁ ÚSTŘEDNA SPOJŮ PRAHA
závod 01 - ÚSTŘEDNÍ DÍLNY PRAHA
PRAHA 7, [illegible] 16
2

OBSAH :

	Předmluva	11
1	Architektura mikropočítačů	13
1.1	Skupinové schéma	13
1.2	Mikroprocesor a jeho architektura	14
1.3	Činnost mikropočítače	18
1.4	Adresování operační paměti	19
1.5	Sběrnice	20
1.6	Systém přerušení	23
1.7	Přenos dat	26
1.8	Přídavná zařízení mikropočítačů	29
2	Programování mikropočítačů	33
2.1	Základní poznatky a úvahy o programování	33
2.2	Sestavy mikropočítačů	34
2.3	Členění programového vybavení	36
2.4	Programovací jazyky	38
2.5	Jazyk symbolických adres	41
2.5.1	Pseudoinstrukce	43
2.5.2	Literál	45
2.5.3	Asembler	46
2.5.4	Vlastní a nevlastní asembler	48
2.5.5	Podprogram a makroinstrukce	48
2.5.6	Makroasembler	50
2.6	Zavedení programů do počítače	51
2.6.1	Postup zavedení programů	51
2.7	Zaváděcí programy	52
2.7.1	Samozaváděcí program	52
2.7.2	Absolutní zaváděcí program	53

2.7.3 Přemísťující zaváděcí program 53
2.8 Vyšší programovací jazyky 57
2.9 Kompilátor 60
2.10 Jazyk FORTRAN 61
2.11 Jazyk PL/M 63
2.12 Interpretační jazyky, BASIC 66
2.13 Účelnost použití vyšších programovacích jazyků 67
2.13.1 Výhody a nevýhody použití vyššího programovacího jazyka 68
2.14 Programy pro úpravy uživatelských programů a jejich ladění 70
2.14.1 Modifikace uživatelských programů 70
2.14.2 Úpravný program 71
2.14.3 Ladicí program 72
2.14.4 Simulační programy 72
2.15 Operační systém 73
2.15.1 Nespřažený operační systém, diskový operační systém a multiprogramování 75
2.15.2 Spřažený operační systém, sdílení času 76
2.15.3 Operační systém pro činnost v reálném čase 77
2.16 Mikroprogramování 80

3 Mikroprocesor 8080 83
3.1 Architektura mikroprocesoru 8080 86
3.2 Příznakové bity 89
3.3 Instrukční, strojový a hodinový cyklus 91
3.4 Identifikace strojového cyklu 92
3.5 Časování a synchronizace, sled stavů 94
3.6 Dekódování stavu, vazba mikroprocesoru na okolí ... 97
3.7 Soubor instrukcí 98
3.8 Adresování dat a instrukcí 101
3.9 Adresování zásobníkové paměti 102
3.10 Adresování vstupů - výstupů 103
3.11 Přerušení 106
3.12 Rodina integrovaných obvodů k mikroprocesoru 8080 . 108
3.13 Návěstí, operační znak, operand, pseudoinstrukce v jazyku symbolických adres 115

2300
TECHNICKÁ ÚSTŘEDNA SPOJŮ PRAHA
závod 01 - ÚSTŘEDNÍ DÍLNY PRAHA
PRAHA 7, Dienzenhoferova ulice 16

4	Návrh mikropočítačového systému ...?	119
4.1	Kritéria systému	119
4.1.1	Vhodnost použití mikroprocesoru	120
4.1.2	Prostředky pro údržbu	120
4.2	Postup při programování mikropočítače	121
4.2.1	Definice systému	121
4.3	Prostředky k sestavení programu	123
4.4	Kódování	126
4.5	Zkoušení a ladění	127
4.6	Technika programování	128
4.6.1	Užívání instrukcí	129
4.6.2	Přesnost výpočtu, měřítko	130
5	Příklady programování	133
5.1	Sčítání a odčítání v dvojkové soustavě	133
5.2	Vytvoření smyčky	133
5.3	Násobení konstantou použitím smyčky	134
5.4	Sečtení a odečtení vícebytových čísel	135
5.5	Sečtení dvou čísel v dvojkově desítkovém kódu	137
5.6	Násobení a dělení pomocí posuvu	139
5.7	Skok v programu v závislosti na shodě porovnávaných slov	140
5.8	Větvení programu pomocí tabulky větvení a nepřímého adresování	141
5.9	Přenos dat do podprogramů	142
5.10	Sečtení libovolného počtu čísel pomocí podprogramu a seznamu parametrů	143
5.11	Vícebitové logické operace	144
5.12	Převod číslic z kódu ASCII do kódu 8 - 4 - 2 - 1 a ukládání dvou číslic v kódu 8 - 4 - 2 1 do jednoho bytu	147
6	Analogově číslicové a číslicově analogové převodníky s vazbou na mikropočítač	149
6.1	Číslicově analogový převod řešený programovým vybavením	150
6.2	Číslicově analogový převod řešený technickým vybavením	150

6.3 Generování složitějších napěťových průběhů pomocí mikropočítače a ČA převodníku 151
6.4 Princip funkce číslicově analogových převodníků ... 152
6.5 Parametry a přesnost číslicově analogových převodníků 154
6.6 Ananlogově číslicové převodníky 156
6.7 Analogově číslicový převod řešený technickým vybavením 157
6.8 Rozdělení analogově číslicových převodníků 158
6.9 Nejčastěji používané principy číslicových převodníků 159
6.10 Řešení analogově číslicového převodu programovým vybavením 163
6.10.1 Nepodmíněně jednoslovný přenos dat z analogově číslicového převodníku 166
6.10.2 Vstup dat z analogově číslicového převodníku do mikropočítače podmíněným jednoslovným přenosem 167
6.11 Automatické nastavení rozsahu 169
6.12 Parametry analogově číslicového převodníku 169
6.13 Kódy používané u číslicově analogových a analogově číslicových převodníků 173

7 Technické prostředky pro návrh a oživování mikropočítačových systémů 179
7.1 Mikropočítačový vývojový systém 179
7.1.1 Emulační a zkušební adaptor 182
7.2 Postup při vývoji programu pomocí vývojového mikropočítačového systému 183
7.3 Ladění programu pomocí mikropočítačového vývojového systému 184
7.4 Simulátor permanentní paměti 184
7.5 Nepřímé programy 185
7.6 Školní mikropočítače 186
7.6.1 Oblasti využívání 187
7.6.2 Vlastnosti školních mikropočítačů 188
7.6.3 Vývoj programu pomocí elementárního mikropočítačového vývojového systému 189
7.7 Logické analyzátory 189
7.7.1 Skupinové schéma a funkce logického analyzátoru ... 190

TECHNICKÁ ÚSTŘEDNA SPOJŮ PRAHA
Závod 01 - ÚSTŘEDNÍ DÍLNY PRAHA
Praha 2, Slezská ulice číslo 15

7.7.2 Záznam sledovaných dat do paměti 192
7.7.3 Paměť 193
7.7.4 Zobrazování 194
7.7.5 Použití logických analyzátorů, výrobci, cena 195
7.7.6 Mikropočítačové analyzátory 195

8 Příklady architektur mikropočítačů 197

8.1 Mikropočítač Intel MCS-48 197
8.1.2 Rozšíření mikropočítače MCS-48 203
8.1.3 Operační kód 207
8.2 Mikropočítač Zilog Z-80 209
8.2.1 Mikroprocesor Zilog Z-80 210
8.2.2 Operační kód 212
8.3 Mikropočítač Intel MCS-86 213
8.3.1 Jednotka styku se sběrnicí 213
8.3.2 Výkonná jednotka 216
8.3.3 Operační kód mikropočítače MCS-86 217
8.3.4 Styk mikropočítače MCS-86 s okolím 223

9 Mikroprogramování 227

9.1 Úvod 227
9.2 Koncepce mikroprogramování 228
9.2.1 Pevně zapojený řadič 229
9.2.2 Mikroprogramovaný řadič 231
9.3 Aplikace mikroprogramování 233
9.3.1 Univerzální počítače 233
9.3.2 Periferní a přídavná zařízení počítačů 236
9.3.3 Speciální počítače 236
9.4 Architektura a organizace počítače 237
9.5 Architektura mikroprogramovatelného počítače 239
9.5.1 Šíře toku dat 239
9.5.2 Aritmetická jednotka 240
9.5.3 Registry procesoru 241
9.5.4 Řídicí paměť 242
9.5.5 Operační paměť 243
9.5.6 Struktura sběrnic 243
9.5.7 Systém přerušení 244
9.5.8 Struktura vstupů a výstupů 244

9.6 Struktura operačních jednotek mikroprogramovatelných mikropočítačů 245
9.6.1 Operační jednotka Fairchild 9405 245
9.6.2 Operační jednotka Intel 3002 246
9.6.3 Operační jednotka Advanced Micro Devices 2901 247
9.7 Struktura mikroinstrukcí 247
9.7.1 Formát mikroinstrukce 247
9.7.2 Stupeň zakódování řídicí informace 248
9.7.3 Horizontální a vertikální mikroinstrukce 249
9.7.4 Generování adresy příští mikroinstrukce 250
9.7.5 Mikroinstrukční cyklus 251
9.8 Struktura řídicích jednotek mikroprogramovatelných mikropočítačů 251
9.8.1 Řídicí jednotka Fairchild 9408 252
9.8.2 Řídicí jednotka Intel 3001 253
9.8.3 Řídicí jednotka Advanced Micro Devices 2909 253
9.9 Mikroprogramovací jazyky 254

10 Mikropočítač Intel 3000 257
10.1 Řídicí jednotka 3001 257
10.2 Operační jednotka 3002 261
10.3 Generátor zrychleného přenosu 3003 263
10.4 Řadič prioritního přerušení 3214 264
10.5 Sestavy mikropočítače Intel 3000 266
10.5.1 Časování mikropočítače 268

11 Mikroprogramovací systém CROMIS 271
11.1 Jazyk symbolických mikrokódů XMAS 273
11.1.1 Základní prvky a objekty jazyka XMAS. Deklarace a příkazy 273
11.1.2 Výstup mikroasembleru XMAS 278
11.2 Zaváděcí program XMAP 280
11.3 Řídicí jazyk systému CROMIS 281
11.4 Mikroprogramovací techniky 283
11.4.1 Deklarace STRING, IMPLY, KBUS 283
11.4.2 Přiřazování absolutních adres symbolickým adresám mikroinstrukcí 284
11.4.3 Pseudopodprogramy 285

TECHNICKÁ KNIHOVNA ČVUT PRAHA
POVÁ 01 - UNIVERZITNÍ CENTRUM PRAHA

11.4.4 Příklad mikroprogramu 286
11.5 Modifikovaná Backus-Naurova notace 286

12 Emulace 289

12.1 Emulace hypotetického cílového počítače 289
12.1.1 Architektura cílového počítače 289
12.1.2 Mikroprogram emulace 290
12.2 Emulace mikropočítače Intel 8080 293
12.2.1 Architektura cílového počítače 293
12.2.2 Uspořádání hostitelského systému 293
12.2.3 Struktura mikroinstrukčního slova 296
12.2.4 Příklad emulace cílové instrukce 298
12.3 Zhodnocení příkladů emulace 301

13 Sběrnice a rozhraní mikropočítačových systémů 303

13.1 Rozhraní mezi bloky systému na rozpracování dat .. 303
13.2 Sběrnice 304
13.2.1 Technické prostředky sběrnic 305
13.2.2 Metody sdílení sběrnice 306
13.2.3 Metody přenosu po sběrnici 307
13.2.4 Sběrnice v mikropočítačových systémech 309

14 Systémové sběrnice mikropočítačů 313

14.1 Sběrnice Multibus 313
14.1.1 Mechanické a elektrické vlastnosti 314
14.1.2 Funkční vlastnosti 314
14.2 Sběrnice Microbus 318
14.3 Sběrnice S-100 319
14.3.1 Mechanické a elektrické vlastnosti 319
14.3.2 Funkční vlastnosti 320

15 Standardní paralelní asynchronní rozhraní 325

15.1 Vlastnosti rozhraní 326
15.2 Stavebnicové prvky pro realizaci rozhraní mezi mikropočítačem a vnějším zařízením 327
15.3 Vazební obvod 8255 jako standardní paralelní asynchronní rozhraní 330

16 Sběrnice a systémy pro místní přenos dat 335

16.1 Sběrnice IMS-2 335
16.1.2 Funkční vlastnosti 335
16.1.3 IMS-2 a mikropočítačové systémy 342
16.2 Systém CAMAC 344

17 Dálkový přenos dat 345

17.1 Telekomunikační prostředky a metody přenosu dat ... 345
17.1.2 Metody přenosu dat po telekomunikačních spojích ... 346
17.1.3 Měniče signálů 350
17.1.4 Rozhraní v systémech dálkových přenosu dat 352
17.2 Organizace přenosu dat 354
17.2.1 Struktura sítí přenosu dat 354
17.2.2 Řídicí postupy 355
17.2.3 Formát zpráv 357
17.2.4 Příklady řídicích postupů 360
17.3 Dálkový přenos dat a mikropočítačové systémy 362
17.3.1 Stavebnicové prvky pro realizaci rozhraní S2 362
17.3.2 Využití stavebnicových prvků při realizaci řídicích postupů 370

18 Multimikroprocesorové a multipočítačové systémy ... 373

18.1 Klasifikace číslicových systémů 374
18.2 Vlastnosti systémů s několika procesory a jejich srovnání 377
18.3 Synchronizace procesů v multiprocesorovém systému . 378
18.4 Příklad multiprocesorového systému 380
18.4.1 Mechanismus virtuálního adresování 382
18.4.2 Uspořádání adresovacího procesoru 384
18.4.3 Zhodnocení aplikačních možností 385

19 Literatura 387

PŘEDMLUVA

Počítač je nejdůmyslnější stroj, který dosud člověk vymyslel. Přínos mikropočítače spočívá především v tom, že tento složitý stroj je relativně laciný. Tím se otevřely nové možnosti v oblasti aplikací.

Zavádění počítačů do oblasti lidské činnosti není proces jednoduchý. Možnosti počítače jsou někdy nedoceňovány, častěji však bývají v současné době přeceňovány. Přecenění schopností počítačů souvisí s řadou vypracovaných studií zabývající se jejich možnostmi. V těchto studiích se ukazuje, že málokterá lidská činnost je tak „tvůrčí", že by v zásadě nemohla být vykonávána počítačem. Tvůrčí činnost je však činnost vytvářející hodnoty, pro kterou není znám algoritmus, je tedy charakterizována právě tím, že ji nelze provádět počítačem. Jsou však známy případy, kdy počítač hraje šachy, komponuje hudbu atd. Tyto tvůrčí činnosti mají svá pravidla, t.j. můžeme pro ně vypracovat algoritmus. Odtud můžeme dedukovat, že různé lidské činnosti mají více či méně složitá pravidla, pro něž, dosud jen teoreticky, můžeme vypracovat algoritmus a použít počítač. Sestavení algoritmu může mít pouze teoretický nebo praktický význam. S tím souvisí teoretická a praktická mez použitelnosti počítače. Sotva bychom našli činnost, která by ležela za teoretickou mezí použitelnosti počítače. Praktická mez závisí na možnosti řešení v rozumném čase a hospodárně; tato mez se neustále posunuje se zdokonalováním stavu techniky a rozšiřováním vědomostí lidstva.

Pro systémové pracovníky, pro které je především určena tato publikace, je klíčovou otázkou kde a jak využít mikropočítače. Již dnes je známo několik desítek tisíc různých aplikací mikropočítače. Řada aplikací je již dne zřejmá (řízení obráběcích strojů, v automobilu, řízení šicího stroje, v registračních pokladnách atd.)

Před několika lety vypsal známý americký odborný časopis Electrical design news (EDN) soutěž na nejlepší aplikaci mikropočítačů. První cenu vyhrálo nasazení mikropočítače pro řízení splachovacích záchodů ve výškové budově, druhé místo řízení krouhače jablek pro domácí výrobu jablečného moštu. Tímto výsledkem se nezesměšňuje význam využití mikropočítačů, naopak, tím se jen zdůrazňuje, že jejich využití je často za hranicí našich momentálních představ. To potvrzuje i článek v časopisu Electronique (1979, čís. 271, str. 54), kde se referuje o protéze lidské nohy řízené mikropočítačem.

Kapitoly 1 až 7 vypracoval ing. Z. Sobotka CSc, kapitoly 8 až 18 ing. J. Starý.

1. ARCHITEKTURA MIKROPOČÍTAČŮ

1.1 Skupinové schéma

Architektura mikropočítačů je značně různorodá. Podrobná skupinová schémata jednotlivých typů se mohou značně lišit. Jelikož však mikropočítač je určitým druhem počítače, je jejich základní skupinové schéma společné. Většina mikropočítačů je Von Neumanova typu, který se vyznačuje společnou operační pamětí pro instrukce i data. Tato operační paměť může být sestavena z permanentní paměti a paměti s libovolným výběrem.

Typický počítač (i mikropočítač) má následující části (obr. 1.1):

- vstup (vstupní jednotka)
- výstup (výstupní jednotka)
- paměť (operační paměť)
- aritmetickou jednotku
- řadič

Paměť slouží pro uchování instrukcí (což jsou zakódované příkazy pro činnost počítače) a dat (což jsou zakódované informace, které mají být počítačem zpracovány). Sled logicky navazujících instrukcí tvoří program. Procesor, který se skládá z aritmetické jednotky a řadiče, čte v určitém sledu jednotlivé instrukce z paměti a odvozuje z nich příslušné zpracování dat. Data uložená v operační paměti jsou procesoru rychle dostupná. Často však tato paměť nemá dostatečnou kapacitu pro uchování všech informací potřebných pro určitou úlohu. Tento problém může být řešen pomocí jednoho nebo více vstupů. Procesor může adresovat zvolený vstup a přijmout data, která se na tomto vstupu vyskytují. Tím je umožněno, aby centrální jednotka počítače přijala informace z přídavných zařízení, např. z paměti magnetickým pružným

diskem.

Počítač mívá jeden nebo více výstupů, které umožňují procesoru předat zpracované výsledky vnějšímu zařízení. Z výstupu lze přenášet informace na zobrazovací jednotku, na tiskárnu, na vnější přídavnou paměť, nebo na ovládací signály pro řízený proces. Podobně jako vstupy jsou adresovány i výstupy. Vstupy i výstupy umožňují styk procesoru s okolím.

Procesor sjednocuje činnost celého systému, tj. řídí funkce všech bloků počítače, musí být schopen vybírat instrukce z paměti, dekódovat je a provádět. Vyžadují-li to instrukce, musí být též schopen adresovat paměť a vstupy-výstupy.

Základní sled operací mikropočítače je následující:

a) vyslání adresy požadované instrukce z čítače instrukcí do operační paměti
b) přečtení a dekódování instrukce
c) provedení instrukce.

Tento sled operací nazýváme instrukční cyklus. Často se v popisu počítače slučují funkce uvedené pod a) a b); v tomto případě mluvíme o výběrové a prováděcí době.

1.2 Mikroprocesor a jeho architektura

Jádrem mikropočítače je mikroprocesor. Jeho architektura v po podstatě určuje charakter mikropočítače. Typický mikroprocesor sestává ze základních bloků, které jsou uvedeny na obr. 1.2. Dále si popíšeme jejich funkci.

Střadač

Střadač je registr, který obvykle obsahuje jeden z operandů, jež mají být zpracovány aritmetickou jednotkou. Např. instrukce může dát příkaz aritmetické jednotce k sečtení obsahu střadače a jiného registru a k uložení součtu do střadače. Střadač můžeme považovat za hlavní pracovní registr, do kterého se obvykle přepisují data z paměti a výsledky operací, které se následně ukládají zpět do paměti, nebo přenášejí do vstupně-výstupní jednotky.

Čítač instrukcí

Instrukce, které tvoří program, jsou uloženy v paměti programu v určitém sledu. Čítač instrukcí čítá jednotlivé instrukce tak,

aby je mohl mikroprocesor postupně vybírat a provádět; přesněji řečeno - je to jeho hlavní funkce. Programy však často obsahují podprogramy, jejichž vyvolání může být vyžádáno po různých instrukcích hlavního programu. Podprogram je sled instrukcí sloužící k vykonání určité úlohy, která se několikrát opakuje během programu. Příkladem může být výpočet sinu, odmocniny, logaritmu. Tyto podprogramy mohou mít i další podprogramy pro součet, součin, podíl atd.

Při řízení procesu mohou být jednotlivé činnosti vyvolány jednotlivými podprogramy. Celková činnost mikropočítače je sjednocena hlavním programem, který vyvolá požadované podprogramy.

Vyskytne-li se v programu instrukce „skok do podprogramu", musí se adresa následující instrukce hlavního programu z čítače uložit, aby se po dokončení podprogramu mohl mikroprocesor vrátit do hlavního programu. Do čítače instrukcí se musí vložit adresa první instrukce podprogramu. Tato operace se u mikroprocesorů obvykle provádí pomocí zásobníkové paměti. Ta pracuje tak, jako by obsahovala řadu nad sebou uspořádaných registrů. Ukládá (vybírá) se vždy do (z) nejvyššího registru a obsahy ostatních registrů se dynamicky posouvají dolů (nahoru). Instrukce „skok do podprogramu" způsobí, že adresa instrukce hlavního programu se posune o jeden krok v zásobníkové paměti směrem dolů a do horního registru se uloží adrese první instrukce podprogramu.

Další instrukce skoku může způsobit, že obě uložené adresy v zásobníkové paměti jsou posunuty dále o jeden krok dolů. To nastane, vyvolá-li se další podprogram. Počet podprogramů, které lze vložit bez ztráty původní adresy návratu, závisí na hloubce zásobníkové paměti, která je u různých typů mikroprocesorů různá.

Na konci podprogramu musí nastat skok do příslušného místa původního programu; toho se dosáhne posunutím obsahu zásobníkové paměti o jeden krok nahoru. Tím se vloží do čítače instrukcí příslušná adresa instrukce v původním programu.

Hloubku obsazení zásobníkové paměti indikuje ukazatel zásobníkové paměti: udává adresu další neobsazené adresy v zásobníkové paměti.

Registr instrukcí

Přijme-li mikroprocesor instrukci z paměti, uloží ji do registru instrukcí, kde je ponechána pro dekódování. Délka instrukce (počet bitů, z kterých instrukce sestává) závisí na typu mikroprocesoru. Jednoduchý mikroprocesor by mohl mít např. osmibitový instrukční kód, tj. délku instrukce osm bitů. To by umožňovalo vytvořit 256 = 2^8 odlišných instrukcí, z nichž každá by sestávala z odlišného sledu jedniček a nul. V praxi je tento počet obvykle víc než dostačující. Přesto mnohé typy mikroprocesorů mají proměnnou délku instrukce rovnající se jednomu až třem slovům (např. 1 x 8, 2 x 8, 3 x 8 bitů). Pomocí těchto instrukcí se specifikuje nejen požadovaná operace, ale též jedna nebo více adres pro výběr dat a zápis výsledků.

Dekodér instrukcí

Funkcí dekodéru je instrukce dokódovat a tím určovat operace, které má mikroprocesor provádět. Řídicí signály z výstupu dekódéru instrukcí řídí přesuny informací mezi jednotlivými bloky a určují funkci těchto bloků.

Zápisníková paměť

Tento blok se používá jako dočasná paměť různých informací, neboť jsou snadněji a zpravidla rychleji dosažitelné než informace obsažené v operační paměti. U jednoduchých strojů se velmi často používá též pro záznam adres paměti nebo adres vstupů a výstupů. Tyto adresy lze jednoduchými instrukcemi zvyšovat, takže je možné adresovat následná místa v paměti.

Aritmetická jednotka

Aritmetická jednotka je část mikroprocesoru, která provádí alespoň následující operace:

- sčítání s přenosem
- odečítání s přenosem
- posun doleva a doprava
- čítání nahoru a dolů
- logický součin a logický součet (operace AND a OR)
- porovnání číselně vyjádřených slov pro podmíněné větvení. Toto porovnání může být jen jednoduché s výstupem slova stejná - rozdílná, nebo s porovnáním všech řádů slov.

Složitější aritmetické jednotky mohou vykonávat pomocí tech-

nického vybavení další operace, např. násobení a dělení, a více logických operací.

Časovací a řídicí obvody

Časovací a řídicí obvody s dekodérem instrukcí zahrnujeme pod pojem řadič. Generátor hodinových impulsů je zdrojem taktovacích impulsů pro celý mikropočítač.

Po vybrání a dekódování instrukce časové a řídící obvody vytvoří signály, které započnou požadovanou činnost jednotlivých bloků uvnitř i vně mikroprocesoru. Sled činnosti jednotlivých bloků mikropočítače je neměnný a je určen sledem instrukcí v programu.

V některých případech však časovací a řídicí obvody reagují na vnější signály, např. na signál požadavku přerušení. V tomto případě ustane postupný výběr instrukcí hlavního programu a nastane skok do podprogramu, jenž určí zdroj přerušení, „obslouží" jej a provede skok zpět do hlavního programu.

Dále si uvedeme bloky či obvody, které sice nejsou na obr. 1.2 znázorněny, avšak často je mikroprocesor obsahuje.

Indexový registr

Některé paměti mikroprocesorů jsou adresovány pomocí indexového čísla, které je obsaženo v indexovém registru. Adresa další instrukce může být vyhledána sečtením obsahů čítače instrukcí a indexového registru. Zvětšení obsahu o jedničku způsobí, že mikroprocesor čte na následující stránce paměti.

Klopné obvody příznaků (flags)

Jsou to obvykle klopné obvody pro uchování jednotlivých bitů, které indikují některé stavy mikroprocesoru. Např. klopný obvod příznaku přenosu se nastaví do stavu 1, jestliže aritmetická operace vyvolá přenos. V opačném případě je ve stavu 0. Pomocí stavu klopných obvodů příznaků se vytváří podmíněné skoky v programu.

Jiné klopné obvody příznaků se někdy používají k zamezení či umožnění vstupu signálu přerušení.

Kanál blokových přenosů

Je to obvod, který umožňuje přenést blok dat mezi pamětí mikropočítače a přídavným zařízením jedním nebo druhým směrem, aniž by tento přenos byl ovlivňován mikroprocesorem. Některé mikropro-

cesory mohou během tohoto přenosu pokračovat ve výpočtu.

1.3 Činnost mikropočítače

Především je třeba zdůraznit, že žádný počítač nemůže nic vykonávat dokud v paměti počítače není vložen nějaký program a řízení počítače není nastaveno na první instrukci tohoto programu, která však nemusí být uložena v první buňce paměti. Obsah čítače instrukcí, který je součástí řadiče, lze nastavit na adresu paměťové buňky, s první instrukcí programu ručně. Během další činnosti počítač postupně střídá výběrovou a prováděcí dobu.

Během výběrové doby se čte instrukce, jejíž adresu v paměti určuje čítač instrukcí. Tato instrukce se dekóduje, potom čítač instrukcí zvýší svůj obsah o jedničku, čímž se připraví čtení další instrukce. Je-li instrukce rozložena do dvou či tří slov, je zpracována jako jeden celek.

Během prováděcí doby se uskutečňuje operace určená operačním znakem, který je obsažen v instrukci; např. sečti dvě čísla, ulož slovo na určitou adresu v paměti, naplň čítač instrukcí novým obsahem pro provedení skoku ve sledu instrukcí,atd. Po vykonání této operace následuje další výběrová doba. Předpokládejme, že v paměti je uložena část programu, uvedená na obr. 1.3. Je-li obsah čítače instrukcí 10020, přenese se z paměti slovo (202) do dekodéru instrukcí, a ten dekóduje instrukci jako ADD D (přičti obsah registru D k obsahu střadače). Po přičtení jedničky k obsahu čítače instrukcí je jeho obsah 10021 a nastává výběr další instrukce. Slovo 323 znamená výstup. Vzhledem k nezbytnosti specifikace osmibitové adresy přídavného zařízení, které má přijmout obsah střadače, je instrukce pro výstup obsažena ve dvou slovech. V následující prováděcí době je adresa přídavného zařízení 12 na adresovací sběrnici a informace obsažená ve střadači na datové sběrnici. Přídavné zařízení 12 rozpozná svou vlastní adresu a přijme do svého registru informaci ze střadače prostřednictvím datové sběrnice.

Po naplnění čítače instrukcí adresou 10023 se čte slovo 052, které je počáteční částí tříslovné instrukce. Následující dvě slova této instrukce, která se čtou v důsledku dalších dvou přičtení jedničky k obsahu čítače instrukcí, se uloží do dvou osmibitových registrů H a L. V následující prováděcí době se obsah registru po-

užije pro adresování buněk paměti 26155.

1.4. Adresování operační paměti

Způsoby adresování dat a segmentů programu v paměti se mohou u různých mikropočítačů značně lišit. Problém spočívá v umožnění adresace libovolného místa v paměti, aniž by bylo třeba použít teoreticky potřebný počet bitů v adresní části instrukce.

Úplné adresování každou instrukcí by znamenalo dlouhé instrukce,resp. prodloužení programu víceslovnými instrukcemi.

Během výběrové doby čítač instrukcí obvykle přímo adresuje paměť. Je-li tento čítač 16 bitový, lze jím adresovat až 64 K paměťových míst. Má-li být během prováděcí doby vybrán z paměti nebo do ní uložen operand, je jeho adresa (případně nezbytné informace pro získání této adresy) obsažena v instrukci. Pro adresaci v mikropočítači se používají následující základní způsoby a jejich kombinace:

Při absolutním adresování (obr. 1.4) adresní část instrukce určuje úplnou adresu operandu paměti. Toto adresování je jednoduché, avšak nákladné vzhledem k nezbytnosti dostatečně dlouhého instrukčního slova. Někdy toto adresování nazýváme přímé.

