```
23  OZ4567:   if RI(14) = 1  then goto OZ67 ;
24            if RI(13) = 1  then goto SNZ ;

25  SNU       X := 15 ;  AR := SP ;  SP → ΣL;   -1→ ΣP ;
26                    DR := HP(AR);     SP := SP + (-1);
27  SCYK      if DR(15) = 1  then goto E_ČT_INST ;
28            DR → ΣL ;  0 → ΣP ;
29            DR := L_POSUV(DR + 0);
30            X → ΣL ;   -1→ ΣP ;
31                X := X + (-1);
32            if X(15) = 0  then goto SCYK ;
33            goto SKO ;

34  SNZ:      AR := SP ;    SP →ΣL ;    -1→ ΣP ;
35            DR := HP(AR);    SP := SP + (-1);
36            if  DR(15) = 0  then goto SKU ;
37            goto E_ČT_INST ;

38  OZ67:     if RI(13) = 1  then goto OZ7 ;

39  SPR:      SP→ΣL ;   +1→ΣP ;
40              SP := SP + 1 ;
41              AR := SP ;     ČI→ΣL ;   0→ΣP ;
42                                DR := ČI + 0 ;
43                      HP(AR) := DR ;
44            goto SKO ;

45  OZ7:      if RI(2) = 1  then goto SNA ;
46            if RI(1) = 1  then goto NASDIV ;

47  PLUMIN:   AR := SP ;    SP→ΣL ;  -1→ΣP ;
48            DR := HP(AR);    SP := SP + (-1) ;      /* 1. operand T */
49            if RI(0) = 0  then goto PLU ;
50            DR→ΣL ; +1→ΣP ;          /* jen pro */
51            DR := INVERZE(DR)+ 1 ;       /* MIN     */
52  PLU:      AR := SP ;   A := DR ;
53            DR := HP(AR) ;                /* 2. operand S */
54            A →ΣL ;  DR→ΣP ;
55               DR := A + DR ;
56            HP(AR) := DR ;
57            goto E_ČT_INST ;
58  NASDIV:   if RI(0) = 1  then goto DIV ;

59  NAS:      AR := SP ;      SP→ΣL ;    -1→Σ P ;    X := 15 ;
60            DR := HP(AR);      SP := SP + (-1) ;
61            C := DR ;    AR := SP ;    0→ΣL ;    0→ΣP ;
62                      DR  := HP(AR) ;    A := 0 + 0 ;
```

Obr. 4.7.2 Interpret (1. pokračování)

```
63              if  DR(15) = 0   then  goto  NCYK ;
64              INVERZE(DR) →ΣL ;    +1→ΣP ;
65                    DR := INVERZE(DR) + 1 ;
66              D := DR ; +1→ΣL ; INVERZE(C)→ΣP ;
67                     DR := 1 + INVERZE(C) ;
68              C := DR ;   0→ΣL ;   D→ΣP ;
69                     DR := 0 + D ;
70 NYCK:      if DR(0) = 0   then  goto  NESČITAT;
71              A→ΣL ;   C→ΣP ;
72                     A := A + C ;
73 NESČITAT:  DR→ΣL ;   0→ΣP ;
74              DR := P_POSUV(DR + 0);
75              0→ΣL ;    D→ΣP ;
76              D := P_POSUV(0 + D);
77              if A(0) = 0   then  goto NEPRENOS;  /* do řádu D(15) */
78              MINUS→ΣL ;   D→ΣP ;
79                    D := MINUS + D ;
80 NEPRENOS:  A→ΣL ;   0→ΣP ;
81              A := P_POSUV(A + 0);
82              if C(15) = 0   then  goto KONEC_NAS ;
83              A→ΣL ;  MINUS→ΣP ;
84                    A :=  A + MINUS ;
85 KONEC_NAS: X→ΣL ;  -1→ΣP ;
86                    X := X + (-1);
87              if  X(15) = 0  then  goto NCYK ;
88              0→ΣL ;   D→ΣP ;
89                  DR := 0 + D ;
90                 HP(AR) := DR ;
91              goto  E_ČT_INST ;
92 DIV:       AR := SP ;       SP→ΣL ;   -1→ΣP ;              X := 15;
93              DR := HP(AR);     SP := SP + (-1);
94              C := DR ;         AR := SP ;          B := DR;     0→ΣL;0→ΣP;
95                               DR := HP(AR) ;                   A := 0 + 0 ;
96                               D := DR ;
97                            if DR(15)=1 then goto DCKLAD ;
98                            INVERZE(DR)→ΣL; +1→ΣP ;
99                               DR := INVERZE(DR) +1 ;
100                                 D := DR ;          B→ΣL; +1→ΣP;
101                                                 DR:= INVERZE(B) +1;
102                                                  B :=DR ;
103 DCKLAD:    1→ΣL;  INVERZE(C)→ΣP;
104             DR:=1 + INVERZE(C);
105             if C(15)=0 then goto DCYK;
106             C:=DR; INVERZE(DR)→ΣL;+1→ΣP;
107                          DR:=INVERZE(DR)+1;
108 DCYK        A → ΣL;     0 → ΣP;
109             A := L_POSUV(A + 0) ;
```