Relativní adresování (obr. 1.5) umožňuje zkrácení adresové části instrukce tím, že tato adresová část instrukce vyjadřuje relativní adresu vzhledem k určitému registru procesoru. Tímto registrem bývá nejčastěji čítač instrukcí. Adresová část instrukce, která se při tomto adresování nazývá posunutí, se přičítá k obsahu čítače instrukcí, čímž se získává efektivní adresa operandu (obr. 1.6). Jinými slovy: adresní část instrukce odpovídá číselnému rozdílu mezi adresou operandu a bázovou adresou určenou obsahem čítače instrukcí. Při použití např. 8 bitového posunutí lze obsáhnout 256 sousedních adres paměti.

Indexové adresování je podobné absolutnímu adresování, avšak v adresové části instrukce je obsažena jen část adresy. Další bity nezbytné pro adresaci jsou obsaženy v jednom nebo více registrech, které jsou součástí aritmetické jednotky. Tyto registry nazýváme indexovými registry. Obvykle obsahuje indexní část instrukce nejvyšší dvojkové řády úplné adresy. Indexové adresování je též

někdy nazýváno zkrácené adresování. Poznamenejme, že někdy je jako indexové adresování označováno relativní adresování, při němž je bázovým registrem indexový registr místo čítače instrukcí.

Registrové adresování je indexové adresování bez možnosti přičtení adresové části instrukce, t.j. adresy v paměti jsou určovány obsahy některých registrů aritmetické jednotky. Jelikož počet těchto registrů bývá relativně malý, je pro jejich výběr potřeba malý počet bitů v instrukci.

Nepřímé adresování (obr. 1.7) začíná jako přímé adresování, avšak slovo získané z paměti nevyjadřuje operand, ale jen adresu požadovaného operandu.

Registrově nepřímé adresování je kombinace registrového a relativního adresování. Instrukce specifikuje adresu registru a posunutí. Adresa paměti je určena obsahem zvoleného registru a posunutím.

Post - indexové adresování (obr.1.8) je jinou variantou kombinace registrovaného a nepřímého adresování. Po každé operaci se obsah indexového registru zvětšuje o jedničku.

Stránkování je určitý druh relativního adresování. Paměť je z hlediska adresování rozdělena do několika částí, které nazýváme stránky. Jedna stránka např. obsahuje 256 paměťových buněk. Požadovaná stránka může být označena na počátku programu. Adresní část instrukce určuje paměťovou buňku ve stránce.

Okamžité adresování není v podstatě adresovací způsob, neboť v pozici adresové části instrukce se nachází přímo operand. U některých mikroprocesorů určité instrukce např. JUMP nebo CALL téměř vždy používají okamžité adresování.

1.5 Sběrnice

Přenos informací mezi mikroprocesorem a ostatními částmi počítače, i mezi jednotlivými bloky uvnitř mikroprocesoru se zpravidla, podobně jako u většiny počítačů, uskutečňuje po jedné nebo po několika sběrnicích. Struktura přenosu dat a instrukcí mezi mikroprocesorem a „okolím" má však často jisté zvláštnosti, ve kterých se odráží poměrně malý počet vývodů pouzdra mikroprocesoru.

Sběrnice sestává z určitého počtu souběžně vedených vodičů propojujících zpravidla několik registrů. Jednotlivé vodiče sběrnice přenášejí jednotlivé bity slova vyjadřujícího přenášenou informaci. Přenos po sběrnici jen mezi dvěma zvolenými registry se zajistí vhodným obvodovým řešením, např. pomocí kombinačních logických členů nebo třístavových logických členů[1] (třístavové logiky).

Sběrnicí může být současně přenášena jen jedna informace mezi dvěma registry z množiny registrů připojených na tuto sběrnici. Použití jedné sběrnice může omezovat rychlost mikropočítače, neboť je někdy žádoucí přenést současně více informací. Proto se v mikropočítačích používá více sběrnic.

Typické uspořádání vnějších sběrnic mikroprocesoru je na obr. 1.9. Obousměrná n bitová sběrnice dat se používá pro přenos dat mezi mikroprocesorem a operační pamětí i přídavnými zařízeními. Mikroprocesor, který přenáší po této sběrnici paralelně n bitů, označujeme zpravidla jako n bitový. Pomocí jednosměrné m bitové adresovací sběrnice se adresuje operační paměť i přídavná zařízení. U některých mikroprocesorů nemá adresovací sběrnice samostatné vývody z pouzdra, ale používá je multiplexním způsobem společně s datovou sběrnicí. Pokud je počet adresovacích bitů větší než je šíře datové sběrnice (počet paralelně přenášených bitů v datové sběrnici), přenášejí se adresovací bity ve více krocích. Společné vývody pro datovou a adresovací sběrnici vyžadují obvody pro rozdělení

---

1) Třístavový logický člen je představován logickým členem se dvěma vstupy, z nichž jeden slouží pro vstup informace, druhý k ovládání logického členu. Je-li ovládací vstup ve stavu 1, logický člen přenáší informaci z informačního vstupu na jeho výstup, přičemž výstupní impedance logického členu je nízká. Je-li však ovládací vstup ve stavu 0, logický člen je uzavřen a jeho výstupní impedance je vysoká. Při řešení sběrnice pomocí třístavových logických členů je třeba zajistit, aby se uskutečnil výstup jen z jednoho členu, v opačném případě může nastat zničení výstupních obvodů třístavového logického členu značnými proudy v okamžicích odlišné logické úrovně na jejich výstupech.

obou sběrnic vně mikroprocesoru (demultiplexní obvody).

Řídicí sběrnice mikroprocesoru obsahuje vodiče, které indikují resp. řídí některé stavy. Mnohé mikroprocesory nemají samostatné vývody z pouzdra pro jednotlivé řídicí signály; tyto signály se přenášejí multiplexně menším počtem vývodů z pouzdra a někdy se přenášejí multiplexně pomocí vývodů datové sběrnice. Typické signály, často obsažené (přímo nebo různým způsobem zakódované) v řídicí sběrnici jsou uvedeny v tab. 1.1.

U mikropočítačů podobně jako u minipočítačů se obvykle používají dva základní způsoby organizace sběrnic, které propojují mikroprocesor, operační paměť (paměť s libovolným výběrem i permanentní paměť) a přídavná zařízení. Jedno řešení používá jeden soubor sběrnic (adresovou, řídicí a datovou) pro propojení mikroprocesoru jak s operační pamětí, tak s přídavnými zařízeními. Toto řešení nazýváme jednosběrnicové uspořádání(single bus system, omnibus memory - mapped I/O, virtual memory I/O).

Druhé řešení používá dvou souborů sběrnic a nazýváme jej dvousběrnicové uspořádání[1)](two bus system, I/O bus, isolated I/O). Jeden soubor sběrnic se používá pro komunikaci mikroprocesoru s operační pamětí, druhý soubor sběrnic pro komunikaci mikroprocesoru s přídavnými zařízeními. Rozdělení sběrnic bylo inspirováno skutečností, že mikroprocesor obvykle komunikuje podstatně rychleji s operační pamětí než s přídavnými zařízeními.

Obě řešení organizace sběrnic jsou převzata z minipočítačů. Jednosběrnicové uspořádání je použito např. u minipočítačů PDP 8 (Digital Equipment Corporation, DEC), dvousběrnicové uspořádání je použito u minipočítačů PDP 11 (DEC) a HP 2116 (Hewlett Packard).

Jednosběrnicové uspořádání je z hlediska vstupů a výstupů koncepčně jednoduché. Není třeba zvláštních vstupních a výstupních instrukcí ani speciálních vstupních a výstupních řídicích signálů v souboru sběrnic. Při tomto uspořádání adresa vyrovnávacího registru každého přídavného zařízení zaujímá adresu paměťového místa. Obvykle je určitá část z celkového počtu adres rezervována pro vstupy a výstupy, t.j. pro přídavná zařízení. Teoreticky však mohou být pro vstupy a výstupy použity libovolné adresy, pokud těmto

1) Sběrnice nemusí být fyzicky rozdělené

adresám neodpovídají paměťové buňky v operační paměti. Často je u mikropočítače s možností adresace 32 K slov posledních 4 K slov, t.j. 28672 až 32767, vyhrazeno pro přídavná zařízení (předpokládá se, že paměť je řešena po blocích 4 K slov), ostatních 28 K slov je vyhrazeno pro adresaci operační paměti.

Dvousběrnicové uspořádání umožňuje obvykle připojit podstatně méně registrů přídavných zařízení, případně registrů vazebních obvodů přídavných zařízení. Např. osmibitový mikroprocesor s šestnáctibitovým adresováním paměti obvykle používá jen 8 bitů pro adresování přídavných zařízení. Při dvousběrnicovém uspořádání jsou přídavná zařízení přístupná pomocí speciálních instrukcí vstup - výstup.

Porovnání vlastností jednosběrnicového a dvousběrnicového uspořádání je v tab. 1.2.

1.6 Systém přerušení

Systém přerušení umožňuje probíhající program určitým podnětem přerušit a vyvolat důležitější program s vyšší prioritou. Po ukončení programu s vyšší prioritou se počítač samočinně vrací k přerušenému programu. U některých počítačů nemůže být program přerušen v libovolném bodě, neboť přerušení by mohlo mít za následek logickou chybu ve výpočtech. Instrukce programu při jejichž vykonávání nelze program přerušit, bývají vhodně označené (pasivace systému přerušení).

Časová odezva centrální jednotky počítače na signál přerušení určuje do značné míry maximální počet přímo spřažených (on - line) přídavných zařízení. Aby byl počítač účinný, musí jeho systém reagovat dostatečně rychle na podněty kritické na čas, i účinně řešit ostatní úlohy. Je vhodné, aby měl počítač určitou rezervu jak v počtu úrovní přerušení i v počítačovém čase, aby případným rozšířením systému nenastalo jeho nasycení.

Základní pojmy a kritéria systému přerušení jsou následující:

- Časová odezva přerušení - vyjadřuje čas mezi signálem vnějšího přerušení (popudem k přerušení) a počátkem provedení první užitečné instrukce přerušením vyžádaného programu. Tento čas nemůže být libovolně krátký - počítač nemůže reagovat rychleji než je doba pro provedení instrukce.

- Struktura přerušení (někdy též prioritní struktura) je systém, který umožňuje počítači reagovat se správnou prioritou na signály přerušení. Je-li vyžádáno nejdůležitější přerušení a počítač nevyvolá v daném čase odpovídající program, není struktura optimální.
- Nasycení systému je stav, ve kterém počítač nestačí dostatečně rychle reagovat na vnější požadavky přerušení a tím jsou některé požadavky přerušení ztraceny. Systém je poddimenzován.

Většina současných mikropočítačů má více než jednu úroveň přerušení. Víceúrovňové přerušení znamená, že program vyvolaný přerušením může být přerušen signálem přerušení s vyšší prioritou. U mikropočítačů se často provádí přerušení následujícím sledem operací: Vzniklý požadavek přerušení na některém vstupu je oznámen mikropočítači. Mikropočítač postupně prohledává jednotlivé vstupy až najde ten, který požadavek vyslal (kruhové vyhledávání). Postupné prohledávání jednotlivých vstupů přerušení však vyžaduje relativně dlouhý čas. Vektorovým přerušením se tento čas zkracuje.

Vektorové přerušení vyvolá podprogram příslušející přídavnému zařízení, které vyvolalo přerušení. „Vektorová adresa" může být v mikropočítači pevně zapojena nebo může být poskytnuta přídavným zařízením.

Příklad uspořádání vektorového přerušení je na obr. 1.10. Přídavné zařízení nastavuje klopný obvod požadavku přerušení a nuluje jej mikroprocesor signálem příjmu přerušení. Klopný obvod zamezení přerušení nastavuje i nuluje program.

Systém přerušení, který je realizován na společném čipu s mikroprocesorem, je obvykle velmi jednoduchý. Složitější systémy přerušení se obvykle řeší pomocí vnějších přídavných obvodů, někdy jedním speciálním integrovaným obvodem s velkou integrací.

Typický systém přerušení, který je na společném čipu s mikroprocesorem, obsahuje: vstup „požadavek přerušení", výstup „potvrzení příjmu požadavku přerušení" a vnitřní programem řízený signál, umožňující aktivaci a pasivaci signálu přerušení. Při „aktivaci" vstupu přerušení (signál přerušení je přípustný) a výskytu signálu přerušení mikroprocesor po ukončení prováděné instrukce vstoupí do procesu přerušení. Způsobů, kterými bývá obsloužen signál přeruše-

ní, je celá řada. Nejčastěji používané způsoby u mikropočítačů jsou tyto:

1./ Provede se skok na pevnou adresu v paměti, kde je „obslužný podprogram.
2./ Provede se skok na podprogram, jehož adresa je obsažena na určité adrese paměti.
3./ Provede se skok na podprogram, jehož adresa je poskytnuta zařízením, které vyvolalo přerušení.
4./ Provede se skok na podprogram, jehož adresa je v tabulce zapsané v operační paměti, a index tabulky je poskytován zařízením, které vyvolalo přerušení.
5./ Provede se instrukce (obvykle skok na podprogram) poskytovaná zařízením, které vyvolalo přerušení.

Ve všech uvedených případech může být „skok na podprogram" standardní, u něhož se pro návrat do původního programu ukládá jen obsah čítače instrukcí, nebo speciální „skok na podprogram přerušení", při němž se ukládají kromě obsahu čítače instrukcí i obsahy dalších registrů mikroprocesoru (např. obsahy zápisníkové paměti, registrů indikujících stavy mikroprocesoru atd.).

U způsobů 3, 4 a 5 musí přídavné zařízení, které vyvolává přerušení, poskytnout pomocí vnějších obvodů mikroprocesoru informaci, kterou nazýváme vektor přerušení. Je-li jen jedno přídavné zařízení, které může vyvolat přerušení, jsou vnější obvody i programové řešení přerušení poměrně jednoduché. Je-li jich však více, vnější obvody i programové řešení se obvykle komplikují. V tomto případě jsou výstupy „požadavek přerušení" přídavných zařízení vedeny přes součtový logický člen na vstup „požadavek přerušení" mikroprocesoru.

Poskytne-li každé zařízení odlišný vektor (odlišnou informaci) během procesu přerušení, nazýváme toto přerušení vektorovým (způsob 3, 4 a 5). Dojde-li k požadavku na vektorové přerušení z více zařízení a uskuteční-li se volba zařízení podle priority, mluvíme o vektorovém prioritním přerušení. Jestliže všechna zařízení poskytují stejný vektor přerušení (stejnou informaci), nazýváme toto přerušení nevektorové (způsob 1 a 2).

Při použití vektorového přerušení přísluší každému zařízení, které je může vyvolat, jeho vlastní program přerušení. Při nevek-

torovém přerušení je program jen jeden a jeho úkolem je vyhledat zařízení, které přerušení vyvolalo. V tomto případě např. vodič „požadavku přerušení" od každého zařízení odpovídá jednomu bitu v registru přerušení. Podprogram přerušení vyhledá zařízení, které vyvolalo přerušení tím, že prohledá registr přerušení. K tomu účelu je však třeba přivést do mikroprocesoru (nebo do obvodu přerušení) vodiče, pomocí nichž je určeno příslušné zařízení. Podle charakteru programu, který vyhledává příslušné zařízení, a podle obvodového řešení může být nevektorové přerušení jednoúrovňové (neprioritní) nebo víceúrovňová (prioritní).

1.7 Přenos dat

Přenos dat mezi centrální jednotkou mikropočítače a přídavnými zařízeními se může uskutečňovat dvěma způsoby:

a) Pomocí programu; nazýváme jej jednoslovný přenos nebo programovaný přenos vstup - výstup, někdy též přenos řízený programem. Kanál, po kterém se přenos uskutečňuje, nazýváme kanál jednoslovného přenosu. Přenos se provádí přes střadač mikroprocesoru.

b) Pomocí zvláštních obvodů mikroprocesoru, nebo řadiče přídavného zařízení; není ovlivňován programem a uskutečňuje se speciálním kanálem pro data, který nazýváme kanál blokových přenosů nebo též kanál přímého vstupu do paměti (DMA, direct memory access). Proces se neuskutečňuje přes střadač procesoru, proto se během tohoto přenosu jeho obsah nemusí „uklízet".

Jednoslovný přenos dat mezi pamětí, mikroprocesorem a přídavným zařízením je znázorněn na obr. 1.11. Pro usnadnění přenosu je mezi mikroprocesor a přídavné zařízení často zařazen vstupně-výstupní vazební obvod (vstupně-výstupní stykové zařízení). Je-li požadován přenos z přídavného zařízení do paměti, je v tomto případě prvním spojovacím článkem v přenosovém řetězci vstupně-výstupní vazební obvod. Po přenesení dat z přídavného zařízení do vazebního obvodu tato data čte mikroprocesor a ukončí přenos uložením dat na požadovanou adresu do paměti.

V případě přenosu dat z paměti do přídavného zařízení probíhá sled přenosu opačným směrem. Mikroprocesor vybere data v paměti a přechodně je uloží do vstupně-výstupního vazebního obvodu, odkud jsou přeneseny do přídavného zařízení. Přenos dat z jednoho zaří-

zení do druhého je řízen programem, uskutečňuje se po jednotlivých slovech, proto jednoslovný přenos.

Jednoslovný přenos lze provádět třemi různými způsoby: nepodmíněným přenosem, podmíněným přenosem a přerušením programu.

Nepodmíněný přenos (synchronní přenos) se nepoužívá často, neboť přídavné zařízení musí být vždy připraveno na přenos. Programově je však velmi jednoduchý, neboť instrukce vstup-výstup se řadí do programu v místě, kde je třeba (obr. 1.12).

Častěji se používá podmíněný přenos (asynchronní přenos), obr. 1.13. Je sice řízen programem, ale uskutečňuje se jen v případě připravenosti přídavného zařízení pro přenos. Podmíněný přenos zajišťuje synchronizaci mezi činností mikroprocesoru a přídavným zařízením; jeho nevýhodou je případný ztrátový čas během čekacího cyklu.

Přenos přerušením programu vylučuje ztrátový čas způsobený čekací smyčkou, neboť periodicky nezkoumá připravenost přídavného zařízení. Počítač provádí základní program do doby připravenosti přídavného zařízení, které přeruší probíhající základní program (obr. 1.14).

Iniciátorem přerušení je přídavné zařízení. Počítač přijme požadavek přerušení, potvrdí jeho příjem, dokončí prováděnou instrukci, ošetří nezbytné informace (uloží obsahy registrů, které se zúčastní podprogramu přerušení), provede podprogram spojený s přerušením a po dokončení této úlohy provede návrat do původního programu (včetně obnovení původních obsahů registrů).

Kanál blokových přenosů umožňuje rychlý přenos informací mezi přídavnými zařízeními a operační pamětí počítače. Přenos není řízen programem. Logické obvody počítače, které tvoří kanál blokového přenosu, jsou nezávislé na obvodech uskutečňujících jednoslovný přenos. Mikroprocesor se přenosu kanálem blokových přenosů nezúčastní, je odpojen od sběrnice. Vazební obvod blokových přenosů je přímo napojen na paměť. To vyžaduje, aby vazební obvod blokového přenosu vyřadil z činnosti mikroprocesor (což obvykle provádí pomocí stavu vysoké impedance na obvodech mikroprocesoru navazujících na sběrnici), adresoval paměť a generoval řídicí signály pro požadovaný přenos dat.

Hlavním rysem kanálu blokových přenosů je skutečnost, že se přenos informací neuskutečňuje prostřednictvím střadače mikroprocesoru (který hraje při jednoslovném přenosu nezbytný článek), ale přímo přes informační registr paměti. Přenos po kanálu blokových přenosů jedním nebo druhým směrem je obvykle vyvolán přídavným zařízením, často prostřednictvím vazebního obvodu blokového přenosu. Má-li přídavné zařízení připravena data pro přenos do paměti, vyšle požadavek k jeho uskutečnění. Podobně vyžaduje-li přídavné zařízení informace z paměti kanálu blokových přenosů, vyšle informaci o připravenosti.

Kanál blokových přenosů vyžaduje k činnosti tyto informace:

1./ Požadavek přídavného zařízení na přenos
2./ Potvrzení požadavku mikroprocesorem
3./ Určení směru přenosu
4./ Určení počáteční a konečné adresy paměti
5./ Přenos dat

Kanálem blokových přenosů se přenáší blok dat, který sestává z více slov. Velikost přenášeného bloku je obvykle určena jeho počátkem a koncem, které jsou specifikovány adresami v paměti. Někdy je velikost bloku určena počtem slov. Je samozřejmé, že adresy jednotlivých přenášených slov v paměti během jednoho přenosu na sebe bezprostředně navazují.

Při jednoduchém řešení kanálu blokových přenosů bývá během přenosu tímto kanálem mikroprocesor vypojen z činnosti. Při jiném, složitějším řešení se přenos uskutečňuje „kradením" cyklů probíhajícího programu, t.j. mikropočítač pokračuje zpomalenou rychlostí v běžné činnosti, obr. 1.15.

Jednoslovný a blokový přenos se liší především rychlostí a složitostí obvodového řešení. Kanál blokových přenosů má složitější obvodové řešení, avšak přenos je podstatně rychlejší. Rychlost přenosu dat je obvykle limitována cyklem paměti. Tento přenos se stává velmi cenným v případě, kdy rychlost mikroprocesoru je nedostatečná pro zajištění požadované rychlosti jednoslovným přenosem.

Jednodušší mikropočítače mají jen kanál jednoslovných přenosů, většina má však oba kanály. Na čipu mikroprocesoru jsou zpravidla

umístěny jen ty nejnutnější obvody pro kanál blokových přenosů. Po vytvoření úplného kanálu jsou třeba obvykle další vnější obvody (často nazývané vazební obvody kanálu blokových přenosů). Tyto obvody jsou někdy řešeny pomocí jednoho speciálního obvodu s velkou integrací.

Pro připojení kanálu blokových přenosů má mikroprocesor obvykle vstup „požadavek blokového přenosu" (kterým se vyjadřuje požadavek, aby mikroprocesor nevyužíval sběrnic) a výstup „potvrzení požadavku blokového přenosu" (kterým mikroprocesor sděluje, že je odpojen od sběrnic). Při blokovém přenosu je třeba budit adresovací, řídicí a datovou sběrnici.

1.8 Přídavná zařízení mikropočítačů

Přídavná zařízení (které někdy nazýváme periferní zařízení) umožňují mikropočítačovému systému komunikaci s okolím. Často jsou používány stejné typy přídavných zařízení jako u větších počítačů, avšak s podstatně nižšími technickými parametry. Volba přídavných zařízení je důležitá, neboť v mnohých případech cena těchto zařízení je srovnatelná nebo překračuje cenu mikropočítače.

Mikropočítač bývá využíván různými způsoby:

a) pracuje autonomně v úplném informačním systému
b) tvoří inteligentní terminál
c) je součástí většího elektronického celku (měřicího přístroje, řadiče, diskové paměti atd.).

V informačních systémech, mezi něž můžeme zařadit řízení procesů, ať již laboratorních nebo průmyslových, je třeba pomocí přídavných zařízení zajistit komunikaci mikropočítače s okolím. Tato přídavná zařízení můžeme rozdělit do tří skupin:

1) Velkokapacitní přídavné paměti (nejčastěji magnetická paměť s pružným diskem nebo kazetová paměť).
2) Přídavná zařízení pro komunikaci člověk-stroj (např. dálnopis, zobrazovací jednotka).
3) Přídavná zařízení pro komunikaci stroj-proces, někdy nazývaná přídavná zařízení procesu (např. analogově číslicový a číslicově analogový převodník).

Přídavná zařízení jsou připojena k mikropočítači pomocí vazeb-

ných obvodů, která provádějí nezbytná přizpůsobení (tvar informace, synchronizace atd.).

Přídavná zařízení pro komunikaci stroj-proces, mohou být rozdělena podle vstupního, resp. výstupního signálu na analogová a číslicová.

Analogové přídavné zařízení

Analogové vstupní přídavné zařízení procesu (analogový vstupní podsystém), jehož součástí je analogově číslicový převodník (AČ převodník, AD převodník), přijímá analogové signály z různých snímačů, které jsou navázány v různých bodech na řízený proces a převádí tyto signály na číslicové. Analogová výstupní přídavná zařízení procesu (analogový výstupní podsystém), jehož součástí je číslicově analogový převodník (ČA, DA převodník), převádí číslicové informace z počítače na analogový signál, který může řídit pomocí vhodných ovladačů parametry řízeného procesu.

Analogový vstupní podsystém je charakterizován rychlostí, s kterou převádí analogový signál na číslicový tvar - rychlostí vzorkování, počtem analogových vstupů, které mohou být zpracovány multiplexním způsobem, dále přesností (počtem diskrétních úrovní), rozsahem zpracovaného napětí, šumem, stabilitou atd.

Analogový vstupní podsystém pracuje asynchronně vzhledem k centrální jednotce počítače. Centrální jednotka musí vyvolat počátek analogově číslicového převodu a určit adresu analogového vstupu, který má být zpracován. Převod analogového signálu na číslicový tvar se uskutečňuje s určitým zpožděním, určeným časovým rozdílem mezi příkazem k převodu a vznikem číslicového tvaru signálu.

Analogový vstupní podsystém přenáší do počítače kromě číslicově zakódované analogové veličiny, někdy ještě další informace, např. překročení předem zvolené meze, poruchu a pod. Pokud má analogový vstupní podsystém automatickou volbu měřicího rozsahu, je nezbytné počítači předat informaci o použitém rozsahu. Někdy se předává tato informace počítači jen v případě, že jsou překročeny předem nastavené úrovně. Pak signál představuje často požadavek přerušení.

Analogové výstupní přídavné zařízení procesu (analogový výstupní podsystém) umožňuje řízení analogových zařízení, např. na-

stavení pracovních bodů, zapisovačů a ventilů počítačem. Tento podsystém je, podobně jako analogový vstupní podsystém, chakterizován rychlostí převodu, přesností, šumem a stabilitou. Někdy pracuje multiplexním způsobem, což značí, že výstup z číslicově analogového převodníku je připínán k obvodům, které mají schopnost pamatovat analogovou veličinu po určitou dobu. Rovněž analogové výstupní zařízení pracuje zpravidla asynchronně vzhledem k centrální jednotce počítače. Zpoždění analového výstupního zařízení je obvykle zanedbatelné vzhledem k analogovému vstupnímu zařízení; mnohdy odpovídá době zápisu informace do registru číslicově analogového převodníku.

Pro činnost analogového výstupního podsystému je třeba kromě přenášených dat určit adresu číslicově analogového převodníku a adresu výstupu (pokud je výstupní signál multiplexován). Změna výstupu započne obvykle samočinně po naplnění registru v číslicově analogovém převodníku.

Číslicová přídavná zařízení

Číslicová přídavná zařízení (číslicový vstupní, resp. výstupní podsystém) přijímá informace ze snímačů napojených na řízený proces v číslicovém tvaru nebo je předává ovladačům v číslicovém tvaru. Informace ze snímače představuje jeden ze dvou možných stavů tohoto snímače podobně je tomu s informací pro řízení ovladače. Např. stav spínače v procesu indikuje, zda je motor zapnut či vypnut. Podobně počítač ovládá relé, které zapíná motor. Stavy uvedených diskrétních veličin můžeme vyjádřit jedním bitem, např. sepnutému spínači můžeme přiřadit stav 1, rozepnutému stav 0.

Všimněme si, že zde je jen jeden bit nositelem informace. Jednotlivé bity ve slovu dat nemusí být ve vzájemné vazbě, jak je tomu ve slovu vyjadřujícím ve dvojkové soustavě určité číslo. Obvykle však tyto bity seskupujeme do slov. Jednotlivé stavy bývají zapsány do jednotlivých klopných obvodů paralelního posuvného registru, odkud se informace ve vhodný okamžik sériově či paralelně přenese do počítače. V počítači jsou jednotlivé bity zpracovány pomocí logických instrukcí; k tomu účelu může obsahovat programové vybavení speciální podprogramy.

Některé snímače poskytují sled impulsů, např. snímače polohy, turbinkový měřič toku kapaliny či plynu. V některých případech

značí každý impuls posun o určitou vzdálenost či určité množství prošlé kapaliny. Impulsy by mohl čítat přímo počítač a případnou zvolenou polohu či množství kapaliny nastavit pomocí přerušovacího systému. Vhodnější řešení je přiřazení samostatného čítače přímo snímači. Čítač pak vyšle do počítače informaci po přijetí určitého, předem zvoleného počtu impulsů, po jejimž předání se automaticky vynuluje.

Některé ovladače např. obsahují krokový motor, který potřebuje pro buzení sled impulsů. V tomto případě výstupní číslicový podsystém generuje určitý počet impulsů podle obsahu registru, který byl naplněn centrální jednotkou.

2. PROGRAMOVÁNÍ MIKROPOČÍTAČŮ

Pod pojem technické vybavení (hardware) zahrnujeme všechny elektrické, elektronické a mechanické součástky, které vytvářejí systém. Pod pojem programové vybavení (sofware) zahrnujeme programy, které jsou aplikovatelné (provozovatelné) na technickém vybavení. Někdy pod tento pojem též zahrnujeme dokumentaci přidruženou k programům, tj. pravidla a návody k jejich užívání.

Toto rozdělení systému na technické a programové vybavení částečně zkomplikoval výskyt permanentních pamětí; některé mikropočítače mají v této paměti zapsané „speciální" programy, které zpravidla nelze měnit a mohou provádět jen jednu činnost. Permanentní paměť v uvažovaném případě zasahuje do technického i programového vybavení; tato skutečnost si vynutila pojem mikroprogramové vybavení (firmware).

2.1. Základní poznatky a úvahy o programování

Základní principy programování mikropočítačů jsou stejné jako programování minipočítačů či větších počítačů, neboť mikropočítač je určitým druhem počítače. Je-li vybaven dostatečnou kapacitou paměti a vhodnými přídavnými zařízeními, může provádět stejné druhy výpočtů jako minipočítač, zpravidla však pomaleji.

Sestavení programu pro mikropočítač klade obvykle určité specifické požadavky plynoucí z možností a vlastností mikropočítače. Programátor se především snaží více než u kteréhokoliv jiného počítače „minimalizovat" program s ohledem na kapacitu paměti. Toto minimalizační úsilí je často znásobeno masovou výrobou počítačového systému s určitým programem. V tomto případě náklady na programování v přepočtu na jeden systém jsou nízké,

převažuje cena paměti. Při programování je třeba často přihlédnout k rychlosti mikropočítače, případně k omezenému souboru přídavných zařízení.

Programování mikropočítače je obvykle v současné době obtížnější než většího počítače, což je způsobeno řadou faktorů, z nich nejzávažnější jsou:

- omezený soubor přídavných zařízení a jejich malá rychlost (často z důvodu ceny jen dálnopis)
- málo dokonalý nebo žádný operační systém
- používání nižších programovacích jazyků
- málo účinný soubor instrukcí.

V hrubých rysech lze říci, že současné programování mikropočítačů odpovídá dokonalostí počáteční éře minipočítačů (šedesátá léta). Je však třeba mít na paměti dvě závažné skutečnosti, které napovídají rychlou progresi v programování mikropočítačů:

a) mikropočítač má za sebou poměrně krátkou dobu vývoje s nebývalou progresí
b) neustále se zdokonaluje technické vybavení mikropočítačů, zvláště díky rozvoji integrované techniky.

Sestavení programu pro mikropočítače znamená sestavení sledu kódovaných instrukcí. Každá instrukce zpravidla obsahuje operační znak a adresu místa v paměti, kde je uložen příslušný operand. Každý mikropočítač má určitý operační kód, což je množina operačních znaků, z nichž každému je jednoznačně přiřazena určitá operace. Operační kód, který často nazýváme soubor instrukcí je výsledkem předcházejících zkušeností systémového konstruktéra a programátora. Je volen tak, aby pomocí něho mikropočítač nejúčinněji řešil různé úlohy.

2.2. Sestavy mikropočítačů

Mikropočítače se používají v široké škále sestav, tj. v různé složitosti technického vybavení. Z hlediska programování můžeme sestavy mikropočítačů rozdělit do následujících kategorií:

1) Mikropočítače s minimálním technickým vybavením.

Sestava obsahuje: mikroprocesor, omezený počet permanentních pamětí, někdy paměti s libovolným výběrem a jednoduché obvody pro vstup - výstup. Aplikační oblasti: systémy, ve kterých bylo dosud

ekonomicky neúčelné používat počítačový systém, náhrada neměnných logických obvodů.

2) Mikropočítače s malým rozsahem technického vybavení. Sestava obsahuje: mikroprocesor, permanentní paměti, paměti s libovolným výběrem, někdy kanál blokových přenosů, jednoduchá přídavná zařízení (např. analogově číslicový a číslicově analogový převodník, světelné indikátory stavů, přepínače). Aplikační oblast: náhrada neměnných logických obvodů mikropočítačovým systémem.

3) Mikropočítačové systémy se středním rozsahem technického vybavení. Sestava obsahuje bloky uvedené v bodě 2 a dále složitější přídavná zařízení např. dálnopis nebo snímač a děrovač děrné pásky. Aplikační oblasti: úlohy dříve řešené jednoduchými minipočítači. Tato sestava je též často používaná jako mikropočítačový vývojový systém, tj. systém, na kterém se provádí vývoj uživatelských programů.

4) Mikropočítačový systém s velkým rozsahem technického vybavení. Tento systém se svými možnostmi přibližuje nebo kryje s minipočítači. Sestava obsahuje bloky uvedené v bodě 3, dále řídicí panel operátora, velkokapacitní přídavné paměti (např. paměť s magnetickým pružným diskem, kazetová magnetická paměť) a abecedně číslicovou zobrazovací jednotku. Aplikační oblasti: složitější zpracování dat, řízení procesů. Tato sestava se též používá jako mikropočítačový vývojový systém.

Pomocí sestav uvedených pod bodem 1 a 2 je obtížné vyvíjet program. Pro tento účel se obvykle používají mikropočítačové vývojové systémy se sestavou uvedenou pod bodem 3 nebo 4, nebo větší počítače, např. minipočítače, které v tomto případě nazýváme hostující počítače.

Mikropočítačové vývojové systémy bývají vybaveny „pomocnými" (systémovými) programy, které jsou uloženy na děrné pásce nebo ve vnějších velkokapacitních pamětech. V současné době se tyto programy často ukládají do permanentních pamětí a nazýváme je rezidentní programy. Mezi tyto programy patří: úpravný program (editor), zaváděcí program (loader), ladicí program (debugger), asembler a kompilátor.

Při použití hostujícího počítače pro vývoj uživatelského programu se používají nepřímé programy, které jsou obvykle psané

v jazyku FORTRAN. Mezi tyto např. patří: nepřímý asembler, makroasembler, simulátor (simulační program), nepřímý kompilátor (např. PL/M).

2.3. Členění programového vybavení

Programové vybavení můžeme rozdělit do tří skupin:

1) Uživatelské programy (někdy je nazýváme aplikační programy; aplications software).
Tyto programy jsou „zadávány" počítači. Jinými slovy jsou to programy sestavené pro řešení úloh, pro které byl mikropočítač koupen. Uživatelský program je obvykle sestavován pro každou úlohu. Pro úlohy, které se častěji vyskytují, jsou ukládány do knihovny programů; nazýváme je „aplikační bloky"(applications packages). Tyto aplikační bloky, které zpravidla poskytují výrobci počítačů, nejsou obvykle přímo použitelné pro konkretní úlohu, usnadňují však sestavu programu tím, že mohou být buď upraveny nebo jsou alespoň vodítkem pro sestavu programu. Aplikační bloky jsou sestavovány obvykle pro určité třídy úloh, např. statistickou analýzu, komunikační protokoly atd.

Zvláštní třídu aplikačních bloků tvoří programy resp. podprogramy určené pro vědecko-technické výpočty, např. inverzi matic.

2) Pomocné programy (pomocné programové vybavení; support software). Jsou to programy sestavené a dodávané výrobci počítačů. Usnadňují sestavení a zkoušení uživatelských programů. Do kategorie pomocných programů zahrnujeme: zaváděcí program (loader), úpravný program (editor), ladicí program (debugger), emulátor (program pro emulaci jednoho počítače na jiném počítači), asembler, kompilátor atd. Někdy tyto pomocné programy (kromě posledních dvou, které jsou překladače) nazýváme vývojové programy.

Určitou třídou pomocných programů jsou podprogramy ve strojovém kódu, které mohou být začleněny do uživatelských programů. Typickým příkladem jsou matematické podprogramy, např. aritmetika v pohyblivé řádové čárce, násobení, trigonometrické funkce. Do této kategorie patří též podprogramy pro vstup-výstup, podprogramy pro převod z dvojkové do dvojkově desítkové soustavy atd.

3) Systémové programy (systémové programové vybavení; systems software).
Uživatelský program může být sestaven tak, že počítač s tímto programem vykonává požadovanou funkci. Některé dílčí úlohy se však opakují při různých aplikacích počítače. Příkladem může být přenos informací mezi centrální jednotkou a pamětí s pružným diskem.

Programy určené pro „chod a organizaci" počítače nazýváme systémové programy. Tyto systémové programy především „osvobozují" uživatelské programy od obecné činnosti související s organizací standardních systémových otázek. Ve vývojové etapě mikropočítačového systému se též může projevit účelnost systémového programu, který organizuje pomocné nebo alespoň vývojové programy.

Operační systém, který je součástí systémových programů, je soubor programů, který řídí provádění jednotlivých programů. Operačních systémů je celá řada, liší se dokonalostí, rozsahem, cenou, účinností atd. Může do značné míry ovlivnit stupeň a snadnost využití technického vybavení.

Multiprogramovací operační systém umožňuje multiprogramové využití počítače tj. současné řešení více programů. Operační systém pro činnost v reálném čase umožňuje využít počítač, jehož činnost je vázána na reálný čas.

Poznamenejme, že mnohé mikropočítače nejsou vybaveny operačním systémem.

Uživatelské programy sestavuje uživatel počítače. Pomocné a systémové programy sestavuje a dodává výrobce počítače; můžeme je považovat za součást počítače. Operační systém bývá též závislý na sestavě počítače. Např. diskově orientovaný operační systém (DOS, disk operating system) je učen pro počítač vybavený magnetickou diskovou pamětí.

Je třeba zdůraznit, že rozdělení programového vybavení mikropočítačů a dokonce větších počítačů není obecně ustáleno, nehledě k tomu, že toto vybavení u jednotlivých typů počítačů se více či méně liší. Často používané členění programového vybavení před érou mikropočítačů a které je ovšem použitelné i u mikropočítačů, je následující:

I. Systémové vybavení

II. Uživatelské (aplikační) programy

Při tomto členění systémové vybavení dále rozdělujeme na:

A - Operační systém
B - Překladače (Translators)
C - Diagnostické programy
D - Standardní podprogramy (knihovna podprogramů)

Někdy bývají z operačního systému vyčleňovány služební programy, kam zahrnujeme např. zaváděcí program, program pro tisk, ladění atd.

Někteří autoři programové vybavení člení jen na :

a) Operační systém
b) Uživatelské programy
c) Pomocné (služební) programy.

Pro úplnost uveďme, že členění a též názvosloví v anglosaské literatuře není dosud jednotné. Např. operační systém je někdy nazýván Supervisory Programs, Control Programs, Executive, Monitors nebo Firmware[1]. Uživatelské programy jsou nazývány Applications Programs, Operational Programs, Problem Programs, Processors nebo také Ordinary Processors. Nejednotnost v členění je celkem pochopitelná, neboť programy, které tvoří vybavení počítače na sebe často velice úzce navazují, též velikost počítače (sestava mikropočítače) zde hraje určitou roli. Nejednotnost v názvech je často vyvolána firemním označením, které se více či méně obecně ujalo.

Dále budeme respektovat prvně uvedené členění programového vybavení tj. budeme jej dělit na uživatelské programy, pomocné programy a systémové programy.

2.4. Programovací jazyky

Program je sled instrukcí resp. příkazů, podle kterých počítač zpracuje vstupní data a tím řeší úlohu. Sestavuje jej programátor na základě analýzy daného problému.

---

1) Některé firmy spojují s pojmem Firmware otázku mikroprogramování. Monitor a supervisor bývá jen částí operačního systému.

V počátcích počítačů (1.generace počítačů) programátor sestavoval programy výhradně ve strojovém kódu. Nyní se u počítačů prakticky nepoužívá, u mikropočítačů jen výjimečně pro krátké programy. Jednotlivé instrukce programu ve strojovém kódu zapisované ve dvojkové soustavě odpovídají instrukcím ve stroji. Sestavení programu je značně zdlouhavé, náročné a pravděpodobnost chyby je značná.

Částečné ulehčení práce programátora přineslo použité osmičkové nebo šestnáctkové číselné soustavy pro záznam strojového kódu. Uvažujme dvanáctibitové slovo instrukce; nechť první tři bity představují operační znak, dalších devět bitů adresu. Předpokládejme, že operační znak pro sčítání je 001 a adresa paměti je 115_{10} (= 163_8 = 001110011_2). Strojový kód pro tuto zjednodušenou instrukci je 001001110011 (= 1163_8). První číslice v osmičkové soustavě vyjadřuje operační znak, ostatní číslice vyjadřují adresu.

Pro zlepšení adresování můžeme použít první bit adresy pro indikaci absolutního (přímého) adresování a nepřímého adresování. Tím zbývá pro vyjádření přímé nebo nepřímé adresy osm bitů. Představuje-li 0 přímé a 1 nepřímé adresování, pak 1163_8 představuje v předcházejícím příkladu absolutní (přímé) adresování. Je-li adresování nepřímé (tj. 163_8 vyjadřuje adresu adresy operandu) pak příslušná instrukce je 001101110011 (= 1563_8). Je zřejmé, že zápisem v osmičkové soustavě jsme v tomto případě zatemnili způsob adresování.

V další etapě vývoje počítačů bylo sestavení programu usnadněno zavedením jazyka symbolických adres (někdy nesprávně jazyk asemblerů, symbolický jazyk). V tomto jazyku operační znak instrukce vyjadřujeme písmeny místo číslicemi ve dvojkové soustavě a tato písmena jsou obvykle mnemotechnickou zkratkou příslušné operace. Další zjednodušení, které přináší tento jazyk je zavedení symbolických adres místo absolutních adres.

Pro ilustraci programování v jazyku symbolických adres si uvedeme sečtení dvou čísel uložených v paměťových buňkách A a B; výsledek sečtení má být uložen do paměťové buňky C:

LOAD ACC, A	(zaveď do střadače obsah paměťové buňky A)
ADD, B	(přičti k obsahu střadače obsah paměťové buňky B)
STORE ACC, C	(ulož obsah střadače do paměťové buňky C)

Je zřejmé, že zápis programu v jazyku symbolických adres je podstatně jednodušší než zápis programu ve strojovém jazyku. Všimněme si, že počet instrukcí v symbolickém a strojovém jazyku však zůstává stejný.

Počítač je schopen zpracovat jen program ve strojovém jazyku, proto je třeba zapsaný program v jazyku symbolických adres přeložit (převést) do strojového jazyka. Tento překlad provede počítač pomocí programu, který nazýváme asembler nebo translátor.

Programování v jazyku symbolických adres má některé základní nedostatky: je zdlouhavé, programátor musí znát jednotlivé instrukce použitého počítače, sestavený program je nepřenosný na jiný typ počítače a programátor musí znát alespoň rámcově organizaci počítače.

Pro usnadnění sestavy programu byly vypracovány programovací jazyky vyšší úrovně, které nazýváme vyšší programovací jazyky nebo vyšší problémově orientované jazyky. Tyto jazyky vycházejí ze skutečnosti, že některé operace určitých tříd úloh se často opakují. Např. vynásobení dvou čísel můžeme zapsat v tomto jazyku jedním příkazem, např. A vynásob B. Je samozřejmé, že ve strojovém jazyku, kterému počítač rozumí, tento příkaz obsahuje několik kroků. Značným snížením počtu programovacích kroků podstatně usnadníme programování. Vyšší programovací jazyky jsou též obvykle přenosné z jednoho typu počítače na jiný.

Strojový jazyk a jazyk symbolických adres nazýváme strojově orientované jazyky, neboť jsou nepřenosné z jednoho typu počítače na jiný.

Počítač je schopen řešit úlohu jedině ve strojovém jazyku. Proto program napsaný v jazyku symbolických adres nebo ve vyšším programovacím jazyku je třeba počítačem přeložit do strojového jazyka pomocí programu, který nazýváme překladač. Jak jsme si již uvedli, pro překlad programu napsaného v jazyku symbolických adres používáme překladač, který nazýváme asembler nebo sestavovací program; pro překlad programu napsaném ve vyšším programovacím jazyku používáme překladač, který nazýváme kompilátor nebo kompilující program. Pro některé vyšší programovací jazyky používáme pro překlad interpretační program.

Jazyku symbolických adres a vyšším programovacím jazykům, vzhledem k jejich důležitosti, budou věnovány samostatné náledující odstavce.

2.5. Jazyk symbolických adres

Jazyk symbolických adres je dosud nejpoužívanější programovací jazyk u mikropočítačů. Jak jsme již uvedli, v tomto jazyku se pro operační znaky používají mnemotechnické zkratky operací, pro adresy se používá též symbolických označení.

Uveďme si příklad programu pro sečtení čísel 8 + (-5) + 2 = 5. Odečtení pěti provedeme přičtením druhého doplňku tohoto čísla; $5_{10} = 0101_2$, druhý doplněk je 1010. Tedy

$$\begin{array}{ll} 1000 & = 8_{10} \\ 0010 & = 2_{10} \\ 1011 & = \text{druhý doplněk } 5 \\ \hline 0101 & \text{výsledek} \end{array}$$

Příslušný program, který je uveden v tab. 2.1 začíná vytvořením doplňku. V levé části tabulky je uveden program při použití absolutního adresování, v pravé části je použito symbolické adresování pomocí návěští. Jednotlivé symboly „Start", X, Y někdy nazýváme identifikátory.

Zkratky operací (operační znaky) bývají v anglickém jazyku. Např. ADD může být použito pro vyjádření operace přičtení k obsahu střadače. Dále symbol * může být použit pro vyjádření nepřímého adresování, vypuštění tohoto symbolu značí přímé (absolutní) adresování.

Vraťme se k příkladu, který jsme řešili v předcházejícím odstavci ve strojovém jazyku. V jazyku symbolických adres instrukce ADD 163 značí (podle naší zvolené symboliky) „přičti obsah adresy 163_8 k obsahu střadače". Ve strojovém jazykujsme tuto instrukci označili 001001110011 (= 1163_8). Instrukce ADD *163 je stejná jako předcházející, avšak je použito nepřímé adresování. Ve strojovém jazyku jsme tuto instrukci vyjádřili 00110111001 (= 1563_8).

Instrukce ADD 163 sestává ze šesti znaků a mezery a tvoří zdrojový kód. Ze sledu znaků rozpozná asembler tuto instrukci a vy-

tvoří příslušný cílový, tj. strojový kód.

Dosud jsme uvažovali v instrukci ADD 163 absolutní adresování, tj. 163_8 vyjadřuje skutečnou adresu paměti. Jak jsme již uvedli, v jazyku symbolických adres používáme při zápisu programu symbolické vyjádření adres; tento symbol nazýváme identifikátor. Údaj příslušející identifikátoru může být uveden na počátku nebo na konci programu, tj.

```
A = 163
ADD A
  :
```

nebo

```
ADD A
  :
A:  163
```

Pro zjednodušení programování v jazyku symbolických adres je někdy též možné použít pro adresování jednoduché aritmetické operace. Např. ADD 216 + 111 asembler přeloží 1327_8, neboť 216_8 + 111 = = 327_8 a $001_2 = 1_8$ značí sčítání. Podobně ADD A + 7 asembler přeloží 1172_8, neboť jsme si zvolili A = 163_8.

Asembler může též umožnit překlad programu, který používá číselné soustavy s různým základem. Je samozřejmé, že v tomto případě musíme uvést deklaraci základu číselné soustavy. Např. ze zdrojového kódu

```
. RADIX 10  (deklarace číselné soustavy)
  :
  ADD 115
```

generuje asembler ve dvojkové soustavě cílový kód odpovídající 1163_8, neboť $115_{10} = 163_8$. Po deklaraci určitého základu je asemblerem tento základ stále uvažován, dokud se nevyskytne deklarace nového základu.

Jiný typ jazyka symbolických adres může používat jen osmičkovou a desítkovou soustavu. V tomto případě můžeme desítkovou soustavu označit tečkou a osmičkovou soustavu ponechat bez označení. Při tomto označení instrukce ADD 115. a ADD 163 jsou ekvivalentní.

Rozdělení instrukcí do polí

Příkaz (instrukce) v jazyku symbolických adres je obvykle rozdělen do čtyř polí

návěští: kód operand; komentář.

Jednotlivá pole jsou vhodně oddělena, např. mezerou, nebo znaky :, ;, /, -. Každá instrukce však nemusí obsahovat všechna pole.

Návěští je symbolická adresa. Při překladu z jazyka symbolických adres je návěští přiřazena adresa, na které je instrukce uložena. To usnadňuje odvolávku na instrukci v každém bodě programu.

Kód je symbolické (mnemotechnické) vyjádření instrukce. Pole operandu obsahuje symbolickou adresu operandu. Pole komentáře počítač ignoruje: slouží jen pro dokumentaci.

2.5.1 Pseudoinstrukce

Program v jazyku symbolických adres nemusí obsahovat jen instrukce vyjadřující operace ve strojovém kódu, ale též příkazy pro překlad, tj. pro asembler. Tyto příkazy negenerují cílový kód, avšak pomáhají při generaci strojových instrukcí. Nazýváme je pseudoinstrukce. Jinými slovy pseudoinstrukce je instrukce programu psaného ve zdrojovém jazyku, která se nepřekládá do cílového jazyka, avšak pouze ovlivňuje překlad, např. slouží pro rezervaci míst v paměti. Pseudoinstrukce jsou často identifikovány tečkou na počátku. Příklady dvou pseudoinstrukcí jsme již uvedli: přiřazení identifikátoru (A = 163) a volba základu číselné soustavy (.RADIX 10).

Přiřaďující pseudoinstrukce přiřazuje čísla určitému symbolu. Např. pseudoinstrukce X = 60 nebo X EQ 60 zajistí, aby se symbolu X přiřadila hodnota 60.

Dvěma symbolům může být přiřazena stejná adresa. Např. X = FR zajistí, že X a FR jsou zaměnitelné. Tato operace je užitečná především při navazování programů.

Jelikož asembler při překladu musí přiřadit adresy datům a instrukcím, je třeba asembleru specifikovat počáteční adresu. Proto na počátku programu v jazyku symbolických adres se vyskytuje pseudoinstrukce „počátek" (ORIGIN). Specifikuje adresu, od které mají být uloženy následné instrukce. Při překladu asemblerem jsou používány následné adresy až do okamžiku nové pseudoinstrukce „počátek".

Pro pseudoinstrukci „počátek"se používá různých označení, např. ORG, BEG nebo * .

Pro specifikaci adresy může sloužit též umísťující pseudoinstrukce, která má přímý vztah k čítači instrukcí. Obvykle se označuje jen tečkou. Touto pseudoinstrukcí se též určuje adresa v operační paměti. Např.

```
. = 600
.WORD 1, 2, 3
. = 800
.WORD 5, 6, 7, 10
```

umístí čísla 1_8, 2_8, 3_8 na adresy 600, 601, 602 a čísla 5_8, 6_8, 7_8 a 10_8 umístí na adresy 800, 801, 802 a 803. Pseudoinstrukce .WORD značí, že následující sled čísel má být umístěn na následné adresy.

Podobnou funkci má bloková pseudoinstrukce. Pomocí ní můžeme definovat sled čísel, např. tabulku. Nepř. pseudoinstrukce TAB: BLK 22, 4, 75, 40 způsobí, že čtyři čísla 22, 4, 75, 40 se umístí na následné adresy paměti počínaje adresou TAB. Referencí na TAB se adresuje první číslo, t.j. 22. Většina asemblerů umožňuje relativní adresování pro výběr jiného čísla z bloku, např. TAB + 3 se vztahuje k číslu 40. Pro blokovou pseudoinstrukci se často používá symbol BLK nebo DB.

Další pseudoinstrukce se používají pro rozlišení přemísťujícího a absolutního asembleru.[1] Např.

```
      .ABS
      . = 200
A:    ADD .+ 20
      .REL
      . = 10
B:    ADD A
      ADD B + 10
```

1) Absolutní a přemísťující asembler úzce souvisí se zaváděcím programem, který probereme později.

přiřazuje adresu 200_8 (absolutně) k návěští A a adresu 10_8 (přemístitelně) k návěští B. Instrukce ADD A bude umístěna zaváděcím programem, operand A má absolutní umístění. Instrukce ADD B + 10, která je umístěna na adrese 11_8 (přemístitelně) se vztahuje na operand umístěný na adrese 20_8 (přemístitelně).

Pro označení konce zdrojového programu se používá pseudoinstrukce „konec" (END). Jestliže asembler narazí na tuto pseudoinstrukci, zastaví proces překladu.

2.5.2 Literál

Umožňuje-li asembler použití literálů, pak přičtení konstanty k obsahu střadače může být uskutečněno jedinou instrukcí, např. ADD = 7. Rovnítko je obvykle používáno jako symbol, který signalizuje asembleru, že následující je literál. V tomto případě počítač symbolu 7 přiřadí adresu a číslo 7 v něm uloží. Je-li použit později v programu stejný literál, asembler rozpozná symbol a použije adresu, kde je číslo 7 uloženo. Použití literálů snižuje počet symbolů, které je třeba definovat na počátku programu.

Pokud asembler neumožňuje použití literálů, je samozřejmě možné uskutečnit přičtení čísla 7 k obsahu střadače tím, že na symbolické adrese označené návěštím SEDM si uložíme číslo 7 (SEDM EQ 7), které pak instrukcí ADD SEDM přičteme k obsahu střadače. Jinou metodou pro přičtení konstanty k obsahu střadače je použití instrukce okamžitého (následného) přičtení (add-immediate instruction), např. pro Intel 8080 ADI 7.

Použití literálu vzhledem k oběma uvedeným způsobům je výhodnější; oproti prvnímu způsobu snižuje počet symbolů, které je třeba definovat na počátku programu, vzhledem k druhému způsobu je účinnější.

2.5.3 Asembler

Asembler je program, který převádí program napsaný v jazyku symbolických adres do programu ve strojovém jazyku. Vyžaduje-li uživatel počítače překlad programu v jazyku symbolických adres, sdělí tento požadavek počítači a ten přesune asembler z vnější paměti (např. děrné pásky) do operační paměti.

Program napsaný v jazyku symbolických adres počítač postupně čte a zapisuje dle příkazu asembleru do operační paměti za současné analýzy programu v jazyku symbolických adres. Paměťovým buňkám se přiřazují proměnné a instrukce jsou převáděny do strojového jazyka. Některé typy chyb způsobené programátorem asembler vyhledá a vypíše pomocí tiskárny (dálnopisu).

Instrukce ve strojovém jazyku získávané překladem se postupně ukládají do vnější paměti, např. na děrnou pásku. Tento cílový program je třeba pro uskutečnění výpočtu opět zavést do počítače. Výpočet se provede pouze za předpokladu, že se v programu nevyskytují chyby. Tisk, který probíhá současně s překladem, může programátor použít pro snažší vyhledání chyb.

Příklad možného tisku při překladu je následující:

```
A    VAR        1        ----
B    VAR        2        ----
C    VAR        3        ----
D    VAR        4        ----
:    :          :        :
     LDA C      50       6003
     ADD B      51       2002
     STA A      52       6801
     IAP NEXT   53       8437
     STA D      54       6804
NEXT LIX A      ILLEGAL  55
```

První tři pole (sloupce) jsou přímým přepisem uživatelského programu tj. programu v jazyku symbolických adres. Další sloupec je důsledek překladu. Pro usnadnění práce programátora je tento sloupec, který odpovídá strojovému jazyku, zapisován v šestnáctkové soustavě.

Jednotlivé mnemotechnické zkratky instrukcí značí: VAR - VARIABLE ASSIGNMENT (proměnná), LDA - LOAD ACC (naplň střadač), ADD - sečti, STA - STORE ACC (ulož obsah střadače), IAP - IF ACC POSITIVE, TRANSFER TO (je-li obsah střadače kladný, skoč na), LDX - LOAD INDEX (naplň index).

První čtyři řádky přiřazují paměťové buňky proměnným A, B, C, D. U instrukce IAP je použito symbolické adresy NEXT (další), což je výhodnější než uvádět absolutní adresu na kterou má instrukce IAP provést první skok. Jelikož asembler nezná instrukci LIX (programátor udělal chybu, správně LDX), vytkne ILLEGAL.

Překlad programu v jazyku symbolických adres asemblerem probíhá u mikropočítačů často v několika krocích; používáme jednoprůchodový až tříprůchodový překlad. Např. v prvním kroku ještě před generováním cílového programu asembler prohledává program napsaný v jazyku symbolických adres, vyhledává a třídí symboly použité v programu, přiřazuje adresy návěštím (pokud nebyly přiřazeny pseudoinstrukcemi) dále se zabývá informacemi o přemísťování, globálními proměnnými atd. Na konci prvního kroku asembler zajistí tisk užitečných informací, např. tabulky symbolů.

V druhém kroku se provádí vlastní překlad zdrojového programu (programu v jazyku symbolických adres) do cílového programu (programu ve strojovém jazyku). Cílový program mikropočítač zapíše na vhodné medium (na děrnou pásku, na magnetický pružný disk, do magnetické kazetové paměti atd.). Během překladu bývá tištěna každá instrukce v jazyku symbolických adres, strojová instrukce a její adresa. Tisk se provádí v osmičkové nebo šestnáctkové soustavě a nazýváme jej výpis překladu (assembly listing).

U jednoduchých mikropočítačových sestav, které používají jako přídavné zařízení jen dálnopis sloužící pro tisk i jako děrovač děrné pásky, nelze současně děrovat cílový program a tisknout výpis překladu. Tisk „výpisu programu" se provádí až ve třetím kroku.

U mikropočítačů se jen zřídka používá jednoprůchodový asembler, který umožňuje obvykle rychlý překlad, avšak s určitými omezeními ve zdrojovém programu: nelze provádět dopředné odkazy (odkazy na následující instrukce). Dvouprůchodový nebo tříprůchodový asembler

je nejčastější.

Při sestavě programu v jazyku symbolických adres je účelné uvádět komentář. Asembler jej při překladu ignoruje, avšak ve výpisu překladu je vytištěn.

2.5.4 Vlastní a nevlastní asembler

Asembler patří k nejdůležitějšímu programovému vybavení současných mikropočítačů. Používají se dva druhy asembleru: vlastní (self-assembler, resident-assembler) a nevlastní (gross-assembler). Překlad s vlastním asemblerem provádí přímo mikropočítač.

Překlad nevlastním asemblerem se provádí na jiném, větším počítači, který nazýváme hostující počítač nebo překládající počítač. Počítač, který provádí předložený program nazýváme cílový počítač.

Každý, v současné době vyráběný mikroprocesor je vybaven jedním nebo více nevlastními asemblery. Nevlastní asembler se obvykle sestavuje již během vývoje mikroprocesoru a slouží v této fázi při jeho simulaci k optimalizaci souboru instrukcí a architektury.

Použití nevlastního asembleru je velmi časté, neboť větší počítače jsou vybaveny dokonalejšími přídavnými zařízeními a mají též větší kapacitu paměti. Vlastní i nevlastní asembler je sestaven pro určitý typ mikropočítače. Obvykle je však nevlastní asembler napsán ve vyšším jazyku, jako je např. FORTRAN; umožňuje se tím použití různých počítačů. Činnost vlastního asembleru je značně závislá na vstupních a výstupních zařízeních mikropočítače. Tato specifický systémová závislost může v některých případech způsobit určité potíže.