Obr. 4.7.2 Interpret (2. pokračování)

```
110             if D(15) = 0   then  goto  NEPIS ;
111             A→ΣL ;  +1→ΣP ;
112                A := A + 1 ;
113  NEPIS:     0→ΣL ;  D→ΣP ;
114             D := L_POSUV(0 + D);
115             A→ΣL ;  DR →ΣP ;
116                A := A + DR ;
117             if A(15) = 1  then  goto  ZBYTZAP ;
118             1→ΣL ;  D→ΣP ;
119                D := 1 + D ;
120  ZBYTZAP:   if A(15) = 0  then goto   ZBYTKLAD ;
121             A→ΣL;  C→ ΣP ;
122                A := A + C ;
123  ZBYTKLAD:  X→ΣL ;  -1→ΣP ;
124                X := X + (-1);
125             if X(15) = 0  then  goto DCYK ;
126             if B(15) = 0  then  goto KONECDIV ;
127             1→ΣL ; INVERZE(D)→ Σ P ;
128                D := 1 + INVERZE(D) ;
129  KONECDIV:  0 →ΣL ; D→ΣP ;
130               DR := 0 + D;
131              HP(AR) := DR ;
132             goto E_ČT_INST  ;

133  SNA:       AR := SP ;   SP→ΣL ; -1→Σ P ;
134             DR := HP(DR);   SP := SP + (-1) ;
135             RI := DR ;
136             goto SKO ;
    end INTERPRET_CILOVEHO_POCITACE
```

Obr. 4.7.2 Interpret (konec)

Analýza interpretu instrukcí CIKAP ne HOMIP (obr. 4.7.2)

P 5.1 Mikroinstrukce s návěštím E_CT_INST je začátek Etapy ČTení INSTrukce; řádky 0, 1 a 2 "čtou" instrukci strojového jazyka CIKAP a HP a připravují ČI na čtení následující instrukce.

Řádky 3, 4, 5, 16, 23, 24, 38, 45, 46, 49 a 58 analyzují OZ instrukce postupným testováním řádů 15, 14, 13, 13, 14, 13, 13, 2, 1 a 0 registru instrukcí. Uvedené řádky realizují dekódovací strom z obr. 4.7.1. Poznamenejme, že instrukce PLU a MIN sdílejí společnou část programu.

Uvedený způsob dekódování libovolné struktury OZ je velmi všeobecný. Jeho pomocí je možno analyzovat libovolnou strukturu OZ. Na druhé straně je nevýhodný, protože je náročný na čas. Na každý řád OZ je potřebný jeden test neboli takt μ-počítače. Na počítačích EC, kde OZ má 8 dvojkových řádů, bychom potřebovali 8 taktů. Na hostitelském počítači může být použit rychlejší způsob dekódování. U nejrychlejšího způsobu je kombinace bitů v OZ současně adresou v μHP, kde začíná μ-program provádění dané instrukce strojového jazyka. Potom by stačilo přesunout obsah RI(OZ) do ČμI (implicitní skok). Mezi dvěma extrémními způsoby dekódování OZ přirozeně existuje množství dalších kompromisních řešení.

P 5.2 Interpretace instrukcí PUSH a POP je jednoduchá. Každá z nich potřebuje 5 μ-instrukcí. Poslední μ-instrukce je skok na začátek μ-programu (začátek etapy čtení následující instrukce strojového jazyka z HP).

Interpretace podmíněných skoků SZA a SNZ vyžaduje jen po čtyřech μ-instrukcích; nepodmíněný skok na SKO dokonce jen dvě.

Interpretace podmíněného skoku SNU je složitější než SZA a SNZ, protože je třeba testovat všech 16 bitů slova vybraného z vrcholu zásobníku. Nejjednodušší způsob interpretace by se skládal ze šestnácti μ-operací TEST_A_PODMINENY_SKOK. Tento způsob je na našem počítači v HOMIP i nejrychlejší, ale spotřebuje více slov μHP. V interpretaci podle obr. 4.7.2 se používá registr jako čítač cyklů; cyklus se opakuje 16krát. V každém cyklu se otestuje znaménkový řád DR a potom se vždy obsah DR posune o 1 bit vlevo. Čítač cyklů X se před cyklem nastaví na hodnotu 15. Cyklus končí:

- buď tehdy, když se v DR(15) objeví jednička,
- nebo tehdy, když se znaménkový bit registru X (bit v X(15)) změní na jedničku; to znamená, když X dosáhne hodnoty -1. Tento způsob se používá proto, že je výhodnější testovat jeden bit v X než testovat všechny bity X na nulu.

Interpretaci instrukcí PLU a MIN tu uvádíme jen proto, že se pro interpretaci obou instrukcí použitá tatáž část interpretu; rozdíl S - T se počítá jako součet S + (-T). Pouze řádek 49 zjišťuje, zda jde o instrukci PLU. Pokud se nejedná o PLU, připočte se jednoduše k 1. operandu doplněk 2. operandu. Jestliže je 2. operand, přicházející z HP, kladný, převede se na záporný; je-li záporný, převádí se na kladný. Čísla se v HP CIKAP uchovávají v doplňkovém kódu - tak jsme se právě teď dohodli.

Interpretaci SPR a SNA si prostudujeme přímo v interpretu.

Dále podrobně probereme interpretaci instrukcí NAS a DIV tak, jak jsme na to předem upozornili v podkapitole 2.5.

Interpretace NAS

P 5.3 Algoritmy násobení a dělení jsou relativně složité. Projevuje se to i na μ-programu. Algoritmus násobení je realizován částí μ-programu, která začíná návěštím NAS (adresa 59).

Vyjdeme opět z ručního násobení. Na obr. P 5.1 je rozepsáno, jak bychom ručně násobili 69 x 25 ve dvojkové soustavě s šestnáctiřádovými slovy. Na začátku předpokládejme pouze násobení celých kladných čísel (pevná řádová čárka bez znaménka). Každý ze šestnácti parciálních součinů se tvoří násobením násobence s jedním bitem násobitele (0 nebo 1). Z toho plyne, že parciální součin je buď roven násobenci, nebo nule.