Vlastní asembler může být zapsán na děrné pásce nebo v magnetické paměti s pružným diskem, častěji však bývá zapsán v permanentní paměti. Vlastní asembler zapsaný v permanentní paměti někdy nazýváme rezidentní asembler.

2.5.5 Podprogram a makroinstrukce

Vyskytuje-li se v jednom programu vícekrát stejný sled instrukcí, obvykle jej vyjímáme z hlavního programu a vytvoříme z něj podprogram. Tento podprogram je vyvolán vždy v žádaném místě hlavního programu a na konci podprogramu je příkaz k návratu do hlavního

programu.

Vyvolání podprogramu provádíme obvykle instrukcí CALL, která je u většiny mikropočítačů víceslovná. Některé mikropočítače mají instrukci RESTART (např. 8080 instrukci RST), což je v podstatě jednoslovné vyvolání podprogramu. Touto instrukcí lze vyvolávat podprogramy z omezeného počtu adres, její použití je výhodné při zpracování přerušení nebo podprogramů v případě, kdy klademe důraz na rychlost.

Jiný způsob vkládání víckrát se vyskytujícího sledu instrukcí v programu je použití makroinstrukcí. Makroinstrukce je pseudoinstrukce. Pro její aplikaci je třeba tří následujících kroků: definovat makroinstrukci, provést odkaz na makroinstrukci pomocí návěští a „rozepsat" makroinstrukci asemblerem při překladu.

Makroinstrukce [1] označuje v jazyku symbolických adres předem určený sled strojových instrukcí. Má-li se makroinstrukce vyskytnout v programu, uvede ji programátor pomocí návěští. Asembler ji rozpozná a nahradí příslušným sledem strojových instrukcí.

Makroinstrukce usnadňuje programování tím, že programátor nemusí psát celý sled opakujících se instrukcí, kdykoliv se má vyskytnout v programu, ale nahradí jej makroinstrukcí.

Makroinstrukce nejsou začleňovány do cílového programu jako podprogram a nesnižují počet strojových instrukcí, tj. nesnižují požadavky na kapacitu paměti. (obr. 2.1.) Na druhé straně makroinstrukce proti podprogramům urychlují provedení programu tím, že nevyžadují vyvolání a návrat z podprogramu. Je-li sled vícekrát se opakujících instrukcí delší, je zpravidla výhodné použít podprogram, v opačném případě makroinstrukci.

Jako příklad úspory paměti při požití podprogramu vzhledem k makroinstrukci uvažujme 10 instrukcí, které mají být vloženy 20x do hlavního programu. Při použití podprogramu je třeba paměť pro 20x vyvolání (CALL), 1x návrat (RETURN) a 10 užitečných instrukcí tj. celkem 31 pamětových míst. Při použití makrointrukce je třeba

---

1) Někteří autoři označují jako makroinstrukci jedinou „klasickou" instrukci pro odlišení od mikroinstrukce, která souvisí s mikroprogramováním.

20.10 = 200 paměťových míst (pro jednoduchost předpokládáme jednoslovné instrukce). Řešení pomocí makroinstrukcí může však být rychlejší, proto je třeba vážit použití podprogramů nebo makroinstrukcí pro každou aplikaci.

2.5.6 Makroasembler

Makroasembler je přemísťující asembler, který též umožňuje zpracování mikroinstrukcí. Jako příklad makroinstrukce si uveďme

```
. MACRO  PEPA  X, Y, Z
  :      skupina instrukcí v jazyku symbolických adres
  :      s proměnnými X, Y, Z; např. Z = (X + Y) . Y
. END M
```

Makroinstrukci, kterou jsme nazvali PEPA, např. PEPA M, N, P, můžeme použít dále libovolně v jazyku symbolických adres, vždy se vygeneruje příslušný cílový kód, tj. příslušný sled strojových instrukcí.

Jako další příklad použití makroinstrukce uveďme zavedení adresy do adresového registru (memory addres registr, MAR), což je častá operace. Bohužel některé mikropočítače neumožňují tuto operaci jedinou instrukcí. Uvažujme mikropočítač využívající mikroprocesor Intel 8008, u kterého adresový registr se skládá z registrů H a L. Každý z těchto registrů H a L musí být naplněn odděleně. Uvažujme dvoubytovou adresu, kterou si označme X. K zavedení této adresy do adresových registrů můžeme použít instrukce pro okamžitý přesun (instrukce MOVE IMMEDIATE, MVI):

MVI H,X SHR 8	do registru H se zavede číslo X, které je posunuto o osm míst vpravo (SHR 8, shift right)
MVI L, X AND OFFH	do registru L se zavede číslo X, na něž byl proveden logický součin s číslem $OFF_{16} = 0000000011111111_2$

Jednotlivé kroky jsou znázorněny na obr. 2.2.
Poznamenejme, že bity X_{14} a X_{15} nejsou u mikroprocesoru 8008 pro adresování použity, používají se pro jiné účely.

Adresování u uvažovaného mikroprocesoru 8008 můžeme zjednodušit, definujeme-li si makroinstrukci ADRE pro zavedení adresy do registrů H a L následujícím způsobem:

```
ADRE    MACRO  XYZ
        MVI    H, XYZ SHR 8
        MVI    L, XYZ AND ØFFH
```

XYZ je obecný parametr představující adresu; je specifikován až při odkazu na makroinstrukci (tj. při použití návěští příslušející makroinstrukci) v programu. Zavedení X do adresového registru tj. do H a L provedeme odkazem na makroinstrukci ADRE X, která je definována v programu.

Použití makroinstrukcí umožňuje emulaci [1] jednoho počítače na jiném typu počítače tím, že instrukce emulovaného počítače vyjádříme pomocí makroinstrukcí.

2.6. Zavední programů do počítače

2.6.1 Postup zavedení programů

Předpokládejme, že paměť zakoupeného počítače je úplně prázdná. Aby počítač mohl konat požadovanou funkci, je třeba do něho zavést program. Počáteční možnost komunikace uživatele s počítačem je pomocí přepínačů, které má programátor k dispozici na ovládacím panelu. Pomocí těchto přepínačů můžeme zavádět postupně jednotlivé bity slov. Je samozřejmé, že tento způsob zavedení i jednoduchého programu je značně zdlouhavý a má značnou pravděpodobnost vzniku chyby. Lepší možnost komunikace programátora s počítačem je pomocí klávesnice dálnopisu nebo snímače a děrovače děrné pásky.

První, elementární program bývá často zapisován do počítače nastavením jednotlivých adres přepínači (tlačítky), za každou adresou následuje zápis instrukce ve dvojkovém kódu (strojový program). Tento program se nazývá samozaváděcí program (bootstrap loader). Samozaváděcí program je sestaven výrobcem počítače. Účelem tohoto krátkého programu je jen aktivování určitého vstupního zařízení, např. klávesnice dálnopisu nebo snímače děrné pásky. V dru-

1) Emulace je použití určitého počítače pro provedení programu, který byl sestaven pro jiný počítač. Emulaci obvykle provádíme tím, že buď „přizpůsobíme" instrukce počítače, který provádí emulaci instrukcím emulovaného počítače pomocí mikroprogramů, nebo použijeme makroinstrukce.

hém kroku, kdy je již aktivován např. snímač děrné pásky, můžeme pomocí tohoto snímače vložit do operační paměti počítače složitější program, který je připraven na děrné pásce. Tímto programem může být asembler, který přeloží zdrojový program do cílového programu. V tomto případě zdrojovým programem je program v jazyku symbolických adres a cílovým programem je program ve strojovém jazyku, který se v této fázi vypíše např. na děrnou pásku.

V dalším kroku se zavede přeložený program z děrné pásky do operační paměti a současně se zápisem se vymaže asembler. Následuje provedení tohoto programu. Pro usnadnění oprav chyb v uživatelském programu mohou být zavedeny další programy, např. úpravný program (editor) a ladicí program (debugger).

Pro urychlení prací může být místo dálnopisu použito rychlého snímače a děrovače děrné pásky tím, že je změněna identifikační adresa v samozaváděcím programu.

2.7 Zaváděcí programy

Nyní si podrobněji probereme některé programy, s jejímiž názvy jsme se již setkali v předcházejícím odstavci. Jak jsme si již uvedli, zaváděcí program je pomocný program sestavený výrobcem počítače. Jeho úkolem je aktivovat přídavná zařízení, aby mohl být do operační paměti počítače zaveden další program. Zaváděcích programů je celá řada s různým stupněm složitosti a dokonalosti.

2.7.1 Samozaváděcí program

Samozaváděcí program je nejjednodušší zaváděcí program, který aktivuje pouze vstupní zařízení, např. snímač děrné pásky, magnetickou páskovou nebo magnetickou diskovou paměť. Je samozřejmé, že samozaváděcí program pro snímač děrné pásky je zpravidla kratší než samozaváděcí program pro magnetickou nebo diskovou paměť.

Samozaváděcí program pro snímač děrné pásky obsahuje 10 až 20 instrukcí ve strojovém kódu. Tyto instrukce postupně zajistí:
- přečtení znaku z děrné pásky (resp. si jej vyžádají)
- uložení tohoto znaku na určitou adresu operační paměti
- opakované testování snímače děrné pásky dokud tento nepřečte další znak.

V současné době samozaváděcí program bývá často uložen v permanentní paměti, což je výhodnější než jeho zavedení pomocí přepínačů na ovládacím panelu počítače.

2.7.2 Absolutní zaváděcí program

Někdy po samozaváděcím programu zavedeném pomocí přepínačů na ovládacím panelu počítače následuje zavedení absolutního zaváděcího programu, který je vkládán již pomocí klávesnice dálnopisu nebo snímače děrné pásky. Tento program umožní zavedení dalších programů, přičemž sám může provádět další funkce, např. kontrolovat formát, paritu a detekovat případné chyby.

Podobně jako samozaváděcí program může být absolutní zaváděcí program uložen v permanentní paměti. V tomto případě se omezí činnost operátora na minimum.

2.7.3 Přemísťující zaváděcí program

Uveďme si nejprve rozdíl mezi absolutním a přemístitelným strojovým kódem. Absolutní strojový kód vzniká překladem pomocí absolutního asembleru. Charakteristickým znakem je to, že adresy jsou absolutní, tj. určují skutečné místo v operační paměti. Program je zaváděn na adresy specifikované asemblerem před generováním strojového kódu (cílového kódu).

Přemístitelný strojový kód vzniká překladem pomocí přemísťujícího asembleru. Adresy generované asemblerem jsou relativní vzhledem k základní adrese. Přemísťující zavádějí program mění relativní adresy na absolutní při zavádění programu přičtením posunutí k základní (bázové) adrese. Tato základní adresa je specifikována zaváděcímu programu před zavedením programu. Opětné zavedení strojového programu na nové adrey vyžaduje pouze změnu základní adresy.

Je zřejmé, že zaváděcí program a asembler jsou v úzkém vztahu; pro přemísťující zaváděcí program je třeba použít přemísťující asembler. Asembler přidává určitou informaci ke generovanému cílovému programu, kterou využije zaváděcí program.

Uvažujme následující část programu v absolutním kódu:

```
500      posuň 550, 560; nahraď data B daty A
 :
540      data C
 :
550      data A
 :
560      data B
```

Instrukce na adrese 500 značí posunout obsah na adrese 550 na adresu 560, tj. přepsat data B daty A.

Uvažujme stejný program, který je nesprávně zavedený (počáteční adresa je 510):

```
510      posuň 550, 560
 :
550      data C
 :
560      data A
 :
570      data B
```

V tomto případě instrukce na adrese 510 přepíše data A datem C.

Stejnou část programu můžeme zapsat v přemístitelném kódu následovně:

```
 200
000'  posuň 50, 60
 :
040   data C
 :
050   data A
 :
060   data B
```

Symbol *200 značí základní adresu tj. uvedená část programu bude začínat na adrese 200. Tato hodnota je poskytována zaváděcímu programu. Čárka u 000' značí, že adresy v kódu tohoto řádku jsou relativní. Přemísťující zaváděcí program zavede uvedenou část programu na absolutní adresy 200, ...240, ...250, ...260.

Základní adresa může být dána zaváděcímu programu třemi způsoby:

1) Zapsáním speciální instrukce (pseudoinstrukce) do programu v jazyku symbolických adres, kterou přeloží asembler do strojového kódu. Tento speciální kód identifikuje zaváděcí program
2) Pomocí přepínačů na ovládacím panelu počítače
3) Pomocí spojovacího programu.

Spojovací program (linker) patří do kategorie pomocných programů. Jeho úkolem je provést vazbu mezi jednotlivými moduly cílového programu, které byly separátně přeloženy. Spojovací program přijímá „směs" absolutního a přemístitelného kódu a generuje absolutní kód, který může být zaveden absolutním zaváděcím programem.

Spojovací a zaváděcí program (linking loader) patří též do kategorie pomocných programů; je kombinací spojovacího programu a zaváděcího programu. Jsou-li spojované moduly programu plně nezávislé, spojování programu je jednoduché. Mnohdy tomu tak nebývá. U jednotlivých modulů programů jsou některé symboly pro označení dat společné. Je-li při sestavě programu používán určitý symbol v různých modulech, je deklarován asembleru jako „globální". Asembler vyhotoví seznam globálních proměnných generovanému přemístitelnému cílovému kódu, který je uspořádán pomocí spojovacího programu tak, aby globální proměnná měla jednu adresu v operační paměti.

Od zaváděcího programu mnohdy požadujeme (kromě jeho hlavní funkce spočívající v přenosu kódu z děrné pásky resp. jiné vnější paměti do operační paměti počítače) kontrolu přenosu dat. Běžná metoda kontroly spočívá ve vytvoření kontrolního součtu (checksum). V tomto případě se sčítá blok cílového kódu, který uvažujeme jako sled bytů. Negace tohoto výsledného součtu je přidána jako zvláštní byte dat k bloku kódu; nazýváme jej kontrolní součet. Je-li blok kódu zaváděn, zaváděcí program provede součet kódů a kontrolního součtu; celkový součet má být nula. Nenulový výsledek bývá indikován na ovládacím panelu počítače a upozorňuje na chybu kódu čteného zaváděcím programem. Místo chyby nemůže být identifikováno a je třeba provést nové zavedení programu.

Uvažujme nyní příklad cílového programu v absolutním kódu (absolutní strojový kód), který sestává ze 100 slov instrukcí a dat, která mají být uložena od adresy 500 a 200 slov, která mají být uložena od adresy 3000. Zaváděcí program nesmí ukládat nic na adresy 0 až 499, 600 až 2999 a od adresy 3199 do operační paměti; předpoklá-

dáme, že na těchto adresách jsou uloženy jiné užitečné programy. Tudíž kód poskytovaný asemblerem zaváděcímu programu misí obsahovat kromě cílového kódu informaci specifikující adresy paměti.

Každá skupina slov kódu, která má bezprostředně následné adresy, se nazývá blok. Před každý blok cílového jazyka umísťujeme tři slova:

1) prázdný znak nebo jiné jednoduché slovo označující počátek bloku
2) slovo vyjadřující celkový počet slov v bloku cílového kódu
3) adresu na kterou má být umístěno první slovo bloku.

Kontrolní součet se přidává na konec každého bloku.

Pro předcházející příklad budou generovány dva bloky. Kód, který se přidává zaváděcímu programu má tvar:

```
0
100
500
 ⋮        100 slov cílového kódu
kontrolní součet 1
0
200
3000
 ⋮        200 slov cílového kódu
kontrolní součet 2
```

Zaváděcí program vyžaduje při zavádění cílového programu ještě blok dat, které nazýváme přemísťující adresář (relocation directory). Spojovací resp. spojovací a zaváděcí program vyžaduje tabulku globálních symbolů, které se přidávají k asemblerem přeloženému programu; nazýváme ji globální adresář symbolů (global symbol directory).

Vypisovací program

Vypisovací program (dumper) je pomocný program, který slouží k výpisu obsahu operační paměti počítače. Obvykle pomocí něho můžeme vypsat i část obsahu paměti, kterou si zvolíme. Pomocí vypisovacího programu můžeme provést výpis i z magnetické diskové nebo páskové paměti, přičemž musí být uveden počátek a konec výpisu, např. formou počtu slov a počátku.

Některé dokonalejší vypisovací programy provádějí výpis po převodu z dvojkové do osmičkové nebo šestnáctkové soustavy.

2.8 Vyšší programovací jazyky

Ačkoliv jazyk symbolických adres podstatně zjednodušuje programování vzhledem k strojovému jazyku, mnohé nesnáze ještě zůstávají. Především každý typ počítače má svůj specifický jazyk symbolických adres, nelze jej použít na jiném typu počítače, tj. je nepřenosný. Rovněž přechod programátora k jinému typu počítače znamená učit se jinou variantu jazyka symbolických adres.

Dalším nedostatkem jazyka symbolických adres jsou poměrně jednoduché (tj. málo účinné) instrukce, což si vynucuje značný objem kódování i pro poměrně jednoduchý program - to je zdrojem chyb a nesnází při dokumentaci.

U mikropočítačů, podobně jako u větších počítačů, je snahou zjednodušit programování. Pro větší počítače bylo vypracováno několik set vyšších programovacích jazyků, např. FORTRAN, COBOL, které usnadňují programování určité třídy úloh. Pro mikropočítače byl vypracován např. jazyk PL/M firmou Intel. Tento jazyk je určitou zjednodušenou variantou jazyka PL/1. Uvádí se, že pomocí jazyka PL/M se zkrátí doba programování asi na 10% vzhledem k programování v jazyku symbolických adres. Kromě toho je jednodušší zaškolení programátora, jednodušší dokumentace a napsaný program je přehlednější.

Vyšší programovací jazyky nejsou však u mikropočítačů při hromadných aplikacích mnoho rozšířené, neboť účinnost překladu vzhledem k využití kapacity paměti a rychlosti výpočtu se nejeví mnohdy dostatečná. Také překlad programu z jazyka PL/M se často provádí pomocí většího počítače. Lze však očekávat, že zdokonalováním technického vybavení mikropočítačů se bude rozšiřovat jejich programování ve vyšších programovacích jazycích.

Vyšší programovací jazyky (někdy je též nazýváme vyšší problémově orientované jazyky) používají příkazy (statements), které jsou obvykle zkratkami operací v anglickém jazyku. Typický aritmetický příkaz je

X = M . N + COS (P - R)

Tento jediný příkaz je ekvivalentní několika strojovým instrukcím včetně dvou skoků do podprogramů (jeden pro násobení a jeden pro cosinus).

Překlad zdrojového programu, v tomto případě programu ve vyšším programovacím jazyku, do strojového kódu (cílového programu) provádí kompilátor. V prvním přiblížení můžeme kompilátor považovat za velmi dokonalý asembler. Na obr. 2.3 jsou uvedeny metody generování strojového kódu, v němž je počítač teprve schopen řešit požadovanou úlohu. Čárkovanými obdélníky jsou vyznačeny programy, které jsou vázány na určitý typ počítače (strojový kód, asembler, program v jazyku symbolických adres, kompilátor). Vyšší programovací jazyk je strojově nezávislý, můžeme jej použít pro libovolný počítač,pokud je vybaven příslušným kompilátorem. V praxi některé kompilátory negenerují přímo strojový kód, ale program v jazyku symbolických adres (nebo program na podobné úrovni), který je nutno přeložit asemblerem do strojového kódu.

Porovnáváme-li kompilátor a asembler, můžeme konstatovat, že:

1) Kompilátor je odpovědný za uložení jak instrukcí tak dat na vhodné adresy operační paměti.
2) Jedné instrukci v jazyku symbolických adres odpovídá jedna strojová instrukce; jednomu příkazu ve vyšším programovacím jazyku odpovídá celý sled strojových instrukcí.
3) Menší počet příkazů při použití vyššího programovacího jazyka vzhledem k počtu instrukcí programu v jazyku symbolických adres znamená, že se program snadněji sestavuje, vzniká méně chyb, snadněji se ladí a dokumentuje.
4) Pro stejnou úlohu generuje obvykle kompilátor více strojových instrukcí než asembler; v důsledku toho je třeba větší kapacita paměti a také řešení úlohy trvá déle.

Vyšší programovací jazyky jsou obvykle problémově orientovány, odtud též název vyšší problémově orientované jazyky. Uveďme si základní vlastnosti nejznámějších vyšších programovacích jazyků.

FORTRAN (zkratka z formula translation) a ALGOL (zkratka z algoritmic language) se používají pro vědecko-technické (aritmetické) výpočty. Pracují v „nespřaženém režimu" tj. program včetně dat je nejprve zapsán na vhodné medium (např. děrnou pásku) a pak je s maximální možnou rychlostí zpracován na počítači.

Vědecko-technické výpočty vyžadují obvykle značnou přesnost výpočtu. FORTRAN zpracovává data především v pohyblivé řádové čárce. Pro jednoduchý číslicový systém to může být zbytečné. Taktéž oba tyto jazyky jsou zpravidla optimalizovány pro vstup pomocí děrných štítků. Dále jsou především určeny pro výstup ve formě tisku.

BASIC (zkratka z beginners all purpose symbolic instruction code) se též především používá pro vědecko-technické výpočty. Pracuje interaktivně, v průběhu výpočtu může počítač vyžadovat další data. Též je prováděn tisk výsledků ihned po jejich výpočtu. Proto je pro tento jazyk obvykle rychlost vstupu a výstupu limitujícím faktorem pro rychlost výpočtu.

COBOL (zkratka z common business oriented language) a RPG se používají pro hromadné zpracování dat. V těchto aplikacích se používají poměrně jednoduché aritmetické operace.

Jazyk PL/1 (zkratka z programming language one) a APL (zkratka z a programming language) byly vyvinuty jako univerzální jazyky určené jak pro vědecko-technické výpočty tak pro hromadné zpracování dat. Se stejným cílem byly rozšířeny jazyky FORTRAN a BASIC o další příkazy určené pro hromadné zpracování dat; nazýváme je rozšířený FORTRAN (Extended-FORTRAN) a BASIC - PLUS.

Žádný z dosud uvedených vyšších programovacích jazyků není vyhovující pro řešení úloh v reálném čase, tj. kdy události jsou vázány na přesně definovaný čas. Pro tyto úlohy je zvláště náročné zpracování vstupů a výstupů pomocí přerušení a priorit. Pro úlohy v reálném čase byly upraveny jazyky FORTRAN a BASIC rozšířením o další příkazy např. čekej 2 sekundy a pak čti výstup a analogově číslicového převodníku. Mnohé programy pro úkoly v reálném čase jsou psány v jazyku symbolických adres, přičemž někdy pro podprogramy je používán FORTRAN.

Pro programování úloh v reálném čase byly též vyvinuty z ALGOLU jazyky CORAL a RTL/2. Umožňují účinně řešit aritmetické a logické operace i operace pro vstupy a výstupy.

2.9 Kompilátor

Kompilátor patří do kategorie pomocných programů. Jak jsme si již uvedli, kompilátor se používá pro překlad programu napsaného ve vyšším programovacím jazyku. Jinými slovy kompilátor přijímá jednotlivé příkazy zdrojového programu a generuje z nich strojový kód, tj. cílový program. Tento cílový program je opět zaveden do počítače a proveden. Při překladu zdrojového programu je v operační paměti kompilátor, během této fáze se neprovádí žádný výpočet. Po překladu zdrojového programu, kdy cílový program se opět zavede do operační paměti počítače za účelem jeho provedení, není kompilátor již potřebný a proto může být z paměti vymazán a tato paměťová místa použita.

Je celá řada různě dokonalých kompilátorů zvláště s ohledem na generovaný strojový kód. Nejjednodušší kompilátory vyvolávají sled podprogramů. Tyto podprogramy provádějí násobení, dělení, trigonometrické funkce, zpracovávají příkazy pro vstup a výstup, zajišťují komunikaci s přídavnými paměťmi atd. Knihovna těchto podprogramů, nebo alespoň podprogramy nezbytné pro řešení dané úlohy, které jsou napsány ve strojovém kódu, musí být zavedeny do počítače současně s cílovým programem, aby tento mohl být proveden.

Dokonalejší kompilátor vkládá přímo příslušný kód při překladu, což je obvykle účinnější než vyvolávat požadované podprogramy, zvláště v případě, kdy může počítač provádět některé operace pomocí technického vybavení, např. násobení.

Poznamenejme, že dokonalejší kompilátor generuje kratší cílový program avšak za cenu delší doby překladu. Je-li program použit jen jednou, může být celkový strojový čas zahrnující překlad a výpočet však delší při dokonalejším kompilátoru. Pro aplikace, u kterých je opakovaně používán stejný program případně s různými daty, je účelný dokonalejší kompilátor umožňující kratší strojový čas.

Během překladu zdrojového programu do cílového programu kompilátor provádí detekci syntaktických[1] chyb a zajišťuje jejich

---

1) Syntaxe jsou pravidla pro vytváření přípustných kombinací symbolů, užívaná bez ohledu na význam symbolů a jejich kombinaci. Sémantika jsou pravidla přisuzující význam možným kombinacím symbolů.

výpis. Některé chyby lze však zjistit až při provádění cílového programu.

Jednodušší kompilátory nejsou rozsáhlé a proto je můžeme zapsat do operační paměti; přijímají zdrojový program z děrné pásky, magnetické kazetové paměti nebo magnetického pružného disku. Na stejná média se může zapisovat cílový program. Složité kompilátory pracují obvykle jen s magnetickou diskovou pamětí. Pro potřebnou knihovnu podprogramů je též mnohdy účelné používat magnetickou diskovou paměť.

Orientačně si uveďme, že jednoduchý kompilátor pro jazyk FORTRAN vyžaduje kapacitu alespoň 8 K slov a knihovna 4 K slov. Knihovna může být modulárně uspořádána a může být použita jen ta část, která je žádoucí pro daný program.

2.10 Jazyk FORTRAN

Jazyk FORTRAN se vyvíjel současně s počítači; v současné době jsou tři verze jazyka FORTRAN, které se liší počtem použitelných příkazů. Je dosud nejrozšířenějším jazykem u větších počítačů, používá se též u některých mikropočítačů. Rozeznáváme základní FORTRAN (Basic FORTRAN), střední FORTRAN (Intermediate FORTRAN) a úplný FORTRAN (FORTRAN IV). Jednotlivé úrovně jsou voleny tak, že nižší úroveň je vždy podmnožinou vyšší úrovně. Jazyk FORTRAN se dále rozpracovává, např. pro programování v reálném čase (R FORTRAN, Real Time FORTRAN).

Pro ilustraci jednoduchosti programování v jazyku FORTRAN si uveďme výpočet vztahu

$$X = \frac{A}{B} + C - (D.F) + G^5$$

Odpovídající zápis v jazyku FORTRAN je

```
X = A/B + C - D*F + G**5
```

Výpočet vztahu

$$Z = \frac{X^3}{Y + 2}$$

v jazyku FORTRAN zapíšeme

```
Z = X**3/(Y + 2)
```

Násobení zapisujeme jednou hvězdičkou, mocninu dvěma hvězdičkami.

Důležité je větvení programu (příkaz podmíněného skoku). V jazyku FORTRAN jej můžeme provést příkazem

IF (aritmetický výraz) n_1, n_2, n_3

kde n_1, n_2, n_3 jsou návěští cílových výkonných příkazů, na něž se provádí skok. Na návěští n_1 se provede skok, je-li hodnota aritmetického výrazu záporná, na n_2, je-li hodnota aritmetického výrazu rovna nule a na n_3, je-li hodnota aritmetického výrazu kladná. Je-li např. příkaz podmíněného skoku

```
IF (5*A + B) 15, 18, 20
```

nastává skok pro $5.A + B < 0$ na návěští 15, pro $5.A + B = 0$ nastává skok na návěští 18 a pro $5.A + B > 0$ nastává skok na návěští 20.

Cyklování (vytváření smyček) provádíme ve FORTRANU příkazem DO. Na příklad:

```
DO 20 I = 1, 30
```

Za proměnnou I, která se vyskytuje ve výrazech za příkazem DO se nejprve dosadí 1 a sled příkazů proběhne až do příkazu s návěštím 20. Pak se výpočet vrátí na první příkaz za „DO", I se zvětší o jedničku a postup se opakuje, pokud je I menší než 30; po jeho dosažení výpočet pokračuje příkazy, které následují za návěštím 20.

Proveďme součet 30 čísel uložených v paměťových buňkách A(1), A(2), až A(30).

```
S = 0
DO 20 I = 1, 30
20 S = S + A(I)
```

První příkaz vynuluje paměťovou buňku S. Následuje postupné přičítání do paměťové buňky S. Po prvním cyklu bude obsah paměťové buňky S = A(1), po druhém cyklu S = A(1) + A(2), atd. Po proběhnutí 30 cyklů bude buňka S obsahovat požadovaný součet a počítač bude konat další příkazy následující za návěštím 20.

2.11 Jazyk PL/M

První vyšší programovací jazyk určený pro mikropočítače byl PL/M. Vyvinula jej firma Intel pro svůj mikroprocesor 8008. Používá se však i pro mikroprocesor 8080 i 8085. Jazyk PL/M se postupně stává standardním programovacím jazykem i pro mikropočítače jiných firem.

Kompilátor je často nepřímý, napsaný v jazyku FORTRAN, pro překlad se používá hostující počítač. Překlad programu v jazyku PL/M do strojového kódu se obvykle provádí ve dvou krocích. Nejprve kompilátor provede překlad do jazyka symbolických adres, načež následuje překlad pomocí asembleru do strojového kódu. Tento překlad kompilátorem do jazyka symbolických adres místo do strojového kódu má určité výhody; umožňuje programátorovi použít výpis pro zkoušení a ověření programu. Dále, je-li překlad málo účinný, může programátor prostudovat tento výpis a vyloučit nadbytečné přesuny dat, t.j. může v jazyku symbolických adres zkrátit program.

Nedostatkem vyššího programovacího jazyka mohou být, kromě dalšího cílového programu, i obtíže s některými typickými mikropočítačovými operacemi, jako je např. zpracování vstupů a výstupů, zpracování časově kritických sledů, rovněž lze nesnadno zkoumat stavy jednotlivých bitů.

Pro porovnání programování v jazyku PL/M a v jazyku symbolických adres hledejme větší z čísel A, B, které si označme C. V jazyku PL/M výpočet provedeme příkazem

```
IF A > B, THEN C = A;  ELSE C = B
```

což značí: je-li A > B, pak C = A, jinak C = B.
Odpovídající program v jazyku symbolických adres je v tab. 2.2.

Jako další příklad programování v jazyku PL/M si uveďme následující sled příkazů:

```
DECLARE (A, B, R) BYTE
IF A > B  THEN  R = A - B + 2
ELSE R = B - A + 2
```

A, B a R se vztahují k obsahům jednobytových paměťových buněk určených příkazem DECLARE. Tento sled příkazů přeloží kompilátor do programu v jazyku symbolických adres mikropočítače Intel 8080.

Cílový program má téměř 20 instrukcí (tab.2.3). Z porovnání je zřejmé, že i přes komentář k programu v jazyku symbolických adres je tento obtížněji pochopitelný než krátký program v jazyku PL/M. Odtud je též zřejmá menší pravděpodobnost vzniku chyby při sestavě programu v jazyku PL/M.

V jazyku PLM zaujímají návěští samostatný sloupec (jsou oddělena „sloupcovou mezerou". Komentář je vždy oddělen na začátku a na konci šikmou úsečkou a hvězdičkou, např. /* POZNÁMKA */. In-Identifikátory[1] mohou být sestaveny z libovolného počtu abecedně číslicových znaků, musí však začínat abecedním znakem a nesmí obsahovat mezeru; pro lepší čitelnost místo mezery může být použit znak $. Jako identifikátor nesmí být použita rezervovaná slova, např. OR, PLUS, IF atd.

Problém adresování se redukuje na definování proměnných a jejich deklaraci[2]. Mikroprocesory nemají obvykle instrukci pro dělení; přesto pro tuto operaci stačí jednoduchý příkaz. Kompilátor PL/M automaticky vloží podprogram pro dělení do cílového programu.

Identifikátor proměnné musí být před použitím nejprve definován pomocí deklarace. Deklarací se např. stanoví, zda proměnná má být považována jako bytová proměnná (8 bitů) nebo adresová proměnná (16 bitů).

Data v deklaracích a příkazech mohou být vyjádřena způsoby uvedenými v tab. 2.4. Všimněme si, že v uvedeném příkladu jsou stejná dvojková data vyjádřena různým způsobem. Příklady deklarací jsou uvedeny v tab. 2.5. Pro jednoduchost jsou identifikátory označeny A, B, C, D a E. Jak jsme si již uvedli, mohou být složeny z více abecedně číslicových znaků. V příkazu se často používají operátory[3] uvedené v tab. 2.6. Řádky jednotlivých operátorů jsou uvedeny ve sledu, podle kterého je jazyk PL/M bere v úvahu. Operá-

---

1) Identifikátor je symbol identifikující či pojmenovávající data v programovacím jazyku.

2) Deklarace = podpis (declaration) je příkaz určující typ, rozsah popřípadě i hodnotu údajů označených identifikátory.

3) Operátor je symbol označující operaci.

tory uvedené v jednom řádku mají stejnou prioritu a jsou vyčíslovány zleva doprava. V důsledku toho je někdy třeba použít závorek. Např.:

```
STREDNIØHODNOTA = (VZOREKL + VZOREK2)/2
```

Vypustíme-li v tomto případě závorku, je k hodnotě VZOREK2/2 přičtena hodnota VZOREK1.

Příklady příkazů jsou uvedeny v tab. 2.7. Každý z těchto příkazů odpovídá několika instrukcím v jazyku symbolických adres. Jelikož můžeme použít výrazy[1], které vyjadřují např. index[2], adresu, podmínky větvení, resp. počet oběhů smyčkou, může být často kombinováno v jednom příkazu více kroků.

Důležitým programovacím prostředkem v jazyku PL/M jsou procedury[3] (podprogramy), které mají analogickou funkci jako makroinstrukce v jazyku symbolických adres. Jako příklad uvažujme proceduru

```
A: PROCEDURE (B, C) ADDRESS; /* definice procedury se symbo-
                                lickým označením A */
   DEFINE (B, C, D) ADDRESS
   :
   RETURN D
   END
```

Příkazy mezi DEFINE a RETURN určují výsledek D jako funkce pomocných proměnných B a C.

Některé procedury jsou přímo součástí jazyka PL/M; tab. 2.8. Jména těchto procedur lze použít ve výrazech jako by to byly identifikátory. Příklady jsou uvedeny v tab. 2.9. Jiný typ procedury, která neposkytuje data programu a proto ji pouze vyvoláváme, místo abychom ji použili ve výrazech, je procedura pro zpoždění.

1) Výraz (expresion) je přípustný řetězec symbolů programovacího jazyka, jemuž lze podle sémantických pravidel tohoto jazyka přiřadit hodnotu.

2) Index = přípona (subscript) je označení polohy (adresy) v jednorozměrném nebo vícerozměrném poli.

3) Procedura (procedure) je uzavřená část programu provádějící samostatnou část algoritmu.

Např. značí-li

CALL TIME(A) vyvolání zpoždění A.0,1 ms,

pak

CALL TIME(35) vyvolá zpoždění 3,5 ms.

Návěští adres mohou být připojena ke každému příkazu v programu. Např.

A : B = C + 1	přiřazuje příkazu návěští A
274 : A = B + 1	přiřazuje příkazu adresu 274
B = .A + 27	.A značí „adresa návěští A" (adresa může být použita jako část výrazu)
EOF	označuje konec programu v jazyku PL/M; toto označení je třeba vždy použít.

2.12 Interpretační jazyky, BASIC

Interpretační jazyky jsou aplikačně orientované jazyky. Překladač interpretačního jazyka překládá zdrojový program tím způsobem, že nejprve přeloží jeden příkaz zdrojového jazyka do jedné nebo více strojových instrukcí, které počítač provede. Pak se přeloží a provede následující příkaz zdrojového programu; tento proces pokračuje až do vyčerpání všech příkazů zdrojového programu.

Mají-li být napsané programy vícekrát využity, jsou interpretační jazyky málo účinné, neboť každé použití programu je prodlužováno překladem. Interpretační jazyky se však používají pro konverzační způsob řešení úlohy na počítači; též jsou používány u některých mikropočítačů.

Typickým představitelem interpretačního jazyka je BASIC, který je ve zdrojovém programu podobný jazyku FORTRAN. Jejich provedení se však podstatně liší. V BASICu se příkaz dekóduje a provede; poté následuje dekódování a provedení dalšího příkazu.

Překladač, který nazýváme častěji interpreter než kompilátor, musí být stále v operační paměti, což je nevýhoda, neboť tím trvale obsazuje část paměti. Další nevýhodou interpretačního jazyka je nezbytnost opakovaného překladu každého příkazu při vytvoření programové smyčky, což značně zpomaluje výpočet. Podstatnou výhodou jazyka BASIC je jeho jednoduchost z hlediska programování.

Jako příklad programování v jazyku BASIC proveďme výpočet kořenů kvadratické rovnice

$$\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

Program pro A = 2, B = 6 a C = 4 je uveden v tab. 2.10.

2.13 Účelnost použití vyšších programovacích jazyků

Proti jiným počítačům bývá mikropočítač dosud zřídka programován uživatelem. Jakmile však bude běžné jej programovat ve vyšších programovacích jazycích (podobných jazyku FORTRAN nebo BASIC), bude uživatel více schopný přizpůsobit mikropočítačový systém svým požadavkům a užívání mikropočítačů ještě více stoupne.

Skutečností je, že dosud konstruktéři nepoužívají často mikroprocesoru resp. mikropočítače pro řešení vysoce flexibilních systémů v mnoha různých aplikacích. Ve většině případů dosud používají mikropočítače pro zjednodušení svých úloh při řešení komplexních logických systémů, které se vyrábějí ve větších výrobních sériích. Pro tyto účely je sestavování programů v jazyku symbolických adres vhodné, neboť obtížnější a značně nákladné programování je při větších výrobních sériích vyváženo snížením nákladů na paměť (menší potřebná kapacita). Při menších výrobních sériích se programování v jazyku symbolických adres může stát méně ekonomicky výhodné.

Pro menší výrobní série je obvykle výhodnější programování ve vyšším programovacím jazyku, které je snadnější a rychlejší a tudíž lacinější i při nezbytnosti použití paměti s větší kapacitou.

V současné době nejsou pro mnohé mikropočítače dostupné kompilátory, které by přeložily programy sestavené ve vyšších programovacích jazycích do programů ve strojovém kódu. Všichni výrobci mikropočítačů poskytují buď nepřímý nebo přímý asembler, případně oba typy.

Dosud jen poměrně málo výrobců mikropočítačů dodává nepřímý nebo přímý kompilátor[1] PL/M, jen někteří výrobci poskytují kompi-

---

1) Pojem nepřímý kompilátor je analogický nepřímému asembleru; pro jeho využití je třeba použít větší počítač.

látor FORTRAN nebo interpreter BASIC. Je však snahou většiny výrobců vybavit své mikroprocesory vyššími programovacími jazyky. Za této situace konstruktéři mikropočítačových systémů jsou nebo budou nuceni se rozhodnout, zda použijí jazyk symbolických adres nebo vyšší programovací jazyk. Kritériem pro toto rozhodnutí je čas potřebný pro vývoj systému včetně sestavení programu a náklady spojené s vývojem a výrobou.

2.13.1 Výhody a nevýhody použití vyššího programovacího jazyka

První argument, který preferuje použití vyššího programovacího jazyka proti jazyku symbolických adres je snížení nákladů na programování a urychlení vývoje. Pro základní orientaci můžeme předpokládat, že programátor sestaví a odladí průměrně 10 příkazů či instrukcí denně bez ohledu na jazyk, ve kterém programuje. Jelikož jeden příkaz ve vyšším jazyku odpovídá několika strojovým instrukcím, kdežto jeden příkaz (instrukce) v jazyku symbolických adres odpovídá jedné strojové instrukci, programátor „vykoná více práce" při programování ve vyšším programovacím jazyku. Tudíž je toto programování rychlejší a méně nákladné.

Další argument pro využití vyššího programovacího jazyka je snadnější čtení již napsaného programu a tím i snadnější ladění. Též inovace programu je snadnější.

Současné programování mikropočítačů v jazyku symbolických adres je často zdůvodňováno větší rychlostí mikropočítače a menší potřebnou kapacitou paměti. Skutečností je, že pro stejnou úlohu překladem programu napsaného ve vyšším programovacím jazyku kompilátorem se generuje více strojových instrukcí než při překladu programu napsaného v jazyku symbolických adres asemblerem. Proto úlohu napsanou ve vyšším programovacím jazyku řeší počítač pomaleji a potřebuje větší kapacitu paměti. V mnohých aplikacích rychlost mikropočítače však není omezujícím faktorem. Program přeložený kompilátorem vyžaduje zvětšení kapacity paměti o 10% až 100% proti programu, který je přeložen asemblerem.

Výhodnost programování v jazyku symbolických adres nebo problémově orientovaném jazyku je ve většině případů závislá na vyráběném množství zařízení se stejným programem. Uveďme si dva extrémní případy. Jako první uvažujme mikropočítač, který je součástí vah

na potraviny. Pro tuto aplikaci sestavíme jeden program, který zůstává nezměněn během výroby vah, která může dosahovat několika tisíc kusů. V tomto případě je s největší pravděpodobností výhodné programovat v jazyku symbolických adres, neboť vyšší náklady na programování jsou zdaleka převáženy nižší cenou pamětí v uvažované výrobní sérii.

Za druhý extrém můžeme považovat použití mikropočítače v automatickém zkušebním zařízení, u kterého je třeba často měnit program pro různé zkušební podmínky. V tomto případě je nepravděpodobné, že cena dílčích bloků případně celého zařízení bude vyšší, než náklady spojené se sestavou, změnami a udržováním programů. K jednomu dodávanému zařízení je třeba více rozdílných programů. Snížení nákladů na programování použitím vyššího programovacího jazyka zdaleka převyšuje náklady spojené s nezbytností vyšší kapacity paměti.

Zbývá alespoň orientačně stanovit hranici mezi „malým počtem" vyráběných zařízení, kdy je výhodné použít pro programování vyšší programovací jazyk a „velkým počtem" vyráběných zařízení, kdy je výhodné použít programování v jazyku symbolických adres. Hranici lze stanovit následovně: náklady na programování jsou L . P; kde L je počet příkazů v programu, P jsou náklady na jeden příkaz. (V prvním přiblížení můžeme uvažovat výkon programátora 10 příkazů za den). Cena paměti je L . B . M; B je počet bitů v jednom příkazu, M je cena jednoho bitu.

Cena paměti v N zařízeních a náklady na programování jsou N . L . B . M + L . P. Označíme-li si indexem s případ programování v jazyku symbolických adres a indexem v programování ve vyšším programovacím jazyku, pak hranice vyráběného množství určující výhodnost použití jednoho či druhého typu programovacího jazyka je určena vztahem

$$N \cdot L_s \cdot B_s \cdot M + L_s \cdot P = N \cdot L_v \cdot B_v \cdot M + L_v \cdot P$$

Nechť celý program (strojový program) získaný kompilátorem má E_v násobek bitů než je počet bitů, z kterého sestává strojový program předložený asemblerem t.j.

$$L_s \cdot B_s \cdot E_v = L_v \cdot B_v$$

Např. generuje-li kompilátor strojový jazyk o 25% rozsáhlejší než generuje asembler, pak $E_v = 1,25$. Z předcházejících vztahů můžeme určit

$$N = \frac{P\ [(1/B_s) - E_v/B_v]}{M\ (E_v - 1)}$$

Parametr M (cena jednoho bitu) především značně závisí na typu použité paměti (reprogramovatelná, programovatelná, programovaná maskou); může se měnit v poměru 1 : 50 i více. Též je třeba si uvědomit, že kapacitu paměti nelze měnit plynule, ale „po krocích", odpovídající kapacitě jednoho integrovaného obvodu (obvykle 1 K až 16 K).

Počet bitů příslušející jednomu příkazu v jazyku symbolických adres (B_s) bývá přibližně 16, počet bitů příslušející jednomu příkazu ve vyšším programovacím jazyku (B_v) můžeme volit přibližně 80. Parametr E_v bývá v rozmezí 1,1 až 2, někdy i vyšší.

Z předcházejícího je zřejmé, že nejvíce proměnný parametr je M. Výhodnost použití jednoho nebo druhého programovacího jazyka závisí na konkretním případu. Pro první přiblížení můžeme však za hranici výhodnosti použití jednoho či druhého jazyka považovat výrobní sérii stovky kusů.

2.14 Programy pro úpravy uživatelských programů a jejich ladění

2.14.1 Modifikace uživatelských programů

Po sestavení zdrojového programu programátorem se tento program přeloží buď pomocí asembleru nebo kompilátoru do cílového programu. Následuje zavedení tohoto cílového programu do počítače, načež se tento program provede. Během vývoje je velmi často třeba program upravit, což obvykle programátor zjistí až po jeho zavedení. V tomto případě zdrojový program může být změněn, znovu přeložen, zaveden a proveden. Změnit program zapsaný na papíře je poměrně snadné. Je-li program vyděrován na děrných štítcích, což obvyklý způsob při vědecko-technických výpočtech na větším počítači, mohou být vyděrovány nové štítky a tyto vloženy na potřebné místo. U mikropočítačů (a většinou též u minipočítačů) se však obvykle používá jako vstupní medium děrná páska. Je sice možné vyděrovat nový blok na děrnou pásku a tuto vložit do původní děrné pásky,

avšak tento postup je náročný a zdlouhavý.

Pomocný program nazývaný úpravný program (editor, text-editor) umožňuje snadněji změnit zdrojový program. Tento úpravný program přečte zdrojový program z děrné pásky a uloží jej do operační paměti. Následuje úprava zdrojového programu pomocí příkazů, které zadáváme dálnopisem, nebo klávesnicí zobrazovací jednotky. Je-li mikropočítač vybaven magnetickou kazetovou pamětí nebo pamětí s pružným diskem (případně magnetickou diskovou pamětí nebo magnetickou páskovou pamětí), přenese zpravidla zdrojový program z děrné pásky do této přídavné paměti, načež následuje úprava zdrojového programu. Nedostatkem úpravy zdrojového programu pomocí úpravného programu je nezbytnost opakovaného překladu, zavedení a provedení cílového programu po každé změně.

Některé úpravy mohou být tak jednoduché, že stačí změnit jen několik slov v cílovém programu. Tuto změnu můžeme provést dalším pomocným programem, který nazýváme ladicí program (debugger).

2.14.2 Úpravný program

Úpravný program obvykle umožňuje:

1) Modifikaci programu- zvolená část programu může být zrušena nebo nahrazena novým blokem (částí) programu, resp. do zvoleného bodu programu může být vložen nový blok programu;
2) výpis programu- na kontrolu správnosti může být určitá zvolená část programu vytištěna tiskárnou;
3) řízení vstupů a výstupů- zdrojový program se převede z děrné pásky do operační paměti, upravený zdrojový program se z operační paměti přenese na děrnou pásku nebo do jiné přídavné paměti.

Někdy jsou asembler a úpravný program sloučeny do jediného programu; nazýváme jej konverzační asembler.

Někteří výrobci mikroprocesorů poskytují úpravný program jako část standardního programového vybavení jejich mikropočítačového vývojového systému. Úpravný program bývá zaváděn do mikropočítače pomocí děrné pásky; vyžaduje však poměrně velkou kapacitu paměti. To někdy vyžaduje rozdělit program na části a provádět úpravu zdrojového programu postupně; to je však zdlouhavé.

Pokud je používán nepřímý asembler, je též úpravný program používán na hostujícím počítači.

2.14.3 Ladicí program

Zatímco úpravný program umožňuje upravit zdrojový program, ladicí program (debugger) umožňuje upravit cílový program. Je to v podstatě diagnostický program, který umožňuje analyzovat program během jeho provádění. Obvykle umožňuje:

1)tisk obsahu paměťových buněk na zvolených adresách a tisk obsahů některých registrů počítače, (tisk může být prováděn v osmičkové, šestnáctkové nebo desítkové soustavě);
2)vložení nové informace (i jednotlivých bitů) na zvolené místo programu;
3)zastavení programu v libovolném místě, v kterém chceme zkoumat obsahy zvolených registrů nebo paměťových buněk.

Ladicí program se především používá pro diagnostiku cílového programu a též pro jeho malé změny; větší změny programu se provádí pomocí úpravného programu ve zdrojovém programu.

2.14.4 Simulační programy

Simulační programy, někdy nazývané simulátory, jsou často velmi účinným prostředkem při ladění a zkoušení uživatelských programů určených pro mikropočítač, zvláště v těch případech, kdy sestava mikropočítače není ještě dostupná. Simulační programy jsou obvykle psány v jazyku FORTRAN a provozovány na hostujícím počítači. Tyto programy simulují specifický mikroprocesor, tj. interpretují instrukce mikropočítače a simulují jejich provádění. Jsou schopny zobrazit každý detail prováděného programu.

Obecně čím je dokonalejší (přesnější) simulace, tím je nákladnější simulační program a tím déle je prováděn. Simulace vstupů a výstupů je zvláště obtížná, avšak velmi žádoucí. Např. má-li účinně pracovat snímač děrné pásky, vyžaduje přenos dat během přesně vymezených dob. Má-li být simulace vstupů a výstupů přesná, program musí sledovat simulovaný reálný čas, což může být nesnadné a nákladné.

Simulátor přijímá program ve strojovém kódu mikropočítače a podle požadavku sleduje jeho provádění. Příklad použití simulá-

toru je v tab. 2.11. Výstup simulátoru vyjadřuje obsahy různých registrů (včetně registrů příznaků) a pamětových míst. Simulační příkazy umožňují programátorovi obdržet zvolený výstup (obsah paměti a registrů) v požadovaném bodě programu. Seznam možných simulačních příkazů je v tab. 2.12. Mnohé příkazy uvedené v této tabulce jsou podobné těm, které umožňuje ladicí program. Odtud je zřejmé, že programová simulace může být použita jako alternativa k použití ladicího programu.

Ačkoliv přesná simulace některých operací může být nesnadná, kontrola podprogramů (např. aritmetických) nebo kontrola přesnosti výpočtu či vhodnost zvoleného číselného rozsahu mikropočítačového systému může být velmi dobře řešena simulací.

2.15 Operační systém

Operační systém můžeme charakterizovat jako soubor programů, který koordinuje činnost uživatelských programů a zajišťuje jejich bezporuchový průběh. Z jistého hlediska můžeme na uživatelské programy pohlížet jako na podprogramy operačního systému. Operační systém může např. ovládat vstupy a výstupy. Je řešen tak, aby koordinoval a optimalizoval funkci počítače při různé zátěži. Může řešit otázky spojené s přerušením programů, poruchou (chybou) nebo s mimořádnými podmínkami.

Operační systém je řízen hlavním programem tohoto souboru, který nazýváme řídicí program nebo supervizor, dohlížecí program, exekutivní program (executive, monitor, supervisor). K tomuto programu se počítač vždy vrací po dokončení jiných programů, ať již spadají do kategorie operačního systému nebo uživatelských programů. Řídicí program určuje, jakou činnost bude počítač vykonávat, tj. který program bude řešit a současně předá řízení tomuto programu. V operačním programu mohou být sloučeny všechny dosud uvažované pomocné programy (zaváděcí, úpravný, ladicí a překladače) i řídicí program.

Obecným cílem operačního systému je:

1) Maximálně ulehčit práci programátorovi v jednoduchých a opakujících se úkonech (např. vyhledávání zdrojového programu, zavedení kompilátoru, provedení překladu zdrojového programu, zavedení cílového programu a jeho provedení) tím, že příkazy pro

tyto jednotlivé úkony se zadávají pouze klávesnicí.

2) Maximálně zvýšit propustnost a pružnost počítače. Čas, po který je aktivně využíván procesor je relativně krátký (často pod 10%). Toto nízké využívání procesoru je způsobeno zpravidla rozdílnou rychlostí procesoru a přídavných zařízení.

Mnohé dílčí úlohy, které by měly být zapsány do uživatelského programu se stávají standardní záležitostí operačního systému, zvláště pro řízení vstupů a výstupů, vzájemného propojení jednotlivých programů atd. Dále se mohou sledováním programů detekovat chyby nebo nesprávné využívání jednotlivých bloků, čímž můžeme předejít případnému výpadku celého systému. Operační systém může též provádět periodickou diagnostiku systému včetně výpisu zprávy.

Obecné použití operačního systému způsobuje lepší využívání jednotlivých bloků systému za cenu větší potřebné kapacity jak operační paměti tak přídavné velkokapacitní paměti. Použití operačního systému může někdy i vyvolat částečné zmenšení propustnosti počítače vzhledem k speciálně psaným programům pro určitou úlohu (např. pro vstup - výstup), avšak zpravidla zajišťuje lepší využití systému jako celku.

Není pochyb o tom, že cena technického vybavení bude stále klesat, jeho rychlost se zvětšovat, ale naproti tomu cena programového vybavení bude stoupat. Tyto skutečnosti budou vyúsťovat ve větší využívání operačních systémů a ve zvyšování jejich dokonalosti.

Operačních systémů je celá řada, obvykle se vážou k určité konfiguraci počítače. Dokonalý operační systém by byl značně rozsáhlý a nákladný. Proto existuje celá řada operačních systémů, z nichž každý je určen pro určitou třídu aplikací. Mohou být rozděleny do dvou kategorií: operační systémy pro činnost v nespřaženém režimu a operační systémy pro činnost ve spřaženém režimu.

Pro jednoduché sestavy počítačů, které nemají přídavnou velkokapacitní paměť - tj. mikropočítače s dálnopisem, abecedně číslicovou zobrazovací jednotku případně s magnetickou kazetovou pamětí, je operační systém obvykle velmi jednoduchý, zpravidla provádí jen zavádění programů a jejich vazbu. Supervizor je zpravidla trvale v operační paměti. Vzhledem ke klesající ceně paměti a časté-

mu používání této malé sestavy je věnována těmto typům supervizoru stoupající pozornost.

Supervizor může být pro velký mikropočítačový systém značně rozsáhlý. Obvykle je rozdělen do modulů, z nichž pouze jeden je trvale v operační paměti. Ostatní jsou v přídavné velkokapacitní paměti a jsou přesouvány do operační paměti jen v případě potřeby.

2.15.1 Nespřažený operační systém, diskový operační systém a multiprogramování

Je-li mikropočítač vybaven jen dálnopisem, je zavádění jednotlivých pomocných programů i uživatelského programu zdlouhavé. Tuto činnost můžeme podstatně zjednodušit použitím magnetického pružného disku. Další výhodou použití magnetického pružného disku je možnost zpracování rozsáhlejších uživatelských programů, pokud je rozdělíme do částí. Systém, který umožňuje tuto činnost nazýváme diskový operační systém (DOS, Disk Operating System).

Značně rozsáhlejší supervizor bývá u větších výpočetních systémů určených pro zpracování úloh po dávkách (batch processing). Supervizor řadí do fronty jednotlivé úlohy, zapisuje je do seznamu a zajišťuje jejich provedení. Nová úloha může být kdykoliv přidána a je nezbytné ji zařadit na vhodné místo fronty. Úlohy jsou různé povahy; některé jsou pravidelně žádány v určitou denní dobu, jiné jsou vyžadovány s různou prioritou. Každá úloha má tedy přiřazenou určitou prioritu. Po dokončení jedné úlohy supervizor zajistí provedení další úlohy s největší prioritou ve frontě, což nemusí být úloha, která čeká nejdelší dobu ve frontě. Uvedený operační systém, který řeší postupně jednotlivé úlohy, je málo účinný vzhledem k využívání technického vybavení.

Multiprogramování „prokládá" úlohy náročné na čas centrální jednotky s úlohami náročnými na vstupně-výstupní operace, značně zvyšuje propustnost počítače. Uvažujme úlohu, při které se mají číst data z děrné pásky a tisknout výsledky na řádkové tiskárně. Lze předpokládat, že centrální jednotka počítače bude po většinu doby čekat na data, případně tisk. Jinými slovy - čtení a tisk budou úzkým profilem systému. Provedeme-li přepis obsahu děrné pásky přímo do magnetické diskové paměti (zatímco centrální jednotka provádí jinou práci), budou data k dispozici s přenosovou

rychlostí „operační paměť - magnetická disková paměť". Podobně mohou být výsledky před uskutečněním tisku nejprve přeneseny do magnetické diskové paměti; během tisku může centrální jednotka řešit jiné úlohy. Cílem multiprogramování je optimálně vytížit jednotlivé bloky počítače. Supervizor musí být schopen řídit několik programů.

Multiprogramování může být ještě zdokonaleno tím, že v operační paměti je několik programů, což je časově výhodnější než provádět časté přesuny informací mezi operační pamětí a magnetickou diskovou pamětí.

Multiprogramování zřejmě vyžaduje dokonalý systém ukládání informací. Místo dočasně přerušeného programu musí být dokonale zaznamenáno. Též nesmí docházet k porušení jednoho záznamu v paměti druhým záznamem. Toto může být řešeno systémem ochrany paměti, který zamezí zápisu jednoho uživatelského programu do jiného uživatelského programu. Tato ochrana však musí být dynamická, protože zavedené programy se neustále mění (až na několik programů, např. rezidentní část supervizoru).

Ochrana paměti se může provádět technickým vybavením, např. ochranným bitem v paměti, který se přidává ke každé paměťové buňce. Stav tohoto ochranného bitu je určován supervizorem. Jiné řešení pomocí technického vybavení je použití registrů, které označují obě krajní adresy aktivního aplikačního programu. Při použití programového vybavení pro ochranu paměti je možné kontrolovat (vyhodnocovat) každou adresu. Řešení ochrany přidáním dalšího ochranného bitu ke každé adrese paměti je nákladnější než řešení programovým vybavením, je však rychlejší.

2.15.2 Spřažený operační systém, sdílení času

Při zpracování programů po dávkách jsou vstupní data uvedena v souboru již před započetím výpočtu. Během provádění programu (při výpočtu) jsou vstupní data ukládána do jiného souboru. Supervizor pomocí programu pro tisk zajistí tisk výsledků. Podobně je tomu u multiprogramování.

Spřažený operační systém (operační systém počítače, který pracuje ve spřaženém režimu.) umožňuje zpracování dat ihned po jejich

vstupu. Nejznámější způsob je sdílení času.

Při sdílení času využívá např. jeden program centrální jednotku počítače jen po určitou dobu, což může být 0,1 s. Po této době je výpočet přerušen, následuje zavedení a provádění dalšího programu po dobu 0,1 s. V uvažovaném případě se může řešit deset programů „současně". Často jsou jednotlivé programy zadávány koncovými zařízeními, které jsou více či méně vzdálené od počítače a komunikace se uskutečňuje po telefonním vedení. Jelikož rychlost těchto koncových zařízení je malá, jeví se uživateli, že počítač zpracovává jen jeho požadavky. V těchto případech se často používá interpretační jazyk BASIC.

2.15.3 Operační systém pro činnost v reálném čase

Operační systém pro činnost v reálném čase (RTOS, Real-Time Operating System) můžeme považovat za speciální typ spřaženého operačního systému.

U počítače pracujícího v reálném čase jsou známé vnější podněty tj. události, které je nutné řešit, ale nejsou předem známé okamžiky, ve kterých se tyto podněty vyskytnou; některé podněty se též mohou vyskytnout současně. Počítač musí dát na tyto vnější podněty odezvu v čase, který je podstatně kratší než děj, který podnět vyvolal. Vnější podnět může vyvolat např. člověk, který s počítačem komunikuje, nebo řízený průmyslový proces. Počítač pracující v reálném čase dále musí provádět v předem zvolených okamžicích určité funkce periodicky v přesně stanovené denní době. Tyto funkce jsou podmíněny systémem přerušení a systémem priorit.

V uvažovaném případě je počítač řízen operačním systémem určeným pro činnost v reálném čase. Řídí jednotlivé programy a vymezuje pro ně části operační paměti, řídí přerušení podle priorit, kontroluje správnou funkci jednotlivých částí počítače a v případě poruchy může hledat náhradní nouzové řešení atd.

Oproti uživatelským programům, které jsou psány pro jednotlivé konkretní úlohy, je snahou vytvořit obecné operační systémy, které by byly využitelné v široké třídě aplikací. Je-li operační systém značně rozsáhlý, mívá značné nároky na kapacitu paměti i počítačový čas a může v některých případech snižovat propustnost po-

čítače. Mikropočítače si z hlediska kapacity paměti ani nemohou dovolit rozsáhlý operační systém. Proto jsou pro určité aplikace řešeny samostatné „jednoúčelové" operační systémy. Vypracování operačního systému není však jednoduchou záležitostí.

Pro sestavení operačního systému je důležité, zda bude počítač využit pro práci v reálném čase, v jaké sestavě bude pracovat. Mnohé počítače jsou proto vybaveny výrobcem i pro práci v reálném čase několika operačními systémy, z nichž volíme ten, který je vhodný pro danou konfiguraci a nasazení.

Potřeba operačního systému počítače pro práci v reálném čase vyplývá hlavně ze dvou důvodů:

1) Efektivní využívání jednotlivých částí počítače, hlavně pak rychlé centrální jednotky a kapacity paměti.
2) Zajištění krátké časové odezvy.

První uvedený důvod se v podstatě kryje s obecnou potřebou operačního systému. Druhý důvod vyplývá ze skutečnosti, že při řízení procesu je krátká časová odezva mnohdy závažným požadavkem; počítač musí dostatečně rychle reagovat na události, které se vyskytují v řízeném procesu. U mnohých počítačů přerušení probíhajícího programu i start nového, vyvolaného programu provádí operační systém. Poznamenejme však, že zpravidla není možné sestavit obecný operační systém (resp. supervizor), který by měl tak krátkou časovou odezvu přerušení jako speciálně napsaný uživatelský program. Obecná koncepce supervizoru - detekovat požadavek na přerušení, zpracovat jej a předat řízení příslušnému uživatelskému programu - zavádí zpoždění.

Uveďme si některé specifické požadavky na operační systém, který je určen pro řízení průmyslového procesu:

1) Jednotky styku s prostředím

Jednou ze zvláštností počítače určeného pro řízení procesu nebo sběr dat je podsystém vstup - výstup, který navazuje na proces. Kromě klasických přídavných zařízení, které slouží pro vstup a výstup dat, ke komunikaci s operátorem a k rozšíření operační paměti, musí být počítač napojen pomocí číslicových a analogových čidel i ovladačů (ovládacích prvků) na řízený proces.

Tyto jednotky styku s prostředím (někdy též vstupní a výstupní zařízení), ať již analogové nebo číslicové, přenášejí pro sestavu operačního systému některé specifické problémy:

a) Jednotek styku s prostředím navazujících na proces bývá značný počet. Každý z analogových nebo číslicových podsystémů vstup - - výstup může navazovat až na několik set zařízení vstup - výstup a každé z těchto zařízení má svoji adresu.

b) Mnohé jednotky styku s prostředím mají dva stavy (koncový spínač sepnut nebo rozepnut, ventil zavřen nebo otevřen, relé sepnuto nebo rozepnuto). Pro vyjádření stavu takového zařízení stačí jen jeden bit ve slovu a každý bit slova dat může vyjadřovat stav jiného zařízení. V tomto případě musí programátor mít možnost zpracovat samostatně každý bit tohoto slova.

c) Jednotky styku s prostředím navazující na proces jsou mnohdy libovolně adresovány. Jednotky styku s prostředím (čidla, snímače) jsou představovány přístroji pracujícími na různých fyzikálních principech a jsou též různě lokalizovány v procesu. Proto adresy těchto jednotek často nenásledují číselně po sobě.

d) Informace poskytované jednotkami styku s prostředím jsou převážně jednoslovné a jejich změny nebývají časté. Příkladem může být přenos obsahu registru odpovídající analogové veličině snímané analogově číslicovým převodníkem. Přenos jediného slova při vědeckotechnických výpočtech nebo hromadném zpracování dat se vyskytuje ojediněle.

2) Časová odezva počítače

Jeden z hlavních problémů počítače při řízení průmyslového procesu je rychlost reakce na vnější podněty. Požadavek je obvykle v relaci s rychlostí zařízení generujícího podnět (resp. představujícího ovládaný prvek). Rychlá odezva počítače je obvykle vyžadována při spolupráci s elektronickým zařízením, podstatně pomalejší odezva stačí při spolupráci se zařízením mechanickým. Pro orientaci si uveďme, že rychlost reakce elektronického zařízení bývá 0,1 μs až 100 μs, elektromechanického 1 ms až 500 ms, mechanického 100 ms až 100 s a reakce člověka 0,1 až 100 s. Rychlost časové odezvy počítače je výsledkem technického vybavení počítače, zpravidla však odráží i řešení operačního systému.

3) Zatížení počítače

Počítač dimenzujeme z hlediska maximálního zatížení řízeným procesem. Průměrné zatížení je obvykle nízké. Někdy můžeme vhodným návrhem operačního systému snížit „špičku" zatížení.

Někdy, u složitějších počítačů řídicí proces je součástí operačního systému modul „organizátor volného času", což je program umožňující využít počítač v nespřaženém režimu (off-line), a řešit úlohy, které nesouvisí s řízeným procesem.

4) Multiprogramování

Důležitou koncepcí počítače při řízení procesu je multiprogramování, při kterém využíváme počítač pro více programů uložených v operační paměti. Tyto jednotlivé programy pak lze řešit ve vhodnou dobu.

5) Programová integrita

Programy z oblasti hromadného zpracování dat nebo vědecko-technického charakteru se zpracovávají po dávkách a jsou většinou na sobě nezávislé. V tom je podstatný rozdíl proti systému, který řídí proces v reálném čase. Všechny jednotlivé programy jsou současně v počítači (ať již v operační paměti nebo přídavné paměti). V operační paměti se nachází obvykle několik programů současně, zpravidla jsou mezi jednotlivými programy různé vazby a mnohdy se jednotlivé programy ovlivňují.

2.16 Mikroprogramování

Některé mikropočítače, zvláště řešené bipolární technologií, (např. Intel 3000), umožňují „uživatelské" mikroprogramování, tj. umožňují tvorbu vlastního strojového jazyka. Sestavení mikroprogramu znamená sestavení programu pro řadič.

V obvyklém mikropočítači řadič dekóduje každou instrukci do sledu kombinací řídicích bitů, které představují mikrooperace. V mikroprogramovaném mikropočítači mikroprogram sestává ze sledu mikroinstrukcí, které jsou uloženy v řídicí (mikroprogramované) permanentní paměti řadiče. Jinými slovy během provádění mikroprogramované instrukce se vyhledává v řídicí permanentní paměti odpovídající sled mikroinstrukčních slov. V tomto případě řadič, který nazýváme mikrořadič (MCU, microcontrol unit), pracuje jako

„vnitřní počítač". Aby mikropočítač mohl být „uživatelsky" mikroprogramován, je nezbytné, aby byla dostupná řídicí permanentní paměť.

Vhodně voleným souborem mikroinstrukcí je možné modifikovat vnější instrukce počítače. Tyto mikroinstrukce jsou ukládány do řídicí permanentní paměti.

Mikroprogramování umožňuje beze změny zapojení mikropočítače:
- Volit účinný soubor instrukcí i u relativně jednoduchého procesoru.
- Emulovat jiný soubor instrukcí, tj. mikroprogramovat instrukce jiného typu počítače, např. minipočítače. V tomto případě mohou být použity všechny programy sestavené pro minipočítač na mikroprogramovaném mikropočítači (emulátoru).
- Zvětšit propustnost počítače vhodnou volbou instrukcí pro určitou úlohu. Typickými příklady mohou být podprogramy pro vstup - výstup a aritmetiku s pohyblivou řádovou čárkou.

Mikroprogramování se též používá u systému s mikropočítačem v případě větší výrobní série a za předpokladu, že požadovaná funkce mikropočítače není složitá. Podtrhněme však, že v každém případě je sestavení programu pomocí mikroinstrukcí náročné a zdlouhavé.

Sestavení mikroprogramu se provádí obvykle podobným způsobem jako sestavení programu. Příklad postupu při mikroprogramování je následující (obr. 2.4):

1) Sestavení vývojového diagramu pro danou úlohu.
2) Zápis mikroprogramu v symbolickém mikrokódu podle vývojového diagramu.
3) Překlad mikroprogramu zapsaného v symbolickém mikrokódu (symbolického mikroprogramu) pomocí mikroasembleru, který dodává výrobce mikropočítače. Tento překlad může být proveden pomocí jazyka FORTRAN na větším počítači - nevlastní mikroasembler.
4) Ověření mikroprogramu na větším počítači simulací funkce mikropočítače.
5) Zápis mikroprogramu do permanentní paměti pomocí děrné pásky (pomocí děrné pásky obsahující mikroprogram je zhotovena

maska pro výrobu zákaznické permanentní paměti; tato děrná páska též slouží pro kontrolu vyrobených integrovaných obvodů permanentní paměti).

Zdůrazněme, že bez mikrosymbolického jazyka s mnemotechnickým vyjádřením jednotlivých mikroinstrukcí je mikroprogramování velmi obtížné.

3. MIKROPROCESOR 8080

Mikroprocesor 8080 (Intel) je zhotoven na jednom čipu se 40 vývody technologií NMOS. Má délku slova 8 bitů. Přenáší data a informace o vnitřních stavech po osmibitové obousměrné třístavové datové sběrnici. Adresování paměti a přídavných zařízení se provádí šestnáctibitovou třístavovou adresovací sběrnicí.

Některé základní charakteristické vlastnosti mikroprocesoru 8080 můžeme shrnout do následujících bodů:

- vstupy a výstupy (s výjimkou hodinových impulsů) jsou slučitelné s logickými obvody TTL
- doba cyklu 2 μs
- má šest obecných pracovních (zápisníkových) registrů a střadač
- má šestnáctibitový čítač instrukcí pro přímé adresování paměti až do kapacity 64 K bytů
- umožňuje činnost se zásobníkovou pamětí
- pracuje v dvojkovém kódu, záporná čísla jsou vyjadřována v druhém doplňku
- umožňuje činnost v dvojkově desítkovém kódu
- umožňuje vektorové přerušení
- umožňuje aritmetiku s dvojnásobnou přesností
- umožňuje připojení až 512 přímo adresovatelných vstupů - výstupů

Šest obecných pracovních registrů může být používáno jednotlivě nebo ve dvojicích při operacích s dvojnásobnou přesností. Aritmetické a logické instrukce nastavují a nulují čtyři klopné obvody příznaků. Pátý klopný obvod příznaku usnadňuje činnost v dvojkově desítkovém kódu.

Mikroprocesor 8080 může využívat část operační paměti jako zásobníkové paměti k ukládání a opětnému vybírání obsahu střada-

če, obecných pracovních registrů, klopných obvodů příznaků a čítače instrukcí. Adresování zásobníkové paměti se ovládá šestnáctibitovým ukazatelem zásobníkové paměti. Používání zásobníkové paměti umožňuje přerušování programu, při němž je třeba dočasně ukládat do paměti poslední stav mikroprocesoru; usnadňuje též vkládání podprogramů.

Architektura mikroprocesoru 8080 je volena s ohledem na nejjednodušší návrh systému. Mikroprocesor generuje ovládací signály pro styk s pamětmi a vstupně-výstupními zařízeními. Signálem HOLD lze adresovací a datovou sběrnici odpojit od vnitřních obvodů mikroprocesoru. To umožňuje budit tyto sběrnice i z jiných zdrojů, např. při blokovém přenosu nebo při spolupráci více mikroprocesorů. Skupinové schéma mikroprocesoru 8080 je na obr. 3.1.

Zapojení vývodů pouzdra na obr. 3.2. Jednotlivé vývody pouzdra mají následující funkci:

- A_0 až A_{15}; adresovací sběrnice (Adress bus) - třístavový výstup. Zajišťuje adresování operační paměti do maximální kapacity 64 K byte nebo adresování vstupů-výstupů (na A_0 až A_7 až do maximální kapacity 256 vstupů a 256 výstupů). A_0 je bit s nejnižší váhou.
- D_0 až D_7; datová sběrnice (Data bus) - třístavový vstup-výstup. Zajišťuje přenos instrukcí a dat mezi mikroprocesorem, operační pamětí a vstupně-výstupními zařízeními. Kromě toho vždy v první době (první periodě hodin) každého strojového cyklu, uvede mikroprocesor na této sběrnici stavové slovo, které popisuje probíhající strojový cyklus. D_0 je bit s nejnižší váhou.
- SYNC; synchronizační signál (Synchronizing signal) - výstup. Indikuje začátek každého strojového cyklu.
- DBIN; data na datové sběrnici (Data bus in) - výstup. DBIN oznamuje vnějším obvodům, že datová sběrnice mikroprocesoru je ve vstupním módu. Signál DBIN může být použit k otvírání kombinačních logických členů (hradel), přes která se na datovou sběrnici přivádějí data z operační paměti nebo ze vstupně-výstupních zařízení.
- READY; připraven (Ready) - vstup. Tímto signálem se oznamu-

je mikroprocesoru, že na datové sběrnici jsou přivedena platná data z vnějších zařízení. Tento signál se používá pro synchronizaci mikroprocesoru s pomalými pamětmi nebo vstupně-výstupními zařízeními. Jestliže mikroprocesor vydá adresu a neobdrží od vnějšího zařízení odezvu READY, pak přejde do stavu WAIT (čekání) na tak dlouho, dokud úroveň READY je ve stavu 0. READY může být též použit při ladění programu pro jednoduché krokování mikroprocesoru.

- WAIT; čekání (Wait) - výstup. Tento signál indikuje, že mikroprocesor je ve stavu čekání.
- WR; zápis (Write) - výstup. Signál WR je používán pro řízení zápisu do operační paměti nebo výstupu dat do vstupně-výstupních zařízení. Data předávaná z mikroprocesoru na datovou sběrnici jsou platná, pokud WR je ve stavu 0.
- HOLD; držení (Hold) - vstup. Signál HOLD požaduje, aby mikroprocesor vstoupil do stavu držení. Stav HOLD (držení) umožňuje vzít ovládání adresové a datové sběrnice vnějším zařízením od okamžiku, kdy mikroprocesor skončí právě probíhající strojní cyklus. Je to přípustné za následujících podmínek:
  - mikroprocesor je ve stavu HALT (zastavení)+)
  - mikroprocesor je ve stavu T_2 nebo stavu T_W a signál READY (připraven) je aktivní (READY = 1).

  Při vstoupení mikroprocesoru do stavu HOLD (držení) se ocitnou adresovací a datová sběrnice ve třetím stavu, t.j. stavu výsoké impedance, jinými slovy odpojí se od mikroprocesoru. Mikroprocesor oznamuje tento stav signálem HLDA.
- HLDA; potvrzení držení (Hold acknowledge) - výstup. Je to odezva na signál HOLD a indikuje, že adresovací a datová sběrnice jsou odpojeny od mikroprocesoru. Počátek signálu HLDA se může vyskytnout:
  - v době T_3 při čtení z paměti nebo vstupu dat z přídavného zařízení
  - v periodě hodin následující po době T_3 (zápis do paměti

+) Jednotlivé stavy budou vysvětleny dále.

nebo výstupu dat do přídavného zařízení).
V obou případech se signál HLDA vyskytne po nástupní hraně hodinových impulsů $Ø_1$ a odpojení sběrnic se uskuteční po nástupní hraně hodinových impulsů $Ø_2$.

- INTE; přerušení přípustné (Interrupt enable) - výstup. Indikuje obsah vnitřního klopného obvodu INTE. Tento klopný obvod může být nastaven a nulován instrukcemi EI (enable interrupt, přerušení přípustné) a DI (disable interrupt, přerušení nepřípustné).Ve vynulovaném stavu zamezuje přijetí signálu přerušení. Klopný obvod INTE je automaticky nulován v době T_1 instrukčního cyklu M_1, kdy je přerušení přijímáno. Je také nulován signálem RESET.
- INT; požadavek na přerušení (Interrupt request) - vstup. Mikroprocesor přijme žádost o přerušení na konci probíhající instrukce, nebo je-li zastaven. Je-li mikroprocesor ve stavu HOLD, nebo je-li klopný obvod INTE vynulován, pak na signál požadavku na přerušení INT není brán zřetel.
- RESET; nulování (Reset) - vstup. Tento signál musí trvat minimálně tři periody hodin; je-li aktivní, nuluje se obsah čítače instrukcí. Po nulování program začíná od instrukce uložené na adrese 0 v paměti. Signálem RESET se též nulují klopné obvody INTE a HLDA. Obsahy klopných obvodů příznaků, střadače, obecných pracovních registrů a ukazatele zásobníkové nejsou signálem RESET ovlivněny.
- Uss zemní referenční napětí
- Udd napájecí napětí + 12 V $\pm$ 5 %
- Ucc napájecí napětí + 5 V $\pm$ 5 %
- Ubb napájecí napětí - 5 V $\pm$ 5 % (předpětí substrátu)
- $Ø_1$, $Ø_2$; hodinové impulsy (Clock) - vstupy.
  Dvě fáze vnějších hodinových impulsů jsou neslučitelné s úrovněmi TTL. Požadovaný průběh hodin $Ø_1$ a $Ø_2$ je na obr. 3.3.

3.1 Architektura mikroprocesoru 8080

Mikroprocesor 8080 obsahuje následující funkční bloky (obr. 3.1):

- soubor registrů a adresovací logické obvody
- aritmetickou jednotku
- registr instrukcí a řídicí jednotku (řadič)
- obecný třístavový registr datové sběrnice

Soubor registrů

Soubor registrů sestává ze statické paměti s libovolným výběrem, která je organizována jako šest šestnáctibitových registrů:

- čítač instrukcí (PC, program counter)
- ukazatel zásobníkové paměti (SP, stack pointer)
- šest osmibitových obecných pracovních (zápisníkových) registrů organizovaných po dvojicích, označovaných B, C; D, E; a H, L
- dočasná dvojice registrů, označovaná W, Z.

Čítač instrukcí obsahuje adresu paměti právě vybírané instrukce. Jeho obsah se automaticky zvětšuje při každém výběru instrukce. Ukazatel zásobníkové paměti obsahuje adresu následujícího volného paměťového místa v zásobníkové paměti. Jeho počáteční stav (adresa) může být nastaven tak, že lze využít libovolnou část paměti s libovolným výběrem. Vkládá-li se informace do zásobníkové paměti, zmenší se obsah ukazatele zásobníkové paměti automaticky o jedničku, vybírá-li se informace, jeho obsah se automaticky o jedničku zvětší.

Šest obecných pracovních registrů se může použít jako jednoduché osmibitové registry nebo jako dvojice registrů s obsahem 16 bitů. Dočasná dvojice registrů W, Z není programově adresovatelná a používá se jen pro vnitřní operace.

Osmibitová informace může být přenášena mezi vnitřní sběrnici a jednotlivými registry přes multiplexor volby registru (register-select multiplexer). Mezi jednotlivými obecnými pracovními registry a přičítajícím/odčítajícím adresovacím registrem (blok přičítání a odčítání adres, adresovací vyrovnávací paměť) se může uskutečnit šestnáctibitový přenos informace. Přičítající/odčítající adresovací registr sestává z vlastního adresovacího registru (address latch) a přičítajícího/odčítajícího obvodu. Tento adresovací registr může přijmout informaci z kterýchkoliv tří dvojic obecných pracovních registrů a budí prostřednictvím dalšího adresovacího registru (blok oddělovací obvody adresovací sběrnice) šestnáct

adresovacích výstupů A_0 až A_{15}. Při využití přičítajícího/odčítajícího obvodu může být informace zpětně vložena do jedné ze tří dvojic obecných pracovních registrů. Informace procházející přičítajícím/odčítajícím adresovacím registrem tudíž může zůstat nezměněna nebo může být zvětšena či zmenšena o jedničku.

Aritmetické jednotka

Součástí aritmetické jednotky jsou následující registry:

- osmibitový střadač ACC
- osmibitový dočasný střadač ACT a dočasný klopný obvod přenosu
- pět klopných obvodů příznaků (pětibitový registr příznaků), které uchovávají příznak nuly, přenosu, znaménka, parity a zvláštního přenosu (pro dvojkově-desítkovou korekci)
- osmibitový dočasný registr TMP

V aritmetické jednotce se provádějí aritmetické a logické operace i operace rotace (posunu). Do aritmetické jednotky vstupují informace z dočasného registru TMP, z dočasného střadače ACT a klopného obvodu přenosu. Výsledek operace může být přenesen na vnitřní datovou sběrnici nebo přes tuto sběrnici do střadače. Aritmetická jednotka též ovlivňuje klopné obvody příznaků.

Dočasný registr TMP přijímá informace z vnitřní datové sběrnice a může předávat všechny nebo část těchto informací do aritmetické jednotky, registru příznaků nebo do vnitřní datové sběrnice. Střadač ACC, může přijmout informaci z aritmetické jednotky nebo vnitřní datové sběrnice a předávat informaci dočasnému střadači ACT nebo do vnitřní datové sběrnice. Obsah střadače ACC a klopného obvodu zvláštního přenosu může být testován pro uskutečnění desítkové korekce během provádění instrukce „desítkové nastavení střadače" (DAA, Decimal Adjust Accumulator).

Registr intrukce a řídicí jednotka

Během výběru instrukce první byte instrukce obsahující operační kód se přenese z vnitřní datové sběrnice do osmibitového registru instrukce. Obsah tohoto registru se dekóduje dekodérem instrukce. Výstupy z dekodéru instrukcí kombinované s různými časovacími signály vytváří řídicí signály pro soubor registrů, aritmetickou jednotku a další obvody mikroprocesoru. Kromě toho výstupy z dekodéru instrukce a vnější řídicí signály slouží jako vstupy pro časovací

a řídicí obvody, které generují stavové časovací signály.

Registr datové sběrnice

Osmibitový obousměrný třístavový registr (blok oddělovací obvody a vyrovnávací paměť datové sběrnice) odděluje vnitřní datovou sběrnici mikroprocesoru od vnější datové sběrnice D_0 až D_7. Při výstupu se obsah vnitřní datové sběrnice přenese do osmibitového registru, který budí výstupní výkonové stupně datové sběrnice. Tyto výkonové stupně jsou vypnuty během doby, kdy se neuskutečňuje výstup.

Při vstupu se přenášejí data z vnější datové sběrnice přes oddělovací obvody a vyrovnávací paměť do vnitřní datové sběrnice.

3.2 Příznakové bity

Operace s daty prováděné některými instrukcemi mohou ovlivnit pět klopných obvodů příznaků, které někdy nazýváme registr příznaků. Jednotlivé klopné obvody uchovávají příznak nuly, přenosu, znaménka, parity a zvláštního přenosu. Výstupy klopných obvodů příznaků vyjadřují příznakové bity, jež někdy nazýváme podmínkové bity.

Jednotlivé příznakové bity (kromě bitu vyjadřujícího zvláštní přenos) mohou být testovány instrukcemi, které podle výsledku testu ovlivňují další průběh programu.

Příznak přenosu

Klopný obvod přenosu (přesněji klopný obvod příznaku přenosu) lze ovlivnit operacemi sčítání, odečítání, rotace (posun) a logickými operacemi. Jako příklad si uveďme sečtení dvou jednobytových čísel

$$
\begin{array}{rl}
 & 1\,1\,0\,0\,0\,1\,0\,1 = C5_{16} \\
+ & 0\,1\,1\,1\,0\,1\,1\,0 = 76_{16} \\
\hline
 & 0\,0\,1\,1\,1\,0\,1\,1 \quad 13B_{16}
\end{array}
$$

přenosový bit (nastavení klopného obvodu přenosu)

Další případná operace, která by též vyvolala přenosový bit, nevynuluje klopný obvod přenosu.

Jednotlivé operace sčítání, odečítání, rotace a logické operace ovlivňují klopný obvod přenosu podle různých pravidel. In-

strukce pro sčítání jsou: ADD, ADC, ADI, ACI a DAD. Instrukce, které se používají pro odčítání jsou: SUB, SBB, SUI, SBI, CMP a CPI. Instrukce pro rotaci jsou: RAL, RAR, RLC a RRC. Instrukce pro logické operace jsou: ANA, ORA, XRA, ANI, ORI a XRI.

Příznak zvláštního přenosu

Klopný obvod zvláštního přenosu může být nastaven instrukcí DAA, která upravuje osmibitový čísla obsažená ve střadači vyjádřená ve dvojkovém kódu na osmibitová čísla vyjádřená ve dvojově desítkovém kódu. Příznakový bit zvláštního přenosu indikuje přenos z nižšího dvojkově desítkového řádu. Např.

```
  0 0 0 1 1 1 1 1 =  1F16
+ 0 0 1 1 0 0 1 1 =  3316
  ---------------    ----
  0 1 0 1 0 0 1 0    5216
        └zvláštní přenos = 1
└přenos = 0
```

Klopný obvod zvláštního přenosu nemůže být testován instrukcemi programu; ovlivňuje obvody mikroprocesoru, které provádějí převod do dvojkově desítkového kódu.

Příznak znaménka

Mikroprocesor 8080 může zpracovat v osmibitovém slovu číselný rozsah -128_{10} až $+127_{10}$, nejvyšší (sedmý) bit vyjadřuje znaménko čísla. Je-li sedmý (znaménkový) bit roven 1, je číselný rozsah -128_{10} až -1. Je-li sedmý bit roven 0, je číselný rozsah 0 až $+127_{10}$. Při provádění některých instrukcí je sedmý bit zaznamenán do klopného obvodu příznaku znaménka.

Příznak nuly

Klopný obvod příznaku nuly se nastaví, je-li výsledek operace prováděné určitými instrukcemi nulový. Tento klopný obvod se též nastaví při nulovém výsledku v rámci osmi bitů (při nulovém obsahu střadače) za současného vzniku přenosu. Např.

```
  0 1 1 1 0 0 0 1
+ 1 0 0 0 1 1 1 1
  ---------------
  0 0 0 0 0 0 0 0   příznak nuly = 1
```

```
  0 1 1 1 0 0 0 1
+ 1 0 0 0 1 1 1 1
  ---------------
  0 0 0 0 0 0 0 0
  |_____________|
        příznak nuly = 1
  |
  |→přenos
```

Příznak parity

Po některých instrukcích je vyhodnocována parita. Klopný obvod příznaku parity se nastaví, je-li ve zkoumaném bytu sudý počet jedniček, sudá parita. Je-li parita lichá, klopný obvod příznaku parity se nuluje.

3.3 Instrukční, strojový a hodinový cyklus

Instrukční cyklus je doba potřebná pro výběr a provedení instrukce. Během výběrové doby instrukce zvolená jednobytová, dvoubytová nebo tříbytová instrukce se vybere z paměti a operační znak (první byte) se přenese do registru instrukce. Během prováděcí doby se instrukce dekóduje a provede.

Každý instrukční cyklus sestává z jednoho až pěti strojových cyklů. Strojový cyklus je potřebný vždy, když mikroprocesor komunikuje s pamětí nebo vstupně-výstupním zařízením. Výběrová doba instrukčního cyklu vyžaduje jeden strojový cyklus pro každý vybíraný byte instrukce. Prováděcí doba instrukčního cyklu závisí na typu prováděné instrukce. Některé instrukce nevyžadují pro prováděcí dobu žádný strojový cyklus, t.j. instrukční cyklus těchto instrukcí sestává jen ze strojových cyklů, které jsou nutné pro výběr instrukce. Jiné instrukce vyžadují pro prováděcí dobu strojové cykly, které jsou nezbytné pro čtení (nebo zápis) dat v paměti nebo ve vstupně-výstupním zařízení. Výjimku tvoří instrukce DAD (sčítání obsahů dvojic obecných pracovních registrů), které vyžadují další dva strojové cykly.

Každý strojový cyklus sestává ze tří, čtyř nebo pěti stavů, které někdy nazýváme hodinové cykly. Stav (hodinový cyklus) je časový interval mezi následujícími nástupními hranami hodinových impulsů $Ø_1$, to značí, že stav odpovídá periodě hodinových impulsů.

Úplný instrukční cyklus vyžaduje v závislosti na typu instrukce 4 až 17 stavů.

Mikroprocesor 8080 je vnějším obvodem buzen dvěma fázemi $Ø_1$ $Ø_2$ hodinových impulsů, které se nepřekrývají. V časovacím a řídicím bloku mikroprocesoru se vytvoří z hodinových impulsů $Ø_1$ a $Ø_2$ synchronizační impulsy (SYNC), jak je uvedeno na obr. 3.4.

Jsou tři výjimky z uvedené definice doby trvání stavu: stav čekání (WAIT), stav klidu (držení;HLDA, hold acknowledge) a stav zastavení (HLTA, halt acknowledge). Jejich význam si uvedeme později. Jelikož tyto tři stavy závisí na vnějších událostech, mohou trvat libovolně dlouho. Avšak i tyto stavy musí být synchronizovány hodinovými impulsy; jejich doba trvání je celistvým násobkem periody hodinových impulsů.

3.4 Identifikace strojového cyklu

Počet adresování operační paměti nebo vstupně-výstupního zařízení, který je třeba uskutečnit pro výběr a provedení určité instrukce, odpovídá počtu strojových cyklů instrukčního cyklu. Výjimku tvoří instrukce DAD. Podobně jako mnohé jiné mikroprocesory mikroprocesor 8080 je řešen tak, že může vyslat jen jednu adresu během jednoho strojového cyklu. Proto instrukční cyklus instrukce, která vyžaduje pro výběr a provedení dvakrát vstoupit do paměti, obsahuje dva strojové cykly. Je-li třeba pěti těchto vstupů, instrukční cyklus obsahuje pět strojových cyklů.

Během instrukčního cyklu se vyskytuje alespoň jeden vstup do paměti, během něhož se vybere příslušná instrukce. Jinými slovy první strojový cyklus v každém instrukčním cyklu odpovídá výběru instrukce. Případné další strojové cykly v instrukčním cyklu závisí na typu instrukce. Uveďme si některé příklady. Instrukce ADDr (přičtení obsahu zvoleného obecného pracovního registru k obsahu střadače) vyžaduje jen jeden strojový cyklus, který je třeba pro její výběr; další vstup do paměti instrukce nevyžaduje. Jelikož tato instrukce vyžaduje tři stavy (hodinové cykly) pro výběr instrukce a jeden stav pro její provedení, sestává strojový cyklus ze čtyř stavů, t.j. ze čtyř period hodinových impulsů.

Uvažujme nyní instrukci ADDM (přičtení obsahu zvolené paměťo-

vé buňky k obsahu střadače). Ačkoliv tato instrukce je podobná dříve uvažované instrukci ADDr, vyžaduje další strojový cyklus, který je nezbytný pro adresování zvolené paměťové buňky. Při uvažovaném instrukčním cyklu mikroprocesor nejprve vybere z paměti jednobytovou instrukci adresovanou čítačem instrukcí, což trvá 3 hodinové cykly. Při tomto výběrovém strojovém cyklu se instrukce uložená do registru instrukce, dekóduje a použije k řízení činnosti pro zbývající dobu instrukčního cyklu. Během této zbývající doby instrukčního cyklu mikroprocesor nejprve adresuje operační paměť obsahy registrů H a L. Následuje přečtení osmibitového údaje z paměti a jeho přenesení do dočasného registru uvnitř mikroprocesoru (strojový cyklus „čtení z paměti" vyžaduje 3 stavy). Během sedmého (posledního) stavu se přičte obsah dočasného registru k obsahu střadače. Tudíž instrukce ADDM vyžaduje dva strojové cykly sestávající celkem ze sedmi stavů (hodinových cyklů).

Jako další příklad si uveďme instrukci SHLD (uložení obsahů registrů H a L do paměti adresované druhým a třetím bytem této instrukce), která vyžaduje pět strojových cyklů:

- První, výběrový, strojový cyklus sestává ze čtyř stavů. Během prvních tří stavů mikroprocesor vybírá z paměti instrukci (operační znak) adresovanou čítačem instrukcí, jehož obsah se pak zvýší o jedničku. Během čtvrtého stavu se obsahy registrů H a L přenesou do dočasných registrů W a Z mikroprocesoru.
- Druhý strojový cyklus „čtení z paměti" má tři stavy. Během tohoto strojového cyklu se čte druhý byte instrukce adresovaný čítačem instrukcí z paměti a uloží do registru L. Obsah čítače instrukcí se opět zvětší o jedničku.
- Třetí strojový cyklus, během něhož se provádí čtení třetího bytu instrukce z operační paměti do registru H, obsahuje též tři stavy. Obsah čítače instrukcí se nastaví pro výběr další instrukce.
- Čtvrtý strojový cyklus, během něhož se provádí přenos obsahu registru Z do paměti adresované registry H a L, sestává ze tří stavů. Stav následující za přenosem se používá pro zvětšení obsahu registrů H a L o jedničku pro adresaci následující paměťové buňky.
- Pátý strojový cyklus, během kterého se přenáší obsah registru W do paměti adresované registry H a L, má též tři stavy.

Souhrně instrukce SHLD sestává celkem z 5 strojových cyklů a vyžaduje celkem 16 stavů.

Počet strojových cyklů většiny instrukcí spadá mezi uvedené dva extrémy (instrukce ADDr, SHLD). Např. instrukce INP (vstup) nebo OUT (výstup) vyžadují tři strojové cykly; jeden strojový cyklus pro výběr prvního bytu instrukce (operačního znaku), druhý pro „čtení z paměti" (získání adresy přídavného zařízení; druhý byte instrukce) a třetí pro vstup nebo výstup.

V instrukčním cyklu se může vyskytnout devět typů strojových cyklů: výběr instrukce, čtení z paměti, zápis do paměti, čtení ze zásobníkové paměti, zápis do zásobníkové paměti, vstup z přídavného zařízení, zápis do přídavného zařízení, přerušení a zastavení.

Mikroprocesor poskytuje informaci o typu strojového cyklu pomocí osmibitového stavového slova během každého prvního stavu strojového cyklu. Tyto signály se vyskytují na datové sběrnici D_0 až D_7 během synchronizačního impulsu (SYNC); tab. 3.1. Aby bylo možné odvodit řídicí signály pro vnější obvody, je třeba uvažované stavové signály uložit do registru a dekódovat je. Stavové signály lze též využít pro identifikaci strojových cyklů při zkoušení a ladění během vývoje systému. Aby vnější dekódovací obvody byly jednoduché, jsou jednotlivé typy strojových cyklů poměrně jednoduše kódovány (tab. 3.2).

3.5 Časování a synchronizace, sled stavů

Časování a synchronizace mikroprocesoru s ostatními obvody mikropočítače patří k závažným problémům při návrhu systému. Mikropočítač, podobně jako jiné typy počítačů, je synchronní sekvenční stroj, který provádí určitý sled operací během následných stavů (hodinových cyklů). Sled jednotlivých stavů během strojového cyklu je uveden na obr. 3.5, který představuje stavový přechodový diagram. Tento diagram též znázorňuje, jak jsou časovány „vývody" mikroprocesoru (připraven) READY, (držení) HOLD, (požadavek na přerušení) INT a jak signály na těchto „vývodech" mohou měnit základní sled přechodů.

Jak jsme si již uvedli, každý instrukční cyklus, během něhož se uskutečňuje výběr a provedení instrukce, se skládá z jednoho

až pěti strojových cyklů. Každý strojový cyklus se skládá ze tří až pěti stavů, které označujeme T_1, T_2, T_3, T_4, T_5 a které mohou být doplněny stavem T_W. Tyto jednotlivé stavy jsou ve stavovém přechodovém diagramu označeny kroužky.

Jednotlivé stavy ve strojovém cyklu jsou uvedeny ve vztahu k fázím obou hodinových impulsů $Ø_1$ a $Ø_2$ v časovém diagramu na obr. 3.6., který je též doplněn dalším, šestým stavem T_W (WAIT, stav čekání). Hodinové impulsy $Ø_1$ a $Ø_2$ se nepřekrývají, a mají nestejnou dobu trvání. Nejsou též vzájemně symetrické: náběžná hrana impulsů $Ø_2$ je blíže sestupné hraně impulsů $Ø_1$ než nástupní hrana $Ø_1$ k sestupné hraně $Ø_2$ (obr. 3.3.).

Během stavu T_1 se obsah čítače instrukcí vyskytuje na adresovací sběrnici A_0 až A_{15}. Po sestupné hraně hodinového impulsu $Ø_1$ během stavu T_1, je vysílán synchronizační signál na vodiči synchronizace (SYNC). Tento signál identifikuje první stav t.j. T_1 každého strojového cyklu. Může být použit pro aktivaci vnějších obvodů při vstupu do paměti nebo výstupu z paměti. Během synchronizačního impulsu též je na datové sběrnici D_0 až D_7 stavová informace (stavové slovo), pomocí něhož mohou vnější obvody stanovit současný stav mikroprocesoru.

Z obr. 3.5 je zřejmé, že po stavu T_1 následuje vždy stav T_2. Během stavu T_2 testuje mikroprocesor vstupy READY (připraven), HOLD (držení) a výstup HLTA (Halt acknowledge, potvrzení zastavení). V závislosti na těchto tří signálech mikroprocesor vstupuje do různých stavů. Signály READY a HOLD poskytují vnější obvody. Konstruktér může použít tyto signály pro synchronizaci činnosti mikroprocesoru a vnějšími zařízeními, např. s pamětí s delší vybavovací dobou, nebo pro krokování programu. Po stavu T_2 procesor vstoupí buď do stavu čekání T_W, nebo do stavu T_3. Ve stavu T_W mikroprocesor nejprve zkoumá, je-li prováděna instrukce HLT (zastavení, Halt). V kladném případě (podle obr. 3.5) mikroprocesor vstoupí do stavu čekání T_{WH} a zůstane v tomto stavu dokud nepřijme signál HOLD nebo signál požadavku na přerušení INT. Je-li přijat signál HOLD, nastaví se vnitřní klopný obvod HOLD a mikroprocesor zůstane ve stavu držení (klidu, HOLD), dokud tento signál působí. Po jeho ukončení vrací se mikroprocesor do stavu čekání. Účelem stavu držení (HOLD), je odpojení mikroprocesoru od adreso-

vací a datové sběrnice, t.j. uvést vývody mikroprocesoru A_0 až A_{15} a D_0 až D_{15} do stavu vysoké impedance, takže tyto sběrnice mohou být využívány vnějšími zařízeními, aniž by byly ovlivňovány mikroprocesorem. Přerušení, které je indikováno vstupním signálem INT, způsobí ukončení stavu čekání, aby mohl být proveden podprogram příslušející přerušení.

Neprovádí-li se instrukce HLT, mikroprocesor vstoupí do doby čekání T_W (obr. 3.5), v které zůstane, dokud se nevyskytne na vstupu signál READY. Tento signál indikuje, že na datové sběrnici se vyskytují platná data pro mikroprocesor. Není-li přijat signál HOLD, mikroprocesor vstoupí do stavu T_3.

Proces, který probíhá během stavu T_3 závisí na typu probíhajícího strojového cyklu. Během strojového cyklu, který provádí výběr instrukce, mikroprocesor interpretuje informaci na datové sběrnici jako instrukci, která se přenese do registru instrukce. Během strojového cyklu, v němž se čte z operační nebo zásobníkové paměti, jsou informace na datové sběrnici interpretovány jako data. Během vstupně-výstupních operací mikroprocesor buď přijímá nebo poskytuje na datové sběrnici data podle toho, provádí-li se vstupní nebo výstupní operace.

V případě, že během stavu T_3 byla vybrána instrukce, přenese se během stavu T_4 do dekodéru instrukcí a rovněž dekóduje. Stanoví se, kolik stavů a strojových cyklů je třeba pro provedení instrukce a jsou generovány potřebné časovací a synchronizační signály.

Jako příklad si uveďme jednoduchou instrukci ADD B (přičtení obsahu registru B k obsahu střadače). Tato instrukce vyžaduje jen 4 stavy strojového cyklu. Po dosažení stavu T_4 se procesor vrací na stav T_1 nového strojového cyklu, ve kterém začne provádět další instrukci.

Některé složitější instrukce vyžadují stav T_5 a případně další strojové cykly pro vykonání jediné instrukce. Např. instrukci pro vstup dat IN $\langle B_2 \rangle$, kde $\langle B_2 \rangle$ je druhý byte instrukce, tato je provedena během 3 strojových cyklů, které obsahují celkem 10 stavů (hodinových cyklů); obr. 3.7. Během nástupní hrany $Ø_2$ v době T_2 se odstraní stavová informace 01000101 (viz. tab. 3.2) z datové sběrnice D_0 až D_7 a sběrnice se připraví na vstup dat. Data se musí

vyskytovat na sběrnici dobu t_{DS1} (obr. 3.7) před sestupnou hranou $Ø_1$ ve stavu T_3, také dobu t_{DS2} před nástupní hranou $Ø_2$ a ještě během doby t_{DH}, měřené od nástupní hrany následujícího impulsu $Ø_2$. Data vyskytující se na datové sběrnici z paměti nebo z přídavného zařízení, jsou vzorkována během stavu T_3.

Druhý strojový cyklus M_2 obsahuje jen stavy T_1, T_2, T_3 je označen podle tab. 3.2 stavovým slovem 0100001. Během něho se čte druhý byte $< B_2 >$ instrukce. Třetí strojový cyklus M_3, který má též stavy T_1, T_2, T_3, má stavové slovo 01000010 (v případě čtení z paměti 01000001). Při něm se čtou data z adresovaného vstupního zařízení.

Během vstupu dat do mikroprocesoru vystupuje z mikroprocesoru signál DBIN, který může být využit pro přenos ve vnějších obvodech. Strojové cykly, ve kterých se vyskytuje signál DBIN jsou: výběr instrukce, čtení z paměti, čtení ze zásobníkové paměti a přerušení. Vyskytuje-li se mezi stavy T_2 a T_3 stav T_W (čekání), vyskytuje se signál DBIN i během T_W.

3.6 Dekódování stavů, vazba mikroprocesoru na okolí

Pro synchronizaci činnosti mikroprocesoru 8080 s vnějšími logickými obvody poskytuje mikroprocesor informaci o svém stavu během každého strojového cyklu během stavu T_1 na datové sběrnici. Pro využití tohoto stavového slova vnějšími logickými obvody je nezbytné, aby stavové slovo bylo zapamatováno do příchodu dalších stavových informací, t.j. po dobu jednoho strojového cyklu. Pro tento účel je možné použít integrovaný obvod Intel 8212, který přejímá stavové informace z datové sběrnice D_0 až D_7 (obr. 3.8, 3.9).

Dalšími vstupy obvodu 8212 SYNC a hodinové impulsy $Ø_1$ zajišťují výběr stavových informací ve vhodný okamžik. Obvod 8212 obsahuje klopné obvody typu D, řídicí obvody a třístavové výstupní obvody; je zřejmé, že jej lze nahradit standardními integrovanými obvody nízké případně střední integrace, což však je ekonomicky nevýhodné. Stavové informace na výstupu 8212 musí být pro další použití dekódovány podle tab. 3.2.

Jiný obvod, řešený bipolární technologií, který si zapamatuje

stavovou informaci, je Intel 8228. Tento obvod též dekóduje stavovou informaci a kromě toho má výkonové stupně pro datovou sběrnici. Příklad jeho začlenění do mikropočítače s 8080 je na obr. 3.10.

3.7 Soubor instrukcí

Mikroprocesor 8080 má 78 instrukcí,z nichž 48 instrukcí je stejných jako u staršího mikroprocesoru 8008. Instrukce jsou jednobytové, dvoubytové nebo tříbytové. Vícebytové instrukce jsou uloženy na následných adresách v paměti. První byte instrukce vyjadřuje operační znak. Sestává-li instrukce z více bytů, druhý, případně třetí byte představuje data nebo adresu. Jednobytové instrukce provádějí přenosy mezi obecnými pracovními (zápisníkovými) registry, aritmetické operace, adresování paměti, rotace, vkládání a vybírání ze zásobníkové paměti, aktivaci a pasivaci přerušení. Dvoubytové instrukce obsahují operace s okamžitým adresováním (operand je obsažen v druhém bytu) a vstupně-výstupní operace. Tříbytové instrukce zahrnují např. podmíněné a nepodmíněné skoky, přímé zavádění dat do dvojic obecných pracovních registrů a ukazatele zásobníkové paměti, přímé ukládání (nebo vybírání) obsahů některých registrů do operační paměti.

Soubor instrukcí můžeme rozdělit do následujících pěti skupin:

1) Instrukce pro přesuny dat (11 instrukcí)
   Zahrnuje instrukce pro přenos dat mezi obecnými pracovními registry, mezi obecnými pracovními registry v paměti a instrukce pro adresování paměti pomocí tří dvojic obecných pracovních registrů (B a C, D a E, H a L).
2) Aritmetické instrukce (14 instrukcí)
   Zahrnuje instrukce pro aritmetické operace s daty, která jsou uložena v registrech a paměti, dále instrukce pro zvětšení nebo zmenšení obsahu registrovaných dvojic (INX, DCX) a instrukci pro desítkovou aritmetiku (DAA).
3) Instrukce pro logické operace (15 instrukcí)
   Zahrnuje instrukce pro provedení logických operací na data uložená v obecných pracovních registrech nebo v paměti a na příznakové bity.

4) Instrukce pro větvení programu (29 instrukcí)
   Zahrnuje instrukce pro podmíněné a nepodmíněné skoky. Podmíněné skoky mohou být vyhodnocovány podle osmi podmínek (podle stavu čtyř klopných obvodů příznaků).
5) Instrukce pro řízení zásobníkové paměti, vstupu - výstupu a řídicí instrukce (9 instrukcí)
   Zahrnuje instrukce pro řízení zásobníkové paměti, vstupní - výstupní instrukce, dále instrukce pro ukládání a vybírání obsahů dvojic obecných pracovních registrů a stavového slova do zásobníkové paměti, a též instrukce pro aktivaci a pasivaci (EI, DI).

Soubor instrukcí mikroprocesoru 8080 je uveden v tab. 3.3. Instrukce se vyjadřují ve stroji výhradně ve dvojkovém kódu. Pro snažší zápis instrukce se však někdy používá osmičkový (oktálový) nebo šestnáctkový (hexadecimální) kód podle následujícího schématu:

jednotlivé bity instrukce	$D_7\ D_6$	$D_5\ D_4\ D_3$	$D_2\ D_1\ D_0$
osmičkový kód	X	Y	Z
šestnáctkový kód	U		V

Příklad: první byte instrukce ORI

dvojkově	1 1 1 1 0 1 1 0
osmičkově	3 6 6
šestnáctkově	F 6

Poznamenejme, že rozšířením počtu instrukcí ze 48 u mikroprocesoru 8008 na 78 u mikroprocesoru 8080 se často snižuje požadavek na kapacitu paměti pro stejnou úlohu o 70 až 95 %, čímž se též samozřejmě zrychluje provádění úlohy. Rovněž se snížila doba základního instrukčního cyklu z 12,5 μs (8008) na 2 μs (8080).

Instrukce mikroprocesoru 8080 pracují s následujícími registry:

1) A — střadač (formát 1 byte)

B, C, D, E, H, L — obecné pracovní registry (formát 1 byte)

M — paměťový registr (formát 1 byte). Je to fiktivní registr, který je tvořen kteroukoliv buňkou paměti. Proto musí být M specifikován adresou.

Registr, ze kterého se přenáší byte dat, je v instrukcích označován písmenem S (source index register). Registr, do kterého se ukládá byte dat, je v instrukcích označován písmenem D (destination index register). Za S a D se do kódu instrukcí v tab. 3.3 dosazují kódy požadovaných registrů podle tab. 3.4.

2) B C, D E, H L — dvojice registrů (formát 2 byty). Používají se v instrukcích s dvojnásobnou přesností. Jsou tvořeny sdružením jednoduchých registrů. Registry B, D, H uchovávají vyšší bity, C, E a L nižší bity.

SP — ukazatel zásobníkové paměti (stack pointer, formát 2 byty).

Instrukce mikroprocesoru 8080 pracují s následujícími klopnými obvody příznaků (formát 1 bit):

Z - nula (zero). Příznak se nastaví do hodnoty 1 (pravdivý stav) při nulovém výsledku operace (t.j. nulovém obsahu střadače), jinak je vynulován (nepravdivý stav).

S - znaménko (sign). Příznak se nastaví do hodnoty 1, jestliže je bit s nejvyšší váhou výsledku operace 1, jinak je vynulován.

P - parita (parity). Příznak se nastaví do hodnoty 1 při sudé paritě výsledku operace, při liché paritě je vynulován.

CY- přenos (carry). Příznak se nastaví do hodnoty 1, jestliže přeteče nejvyšší bit výsledku operace t.j. nastane-li přenos v nejvyšším dvojkovém řádu, jinak je vynulován.

AC- zvláštní (pomocný) přenos (auxiliary carry). Příznak se nasta-

ví do hodnoty 1, jestliže přeteče čtvrtý bit výsledku operace, jinak je vynulován.

3.8 Adresování dat a instrukcí

Adresování instrukcí v plném rozsahu 64 K operační paměti se provádí, podobně jako u jiných počítačů, čítačem instrukcí. Adresování dat se provádí u mikroprocesoru 8080 4 způsoby:

1. Přímé adresování

Druhý a třetí byte instrukce obsahuje adresu paměti, na které je uložen operand. Nižší bity adresy jsou obsaženy v druhém bytu, vyšší bity adresy jsou obsaženy ve třetím bytu.

Jako příklad použití přímého adresování uveďme instrukci, kterou máme přenést obsah paměťové buňky (údaj sestávající z 8 bitů) s adresou $1A2F_{16}$ do střadače (instrukce LDA, 1A, 2F). Uložení této instrukce v paměti je na obr. 3.11.

2. Nepřímé adresování pomocí dvojice registrů (registrové nepřímé adresování)

Instrukce specifikuje dvojici obecných pracovních registrů, které obsahují adresu paměťové buňky, ve které je uložen údaj. U většiny instrukcí se používá dvojice registrů H a L; registr H obsahuje vyšší bity adresy, registr L nižší bity adresy. Příkladem může být instrukce ADD M, kde M je adresováno registry H a L. Některé instrukce používají dvojice registrů B a C nebo D a E.

3. Okamžité adresování

Při použití tohoto typu „adresování" není adresována paměť, ale instrukce obsahuje údaj. Některé instrukce obsahují údaj sestávající z 8 bitů, jiné instrukce ze 16 bitů. Příkladem může být instrukce ADI, která provádí přičtení druhého bytu k obsahu střadače.

4. Adresování obecných pracovních registrů mikroprocesoru

Při tomto způsobu adresování údaj není v operační paměti, ale v některém z obecných pracovních registrů, nebo ve dvojici těchto registrů. Instrukce specifikuje požadovaný registr nebo dvojici registrů. Příkladem může být instrukce ADDr, která provádí přičtení obsahu obecného pracovního registru r k obsahu střadače.

Zpracovávané instrukce jsou uloženy v paměti na následných adresách. Výjimku tvoří přerušení a instrukce pro větvení programu. Instrukce pro větvení mohou specifikovat adresu následující instrukce dvěma způsoby:

a) Přímo. Instrukce pro větvení obsahuje adresu následující instrukce, která má být provedena. Příkladem může být instrukce JMP.
b) Registrově nepřímo. Instrukce pro větvení určuje dvojici obecných pracovních registrů, které obsahují adresu následující instrukce. Příklad: instrukce PCHL (přesun obsahů H a L do čítače instrukcí).

Instrukce RST je speciální jednobytová instrukce, kterou obvykle používáme při přerušení. Touto instrukcí se vyvolá instrukce, která je umístěna na adrese rovné osminásobku obsahu tříbitového pole, které obsahuje instrukce RST.

3.9 Adresování zásobníkové paměti

Zásobníkové paměť je vytvořena v části paměti s libovolným výběrem, která je součástí operační paměti mikropočítače. Jsou jen dvě operace, které lze uskutečnit se zásobníkovou pamětí; vložení a vybrání informace.

Při vkládání informace se šestnáct bitů přenese do zásobníkové paměti z dvojice obecných pracovních registrů nebo z čítače instrukcí. Postup při vkládání informace do zásobníkové paměti má následující sled:

a) nejvyšších 8 bitů informace se uloží na adresu, která je o jedničku menší, než obsah ukazatele zásobníkové paměti,
b) nejnižších 8 bitů informace se uloží na adresu, která je o dvě menší, než obsah ukazatele zásobníkové paměti,
c) obsah ukazatele zásobníkové paměti se automaticky sníží o dvě.

Při výběru informace ze zásobníkové paměti se přenese 16 bitů do dvojice obecných pracovních registrů nebo do čítače instrukcí. Postup je následující:

a) přenese se nejnižších 8 bitů z adresy určené obsahem ukazatele zásobníkové paměti do dvojice obecných pracovních registrů nebo do čítače instrukcí.
b) přenese se nejvyšších 8 bitů z adresy určené obsahem ukazatele zásobníkové paměti zvětšeným o jedničku do dvojice pracovních registrů nebo do čítače instrukcí.
c) obsah ukazatele zásobníkové paměti se automaticky zvětší o dvě.

Před využitím zásobníkové paměti musí programátor nastavit ukazatel zásobníkové paměti. Libovolná hodnota se do ukazatele zásobníkové paměti zavede instrukcí LXI SP. Má-li být vyvolán podprogram, je v příslušném místě hlavního programu instrukce pro vyvolání podprogramu (např. CALL, nebo CP). Během této instrukce je obsah čítače instrukcí (t.j. adresa bezprostředně následující instrukce) uložen do zásobníkové paměti. Následuje provádění jednotlivých instrukcí podprogramu. Poslední instrukcí podprogramu je instrukce pro návrat do hlavního programu, která přenese posledně vloženou informaci do zásobníkové paměti čítače instrukcí. Touto informací je adresa první instrukce v hlavním programu, která následuje po instrukci, kterou byl podprogram vyvolán.

Podprogramy mohou být vkládány do libovolné hloubky; jediné omezení je v hloubce zásobníkové paměti, kterou jsme vyčlenili v paměti s libovolným výběrem.

3.10 Adresování vstupů - výstupů

Mikroprocesor musí být schopen komunikovat kromě s operační pamětí ještě s přídavnými zařízeními, např. dálnopisem, snímačem a děrovačem děrné pásky, zobrazovací jednotkou a pamětí s magnetickým prožným diskem. V podstatě na strukturu vstupu - výstupu můžeme pohlížet jako na uspořádání jednobytových paměťových buněk, představované vyrovnávacími registry přídavných zařízení, do kterých můžeme zapisovat (výstup), nebo ze kterých můžeme číst (vstup). Mikroprocesor 8080 má zvláštní instrukce pro vstup a výstup (IN, OUT). Tyto instrukce „oddělují" paměťové buňky v operační paměti a „paměťové buňky" představující přídavná zařízení z hlediska vstupu - výstupu, v tomto případě rozsah adresování operační paměti není ovlivněn strukturou vstup - výstup, přídavná zařízení mají vlastní adresy. Přenos mezi střadačem a zvoleným přídavným zařízením se uskuteční adresováním zvoleného vstupu - výstupu, na kterém je připojeno uvažované přídavné zařízení. Tento způsob přenosu mezi mikroprocesorem, operační pamětí a přídavnými zařízeními nazýváme dvousběrnicové uspořádání (Isolated I/O), i když sběrnice nejsou fyzicky rozděleny.

Existuje ještě druhý způsob adresování přídavných zařízení. Při tomto způsobu adresování jsou považovány „jednotlivé paměťové

buňky" (registry),které zprostředkují vstup nebo výstup na jednotlivá přídavná zařízení, jako část operační paměti. Tento způsob adresování přídavných zařízení nazýváme jednosběrnicové uspořádání (memory mapped I/O).

Adresování vstupů - výstupů při dvousběrnicovém uspořádání

Toto uspořádání je znázorněno na obr. 3.10. Při této architektuře vstupu - výstupu je plně odděleno adresování operační paměti a adresování vstupu - výstupu pomocí signálů řídicí sběrnice. Kapacitu operační paměti je možné volit v maximálním rozsahu 64 K. Přenos mezi střadačem a vstupem - výstupem se uskutečňuje poměrně jednoduše pomocí instrukcí IN (vstup) a OUT (výstup).

Adresování vstupu - výstupu při jednosběrnicovém uspořádání

Vyčleněním určitých adres operační paměti adresám vstupu - výstupu získáme architekturu, při které můžeme použít pro vstupy - výstupy stejné instrukce, které se používají ve spojitosti s operační pamětí. Pro jednosběrnicové uspořádání, podobně jako pro dvousběrnicové uspořádání, můžeme použít integrovaný obvod 8228 viz obr. 3.12, kde jsou generovány nové řídicí signály klíčováním signálů $\overline{MEMR}$ (čtení z paměti) a $\overline{MEMW}$ (zápis do paměti) signálem A_{15} (nejvyšší adresovací bit). Nově generované signály $\overline{I/OR(MM)}$ a $\overline{I/OW(MM)}$ jsou použity stejným způsobem jako signály $\overline{I/OR}$ a $\overline{I/OW}$ při dvousběrnicovém uspořádání, takže systém sběrnic se nezmění.

Přiřadíme-li signálu A_{15} „příznak" vstupu - výstupu, pak $A_{15} = 0$ značí, že je aktivní paměť a $A_{15} = 1$ značí, že je aktivní vstup - výstup. Je samozřejmé, že můžeme použít jiný adresovací bit. Výhoda volby A_{15} spočívá ve snažším řízení programem, přičemž zůstává ještě možnost použít maximální kapacitu operační paměti 32 K.

Jelikož při jednosběrnicovém uspořádání můžeme použít všechny instrukce určené pro činnost s operační pamětí též pro vstupy - výstupy, nemusí nastávat jen přenos mezi střadačem a vstupem - výstupem, ale též mezi vstupem - výstupem a ostatními obecnými pracovními registry mikroprocesoru.

Uveďme si příklady instrukcí, které můžeme použít pro vstup - výstup:

MOV r, M	přenos ze vstupu do obecného pracovního registru
MOV M, r	přenos na výstup z obecného pracovního registru
MVI M	výstup dat obsažených v instrukci na výstup
LDA	přenos ze vstupu do střadače
STA	přenos ze střadače na výstup
LHLD	16 bitový vstup
SHLD	16 bitový výstup
ADD M	přičtení data na vstupu k obsahu střadače
ANA M	logický součin vstupu a obsahu střadače

Možností použití instrukcí původně určených pro operační paměť též pro vstupy - výstupy se značně zvyšuje propustnost mikropočítače.

Adresování vstupů - výstupů lineární selekcí

Nejčastější metodou adresování u obou způsobů řešení přenosu dat mezi vstupy - výstupy a mikroprocesorem (t.j. u jednosběrnicového i dvousběrnicového uspořádání) je dekódování adresy, kterou se aktivuje požadované vstupně - výstupní zařízení. Dekódování je stejného charakteru jako dekódování jednotlivých paměťových buněk operační paměti.

Jinou metodou adresování je lineární selekce, kterou se dosahuje zpravidla materiálová úspora ovšem za cenu možnosti připojení menšího počtu přídavných zařízení. U této metody není třeba dekodér, neboť ke každému bitu adresovací sběrnice je přiřazeno jedno vstupně - výstupní zařízení.

Jako příklad adresování lineární selekcí uvažujeme u dvousběrnicového uspořádání tvar druhého bytu instrukce IN nebo OUT uvedený na obr. 3.13. Vazba na sběrnice nechť je provedena programovatelnými vazebními obvody (programmable peripherical interface) Intel 8255, z nichž každý má 3 vstupa - výstupy. Tyto vstupy - výstupy jsou adresovány bity A_0 a A_1 adresovací sběrnice. Dalšími 6 bity A_2 až A_7 se provádí volba jednoho ze šesti připojených programovatelných vazebních obvodů. Tímto způsobem lze adresovat 18 jednobytových vstupů - výstupů.

Jako další příklad uvažujeme adresování lineární selekcí u jednosběrnicového uspořádání, kterým adresujeme 13 programovatelných vazebních obvodů Intel 8255 připojených na adresovací sběrnici bez použití dekodéru. Tvar druhého a třetího bytu instrukce LDA

nebo STA (případně jiné instrukce používané ve spojitosti s operační pamětí) je na obr. 3.14. Bity A_0 a A_1 se opět provádí volba jednoho ze tří vstupů - výstupů jednotlivých programovatelných vazebních obvodů. Bity A_2 až A_{14} adresovací sběrnice se provádí volba jednoho ze 13 připojených programovatelných vazebních obvodů; tím se získá celkem 13 x 3 = 39 jednobytových vstupů - výstupů. Bit A_{15} rozlišuje adresování operační paměti a vstupů - výstupů.

3.11 Přerušení

Mikroprocesor 8080 umožňuje pomocí technického vybavení přerušit probíhající program. Požadavek na přerušení vznikne z vnějšího zařízení, např. klávesnice, pomocí níž máme vkládat data do mikropočítače. Přerušení umožňuje především přijímat data z relativně pomalých (vzhledem k mikropočítači) vnějších zařízení a „současně" řešit úlohu obsaženou v hlavním programu. Tento způsob nazýváme jednoslovný přenos pomocí přerušení. Přenos sice lze uskutečnit podmíněným jednoslovným přenosem, který je jednodušší, avšak pomalejší, neboť mezi přenosy mikropočítač „nic nedělá", jen se dotazuje, zda jsou připravena další data.

Z mikroprocesoru vychází signál INTE (interrupt enable, přerušení přípustné), který může být ve stavu 0 nebo 1 podle stavu vnitřního klopného obvodu INTE mikroprocesoru. Tento klopný obvod může být nastaven nebo nulován instrukcemi EI a DI (enable interupt, disable interrupt, přerušení přípustné, přerušení nepřípustné). Je-li INTE = 0, je systém přerušení pasivován t.j. žádné přerušení nemůže být přijato.

Rozpozná-li mikroprocesor požadavek na přerušení z přídavného zařízení, nastane sled následujících operací:

a) dokončí se prováděná instrukce

b) klopný obvod, připouštějící přerušení se vynuluje, t.j. INTE = = 0

c) zařízení, které vyvolalo přerušení, poskytne pomocí technického vybavení jednu instrukci, kterou mikroprocesor provede. Tato instrukce se nevyskytuje v operační paměti a programátor ji nemůže ovlivnit; je jen funkcí řadiče zařízení, které vyvolalo přerušení. Před touto instrukcí se obsah čítače instrukce nezvětší.

Instrukcí vyvolávající přerušení, kterou poskytuje zařízení,

je obvykle RST. Tato účinná jednobytová instrukce vyvolává jeden z osmi programů uložených v prvních 64 slovech paměti. Např. dálnopis může poskytnout instrukci RST OH (obr. 3.15) při každém stisknutí klávesnice a tím vyvolat přerušení probíhajícího programu. Každá instrukce RST OH vyvolá podprogram,který zajistí přenos data z dálnopisu do mikroprocesoru. Tento podprogram, který může sestávat až z osmi bytů, je uložen na adresách 0000_{16} až 0007_{16} paměti.

Jiné zařízení může poskytovat instrukci RST 1H. Podprogram, který „obslouží" toto zařízení, je uložen na adresách 0008_{16} až $000F_{16}$ paměti. Poznamenejme, že v případě potřeby každý z uvažovaných osmi programů může vyvolat další podprogram. Přídavné zařízení též nemusí poskytnout jen instrukci RST, ale libovolnou instrukci mikroprocesoru 8080.

Na obr. 3.16 je uveden příklad sledu činnosti probíhající při přerušení. Zařízení č. 1 signalizuje požadavek na přerušení „přerušení od zařízení č. 1", během provádění mikroprocesor instrukce na adrese $3C0B_{16}$. Tato instrukce se dokončí. Čítač instrukcí zůstane nastaven na $3C0C_{16}$ a zařízení č. 1 poskytne instrukci RST OH. Jelikož tato instrukce vyvolává skok na adresu 0000_{16}, uloží se obsah čítače instrukcí $3C0C_{16}$ do zásobníkové paměti a do čítače instrukcí vloží 0000_{16}. Při provádění instrukce RET (návrat z podprogramu), která je na konci podprogramu, se přenese adresa $3C0C_{16}$ ze zásobníkové paměti do čítače instrukcí a tím pokračuje původní program od místa přerušení.

Obecně všechny obsahy obecných pracovních registrů klopných obvodů příznaků (podmínkových klopných obvodů), které mohou být ovlivněny podprogramem přerušení, musí být obnoveny před návratem z podprogramu, jinak vznikne chyba. Jako příklad uvažujme přerušení programu před instrukcí JC, $\langle B_2 \rangle$, $\langle B_3 \rangle$, která značí Jump on Carry, t.j. je-li (přenos) = 1, pak (PC) $\leftarrow \langle B_3 \rangle \langle B_2 \rangle$ jinak (PC) = (PC) + 3. Předpokládejme, že (přenos) = 1. Vynuluje-li se během podprogramu přerušení klopný obvod přenosu, pak po návratu do původního programu nenastane skok na adresu $\langle B_2 \rangle$, $\langle B_3 \rangle$, ale na (PC) + 3 a tím vznikne chyba ve výpočtu.

Podobně jako u jiných podprogramů, tak i u podprogramu přerušení by se měly uložit alespoň obsahy klopných obvodů příznaků a tyto obsahy opět obnovit před návratem do původního programu, t.j.

před instrukcí RET (Return). Nejsnažší cestou je uložení obsahů těchto klopných obvodů do zásobníkové paměti.

Potvrdí-li mikroprocesor na datové sběrnici příjem signálu přerušení signálem INTA (Interrupt acknowledge, potvrzení příjmu signálu přerušení), překlopí se též klopný obvod INTE (Interrupt enable, přerušení přípustné) a signál INTE z mikroprocesoru již nepřipustí další přerušení. Výjimku tvoří případ, kdy podprogram obsluhující přerušení obsahuje instrukci EI (Enable interrupt, přerušení přípustné) od níž je možné zpracovat další přerušení. Jinými slovy po provedení instrukce EI podprogram obsluhující přerušení může být přerušen jiným podprogramem, který obsluhuje nově vzniklé přerušení. Tento proces může pokračovat až do libovolné úrovně. Aby program probíhal bezchybně, je však nezbytné, aby byla správně ukládána a obnovována data. Příklad podprogramu příslušející přerušení je v tab. 3.5.

3.12 Rodina integrovaných obvodů k mikroprocesoru 8080

Pro uskladnění různých sestav mikropočítačů s mikroprocesorem 8080 vyvinula firma Intel (podobně jako jiné firmy pro své mikroprocesory) řadu speciálních „pomocných" integrovaných obvodů. Tyto obvody lze sice (kromě pamětí) nahradit obvody sestavenými ze standardních integrovaných obvodů nízké a střední integrace případně doplněnými diskrétními prvky, avšak je to cesta technicky náročná a zpravidla ekonomicky nevýhodná. Vždyť lze i celý mikroprocesor sestavit ze standardních integrovaných obvodů. Je třeba podtrhnout, že současný vývoj se více orientuje na rozšiřování sortimentu a zdokonalování těchto „pomocných" obvodů než na zdokonalování vlastních mikroprocesorů.

Jako příklad „pomocných" obvodů k mikroprocesoru 8080 si uveďme:

1) Systémové obvody

8224	generátor hodinových impulsů
8228	systémový řadič a budič sběrnice
8238	systémový řadič a budič sběrnic

2) Vazební obvody pro číslicový vstup - výstup

8202	třístavový osmibitový budič sběrnice

8203 třístavový osmibitový budič sběrnice (invertující)
8205 dekodér 1 z 8
82LS05 dekodér 1 z 8
83006 osmibitová vyrovnávací paměť pro vstup - výstup
8208 osmibitový obousměrný budič sběrnice
8212 osmibitový vstup - výstup
8213 osmibitový obousměrný vstup - výstup
8216 čtyřbitový obousměrný budič sběrnice
8226 čtyřbitový obousměrný budič sběrnice (invertující)

3) Vazební obvody pro přídavná zařízení

8244 kodér pro klávesnici s 90 tlačítky
8245 kodér pro klávesnici s 16 tlačítky
8246 kodér pro klávesnici s 20 tlačítky
8247 řadič čtyřčíslicové zobrazovací jednotky
8248 řadič šestičíslicové zobrazovací jednotky
8253 programovatelný časovací obvod
8254 programovatelný bitově adresovatelný vazební obvod
8255 programovatelný vazební obvod
8257 programovatelný řadič blokového přenosu (DMA)
8258 programovatelný řadič blokového přenosu
8214 prioritní přerušení
8259 programovatelný řadič přerušení
8272 řadič paměti s magnetickým pružným diskem
8276 řadič obrazovkové zobrazovací jednotky
8285 generátor znaků
8292 osmibitový analogově číslicový převodník s šestnáctikanálovým analogovým multiplexorem
8294 tříčíslicový voltmetr s multiplexovaným výstupem v dvojkově desítkovém kódu

4) Vazební obvody pro komunikační účely

8250 asynchronní komunikační obvod
8251 programovatelný komunikační vazební obvod
8252 programovatelný komunikační vazební obvod
8261 programovatelný komunikační subsystém
8274 komunikační řadič
8283 komunikační řadič

5) Paměti

8356	permanentní paměť 2048 x 8 a paměť s libovolným výběrem 128 x 8
8154	statická paměť s libovolným výberem 128 x 8
8364	permanentní paměť programovaná maskou 8192 x 8, MOS
8364E	permanentní paměť programovaná maskou 8192 x 8, MOS
8316A/E	permanentní paměť programovaná maskou 2048 x 8, MOS
8332E	permanentní paměť programovaná maskou 4096 x 8, MOS
1702A	reprogramovatelná permanentní paměť 256 x 8
8704	reprogramovatelná permanentní paměť 512 x 8
2708/8708	reprogramovatelná permanentní paměť 1024 x 8
8101A-4	statická paměť s libovolným výběrem 256 x 4
8111A-4	statická paměť s libovolným výběrem 256 x 4
8102A	statická paměť s libovolným výběrem 1024 x 1
74C920	statická paměť s libovolným výběrem 256 x 4, CMOS
74C921	statická paměť s libovolným výběrem 256 x 4, CMOS
74C929	statická paměť s libovolným výběrem 1024 x 1, CMOS
2114	statická paměť s libovolným výběrem 1024 x 4
MM257	statická paměť s libovolným výběrem 4096 x 1
DM87S296	bipolární programovatelná permanentní paměť 512 x 8
5290	dynamická paměť s libovolným výběrem 16K
8316A/E	permanentní paměť programovaná maskou
DM74S472	bipolární programovatelná permanentní paměť 512 x 8
MM5204	reprogramovatelná permanentní paměť 512 x 8
8702A	reprogramovatelná permanentní paměť 256 x 8
8708/8704	reprogramovatelná permanentní paměť 1024 x 8
8302	permanentní paměť programovaná maskou 256 x 8
8308	permanentní paměť programovaná maskou 1024 x 8
8316A	permanentní paměť programovaná maskou, 2048 x 8
81078-4	dynamická paměť s libovolným výběrem 4K x 1
5101	statická paměť s libovolným výběrem 256 x 4, CMOS

Podrobný popis jednotlivých „pomocných" obvodů, z nichž některé svou složitostí jsou srovnatelné se samotným mikroprocesorem, ve vymyká rámci této publikace. Proveďme si alespoň základní popis některých těchto obvodů, které se často vyskytují v různých sestavách mikropočítačů.

Dekodér 1 z 8

Obvod 8205 obsahuje dekodér 1 z 8; je zhotoven bipolární technologií. Přijímá tříbitový dvojkový kód na vstupech A_0, A_1, A_2, který je dekódovaný výstupy $\overline{O_0}$ až $\overline{O_7}$ za předpokladu, že toto dekódování připouštějí aktivační vstupy $\overline{E_1}$, $\overline{E_2}$ a E_3, obr. 3.17.

Činnost obvodu je určena pravdivostní tabulkou 3.6. Např. vyskytuje-li se na vstupech A_0, A_1, A_2 dvojkový kód 101 a současně vstupy $\overline{E_1}$, $\overline{E_2}$, E_3 připouštějí dekódování ($\overline{E_1} = 0$, $\overline{E_2} = 0$, $E_3 = 1$), pak výstupy jsou $\overline{O_0} = 1$, $\overline{O_1} = 1$, $\overline{O_3} = 1$, $\overline{O_4} = 1$, $\overline{O_5} = 0$, $\overline{O_6} = 1$, $\overline{O_7} = 1$.

Obvod 8205 se může použít v mikropočítačovém systému pro různé účely. Např. může sloužit k dokódování adresovací sběrnice pro určení vstupu - výstupu mikropočítače (obr. 3.18). Cstupu A_0 můžeme přisoudit váhu 2^0, vstupu A_4 váhu 2^4. Vstupy A_0 až A_4 jsou buzeny adresovací sběrnicí. Jednotlivé výstupy „0" až „23" obvodu 8205 jsou vedeny k příslušným vstupům - výstupům mikropočítače, které jsou adresovány adresovací sběrnicí.

Jiný příklad použití obvodu je dekódování jednotlivých paměťových obvodů pomocí jejich vstupů „aktivace čipu" (obr. 3.19). Předpokládejme, že chceme adresovat paměť s celkovou kapacitou 24K bytů, která má být sestavena z jednotlivých paměťových obvodů (čipů) o kapacitě 1K bitů. Nechť každý tento paměťový obvod má deset adresovacích vstupů a jeden vstup „aktivace čipu". Pak nižší adresovací bity A_0 až A_9 adresovací sběrnice mikropočítače vedou na všechny paměťové obvody a vyšší adresovací bity A_{10} až A_{14} se dekódují pomocí obvodů 8205 na signály „aktivace čipu", které vedou k jednotlivým skupinám paměťových obvodů.

Osmibitový vstup - výstup 8212

Obvod 8212 je vyroben bipolární technologií. Obsahuje osm klopných obvodů (do kterých se zaznamenává osmibitové vstupní informace), třístavové výstupní obvody, řídicí a výberové logické obvody. Dále obsahuje klopný obvod indikace požadavku na přerušení pro mikroprocesor (obr. 3.9). Pomocí tohoto obvodu může být řešena většina vstupů - výstupů mikroprocesorového systému. Na jednu vstupně - výstupní sběrnici může být připojeno více obvodů 8212, volba vstupně - výstupního zařízení může být provedena dekodérem 1 z 8, který je obsažen v obvodu 8205.

Obvod prioritního přerušení 8214

Obvod 8214 je zhotoven bipolární technologií. Podstatně usnadňuje řešení systému přerušení mikropočítače. Může přijmout až osm požadavků na přerušení, stanovit jejich nejvyšší prioritu, zpracovat víceúrovňové přerušení a vyslat vektor přerušení. Tento vektor přerušení se použije pro nastavení čítače instrukcí mikroprocesoru na adresu první instrukce příslušného podprogramu přerušení. Pro tento účel se obvykle používá pevně zapojené instrukce RST. Skupinové schéma obvodu 8214 je na obr. 3.20, příklad jeho začlenění do systému je na obr. 3.21.

Použitím několika obvodů 8214 lze vytvořit složitější systém přerušení s více vstupy a více úrovněmi.

Čtyřbitový obousměrný budič sběrnice 8216 resp. 8226

Jednotlivé vývody mikroprocesoru jsou obvykle schopny budit jen jednu jednotkovou zátěž TTL. Podobně je tomu u paměti MOS. Tyto výstupy jsou dostačující jen pro malé systémové sestavy. Často je však třeba tyto výstupy zesílit, což lze provést pomocí obvodu 8216 nebo obvodu 8226, který se liší od 8216 jen invertujícími vstupy. Výstupní obvody poskytují ve stavu 1 3,65 V a ve stavu 0 50 mA. Zapojení obvodu 8216 je na obr. 3.22. Např. při požadavku zvětšení logického zisku vývodů obousměrné datové sběrnice mikroprocesoru připojujeme propojené vývody DI a DO obvodu 8226 k jednotlivým vývodům datové sběrnice D mikroprocesoru.

Programovatelný vazební obvod 8255

Obvod 8255 zhotovený technologií MOS, je určen pro vazbu mezi mikroprocesorem a vstupně - výstupním zařízením. Má 24 vstupně - výstupních vývodů z pouzdra, které mohou být individuálně programovány ve dvou skupinách po 12 a použity ve třech základních módech činnosti.

V módu 0, základní vstup - výstup jsou obě skupiny 12 vstupů - výstupů rozděleny do podskupin po čtyřech vstupech - výstupech. Vývody v těchto podskupinách mohou být buď vstupy nebo výstupy.

V módu 1 (strobovaný vstup - výstup), každá skupina dvanácti vstupů - výstupů může být naprogramována tak, že má 8 vstupů nebo výstupů. Čtyři zbylé vodiče v každé skupině jsou určeny pro řídicí signály (strobování) a přerušení.

V módu 2, (obousměrná sběrnice) osm vodičů je použito pro obousměrnou sběrnici a pět vodičů (přičemž jeden je vypůjčen z jiné skupiny) pro řízení. Toto funkční uspořádání umožňuje obousměrnou komunikaci s přídavným zařízením na jedné osmibitové sběrnici.

Programovatelný komunikační vazební obvod 8251

Obvod 8251 je zhotoven technologií MOS. Někdy se nazývá univerzální synchronní - asynchronní přijímač - vysílač (USART, Universal synchronous/asynchronous receiver/transmiter). Přijímá data (znaky) z mikroprocesoru v paralelním tvaru a mění je na sériový tvar vhodný pro přenos. Současně může přijímat seriový sled dat a měnit je na paralelní tvar, který vstupuje do mikroprocesoru. Tento obvod též signalizuje mikroprocesoru, může-li přijmout nový údaj pro vyslání nebo přijal-li údaj pro mikroprocesor.

Před přenosem dat musí být z mikroprocesoru do obvodu 8251 zavedena řídicí slova. Tato řídicí slova plně definují funkci obvodu 8251; určí rychlost přenosu, délku znaku, počet bitů pro „stop", asynchronní nebo synchronní mód přenosu, lichou nebo sudou paritu atd.

Programovatelný časovací obvod 8253

Mikropočítač může být spojen s mechanickým zařízením, např. tiskacím zařízením nebo krokovým motorem. Tato zařízení mají zpoždění, která nelze zanedbat. Jedno řešení je použití smyčky v programu, s níž se tvoří zpoždění - to však mnohdy nebývá výhodné.

Výhodnější je použit programovatelný časovací obvod 8253 (obr. 3.23), který je zhotoven technologií NMOS. Obsahuje tři šestnáctibitové čítače, z nichž každý může čítat až do rychlosti 2MHz; do těchto registrů lze zapsat počáteční stav.

Obsah registru řídicího slova určuje pracovní režimy jednotlivých čítačů. Zápisem řídicího slova z mikroprocesoru zahájí zvolený čítač čítání, a tím odměřování času. Naplnění čítače a ukončený časový interval vyvolá přerušení. Toto přerušení vyvolá podprogram příslušející zvolenému časovému intervalu. Je zřejmé, že programové řízení obvodu 8253 je omezeno na minimum, spočívá v přenosu dvou bytů z mikroprocesoru do obvodu 8253. Vícenásobná zpoždění se získá přiřazením určitých úrovní přerušení různým čítačům.

Obvod 8253 se začleňuje do systému jako běžné vazební obvody

(obr. 3.24); z hlediska systému je považován za čtyři vstupy- výstupy; tři představují čítače a čtvrtý je registr řídicího slova. Výstupy všech tří čítačů jsou přiváděny prostřednictvím vhodných obvodů (např. programovatelného řadiče přerušení 8259) na vstup mikroprocesoru „požadavek na přerušení".

Programovatelný řadič blokového přenosu 8257

Rychlost jednoslovného přenosu je v některých případech (např. pro komunikaci s pamětí s magnetickým pružným diskem) nedostatečná. Mikroprocesor 8080 umožňuje uskutečnit blokový přenos, během kterého nastává přímá komunikace mezi operační pamětí a přídavným zařízením; mikroprocesor se přenosu nezúčastní.

Obvod 8257 umožňuje vytvořit čtyři kanály blokového přenosu. Každému kanálu je přiřazena určitá priorita. Vznikne-li více požadavků na blokový přenos, je obslouženo „důležitější" přídavné zařízení (velkokapacitní paměť). Iniciátorem požadavku na blokový přenos je přídavné zařízení. Je-li tento požadavek přijat z hlediska priority, je vyslán do mikroprocesoru signál HOLD (držení). Signálem HLDA (potvrzení držení) oznámí mikroprocesor, že se odpojil od datové a adresovací sběrnice, čímž uvolňuje tyto sběrnice pro blokový přenos. Obvod 8257 vytváří posloupnost za sebou následujících adres v operační paměti, z kterých se čte nebo do kterých se zapisuje. Tento obvod též čítá jednotlivé cykly (přenášená slova) a ukončí blokový přenos po dosažení programovaného počtu slov, t.j. po programované délce bloku.

Přenos pomocí obvodu 8257 se nemusí uskutečnit vcelku, s vyřazením mikroprocesoru 8080 z činnosti, ale po několika slovech v prováděcích dobách každého instrukčního cyklu, což nazýváme „kradení cyklů". Při tomto přenosu pokračuje i řešení „hlavního" programu.

Řídicí slovo zapsané do 8257 pomocí programu určuje režim činnosti, počáteční adresu bloku v operační paměti, délku bloku a pořadí priorit kanálů. Tyto funkce a parametry se mohou měnit v průběhu programu zápisem nového řídicího slova.

Začlenění programovatelného řadiče blokového přenosu 8257 je na obr. 3.25. Zvláštní řídící signál $\overline{\text{BUSEN}}$ je veden do obvodu 8228, čímž zajišťuje uvolňování datové a řídicí sběrnice ve vhodných dobách.

Programovatelný řadič přerušení 8259

Obvod 8259 může zpracovat až 8 úrovní požadavků na přerušení. Přiřazení priority se provádí zápisem řídicího slova do obvodu 8259 pomocí programu a může se měnit v průběhu programu. Připojením dalších obvodů 8259 lze systém rozšířit až na 64 úrovní přerušení.

Začlenění programovatelného řadiče přerušení 8259 do systému je na obr. 3.26. V případě požadavku přerušení vyšle programovatelný řadič přerušení signál INT do mikroprocesoru. Po přijetí přerušení vyšle mikroprocesor prostřednictvím systémového řadiče 8228 signál INTA (potvrzení přerušení). V důsledku signálu INTA vyšle programovatelný řadič přerušení vektor přerušení, který vyvolá podprogram řešící úlohu spojenou s přerušením.

V případě, že je použito více obvodů 8259, t.j. více než osm úrovní přerušení, určí se zvláštním vstupem SP, který obvod 8259 je řídící (hlavní; $\overline{SP}$ = 1) a který řízený (vedlejší; $\overline{SP}$ = 0). Řídicí programovatelný řadič přerušení 8259 přijme signál INT z řízeného obvodu 8259 a vyšle požadavek na přerušení do mikroprocesoru. Řídicí obvod 8259 přijme signál $\overline{INTA}$ (potvrzení přerušení) ze systémového řadiče 8228 a vyšle do sběrnice první byte „vyvolání". Na následující signál $\overline{INTA}$ vyšle řízený obvod 8259, který vyvolal přerušení, vektor přerušení.

3.13 Návěstí, operační znak, operand, pseudoinstrukce v jazyku symbolických adres

V jazyku symbolických adres, běžně používaném pro mikroprocesor 8080 je třeba dodržovat tyto zásady:

- Návěští může sestávat z 1 až 5 znaků, prvním znakem musí být písmeno, za posledním znakem musí následovat dvojtečka
- pro návěstí nemohou být používány znaky, které se používají pro pseudoinstrukce, operační znaky s kterými se označují obecné pracovní registry
- za operačním znakem musí následovat alespoň jedna mezera
- před komentářem musí být středník.

Operand může být specifikován:

1) Údajem v šestnáctkové soustavě; za číslem musí následovat písmeno H

2) Údajem v desítkové soustavě; za číslem může následovat D, nebo může zůstat neoznačené
3) Údajem v osmičkové soustavě; za číslem musí následovat O nebo Q
4) Údajem v dvojkové soustavě; za číslem musí následovat písmeno B
5) Obsahem čítače instrukcí; je označen znakem $ a je identický s adresou další instrukce. Např. GO: JMP $ + 6
6) Kódem ASCII; znaky musí být v úvozovkách, v nejvyšším bitu je O. Např. MVI E "T" značí naplnit registr E osmi bity představující v kódu ASCII písmeno T s O v nejvyšším bitu
7) Symboly, jimž je přiřazeno asemblerem číslo. Symbolu B je přiřazeno číslo O a značí registr B; podobně symbolům C, D, E, H, L, M a A jsou přiřazena čísla 1, 2, 3, 4, 5, 6, 7 a značí příslušné registry. Příklad: nechť symbolu HODN je přiřazeno číslo $9F_{16}$; pak následující tři instrukce jsou ekvivalentní: MVI D, HODN; MVI 2, 9FH; MVI 2, HODN
8) Návěštím, které se vyskytuje v poli návěští jiné instrukce
9) Aritmetickými a logickými výrazy, které mohou obsahovat operandy uvedené pod body 1) až 8). V těchto výrazech mohou být operace +, -, *, /, MOD (celistvý zbytek získaný dělením prvního operandu druhým operandem), NOT, AND, OR, XOR a posuvný SHR a SHL. Všechny operátory zpracovávají operandy jako šestnáctibitové výrazy a vytváří výsledek též jako šestnáctibitový výraz. Příklad: instrukce MVI H, NOT O AND OFFH zavádí FF_{16} do registru H; instrukce MVI H, NOT O je neplatná, neboť NOT O generuje šestnáctibitové číslo $FFFF_{16}$; instrukce ZDE: MVI C, ZDE SHRS ležící na adrese $2E1A_{16}$ zavádí číslo $2E_{16}$ do registru C; instrukce MVI D, 34 + 4OH/2 zavádí číslo 34 + 64/2 = 66_{10} do registru D; instrukce MVI D, (34 + 4OH)/2 zavádí číslo 49_{10} do registru D.

Zápis dat lze provést následujícími operačními znaky:

A) DB; definuje osmibitové slovo. Operandem může být:
   a) aritmetický nebo logický výraz, který může obsahovat všechny aritmetické a logické operace vztahující se na operandy, které nejsou delší než 8 bitů.
   b) sled znaků v kódu ASCII, na jehož počátku a konci jsou uvozovky.

Příklady: příkazem DB 0A3H se zapisuje $A3_{16}$; příkazem DB 5ABCH SHR8 se zapisuje $5A_{16}$; příkazem DB - 03H se zapisuje FD_{16}; příkazem DB5 * 2, 2FH-0AH se zapisuje $0A25_{16}$.

B) DW; definuje šestnáctibitové slovo. Nižších 8 bitů se ukládá do nižší adresy, vyšších 8 bitů se ukládá na následující (vyšší) adresu. Příklady: je-li přiřazena návěští NAV adresa $3B1C_{16}$ a návěští NAV2 adresa $3EB4_{16}$, pak příkazem DW NAV se zapisuje $1C3B_{16}$, příkazem DW NAV2 se zapisuje $B43E_{16}$. Příkazem DW 3C01H, 3CAEH se zapisuje $013CAE3C_{16}$.

C) DS; definuje (reservuje) paměťové buňky. Příklady: instrukce PETR: DS 10 reservuje deset následujících adres; instrukce DS 10H reservuje šestnáct následujících adres.

V jazyku symbolických adres lze použít následující pseudoinstrukce:

ORG definuje počáteční adresu
Příklad : ORG 2000H značí, že instrukce následující za touto instrukcí se budou ukládat do adresy 2000_{16}

EQU přiřazuje hodnotu návěští
Příklad: VST: EQU 8 přiřazuje návěští VST hodnotu 8

SET přiřazuje hodnotu návěští stejně jako pseudoinstrukce EQU, může však být použita v programu vícekrát pro přiřazení nové hodnota návěští

END značí konec překladu

IF, ENDIF asembler zhodnotí operand příslušející IF; je-li nulový, pak instrukce mezi IF a ENDIF jsou ignorovány, není-li operand nulový, jsou respektovány

Příklad :
```
        COND    SET     OFFH
                IF      COND
                MOV     A, C
                ENDIF
        COND    SET     0
                IF      COND
                MOV     A, C
                ENDIF
                XRA     C
```

TITLE umožňují označení výpisu překladu.
Příklad: TITLE „text"; text může obsahovat maximálně 68 znaků včetně úvozovek

EJECT umožňuje zapisovat na další stranu následující část výpisu překladu

MACRO, ENDM definuje makroinstrukci.