Mezi ručním a strojovým násobením však existuje několik rozdílů. Jedním z nich je to, že v HOMIP je sčítání dvouoperandová operace. Tento počítač neumí tedy sčítat víc operandů současně tak jako lidé. Po vytvoření každého z 15 parciálních součinů je třeba provést součet právě vytvořeného součinu a součtu předcházejících parciálních součinů.

```
                                    bit 15          bit 0
                                      ↓               ↓
Násobenec                          0000000001000101 = 69(10)
Násobitel                    x     0000000000011001 = 25(10)
Parciální součin podle bitu 0 násobitele  0000000001000101
Parciální součin podle bitu 1 násobitele   0000000000000000
                                            0000000000000000
                                             0000000001000101
                                              0000000001000101
                                               0000000000000000
                                                0000000000000000
                                                 0000000000000000
                                                  0000000000000000
                                                   0000000000000000
                                                    0000000000000000
                                                     0000000000000000
                                                      0000000000000000
Parciální součin podle bitu 15                         0000000000000000
Součet parciálních součinů          0000000000000000011010111101 = 1725(10) =
                                                                   = 3275(8)
```

Obr. P 5.1 Násobení 69 x 25 v dvojkové soustavě

Nechť registry A a D tvoří 32řádový akumulátor, kam se uschovávají součty parciálních součinů a výsledný součin. Nechť se násobenec uchovává v registru C a násobitel v DR. Násobící cyklus se řídí čítačem cyklů v registru X. Připočítávání parciálního součinu se uskutečňuje do registru A a potom se obsah (A,D) posunuje celý doprava.

P 5.4 I nadále uvažujeme o (strojovém) násobení kladných čísel a analyzujeme μ-program NAS. V řádcích 59-62 se přesunují operandy z HP (ze zásobníku v ní) do příslušných registrů. Řádky 63-69 a 82-84 zatím z našich úvah vyloučíme - realizují algoritmus pro záporné operandy.

Řádky 70-87 (zatím bez řádků 82-84) realizují tento cyklus:

1. Obsah C se podle hodnoty bitu DR(0) připočítává nebo nepřipočítává k obsahu A - řádky 70 - 72.
2. Obsah DR se posouvá o 1 bit vpravo (v každém cyklu se testuje nejnižší řád násobitele, tj. DR(0)) - řádky 73 a 74.
3. Obsah D se posouvá o 1 bit vpravo (posuv nižších 16 řádů akumulátoru) - řádek 75 a 76.
4. Testuje se hodnota bitu A(0) před posuvem A vpravo - řádek 77:
   - je-li A(0) = 0, je možno okamžitě posouvat vpravo - řádky 80 a 81,
   - je-li A(0) = 1, připočítává se do D hodnota MINUS neboli provede se D(15) := A(0) (přenos z D do A při posuvu vpravo) - řádky 78 a 79,

   a až potom se provedou řádky 80 a 81.
5. Od obsahu X se odečte jednička a testuje se, zda se provedlo 16 cyklů - řádky 85 a 87:
   - jestliže ne, skáče se na NCYK,
   - jestliže ano, NAS se ukončí uložením nižších 16 řádů součinu do zásobníku - řádky 88-91.

Z uvedeného algoritmu je třeba si uvědomit, že největší součin dvou čísel nesmí překročit hodnotu $2^{16}-1 = 65\,535_{(10)}$. Při násobení dvou čísel, u kterých by byl součin větší než $2^{16} - 1$, by vzniklo neodhalené přeplnění (obsah registru A je pro programátora ve strojovém jazyce nedostupný). Podle popisu CIKAP se do zásobníku vkládá vždy jen jedno slovo.

P 5.5 Doplňme nyní algoritmus o možnost násobení záporných čísel - násobení celých čísel v doplňkovém kódu. Znaménko součinu se vypočítává podle 4 známých pravidel:

1 (+) . (+) = (+) 3 (-) . (+) = (-)
2 (-) . (-) = (+) 4 (+) . (-) = (-)

Negováním znamének na levé straně 2., resp. 4. pravidla dostáváme správný výsledek a přitom převádíme případ 2, resp. 4 na případ 1, resp. 3:

2 (-) . (-) = (+) . (+) = (+) 4 (+).(-) = (-).(+) = (-)

Všimněme si přitom, že v 1. a 3. pravidle jsou kladní násobitelé. V našem algoritmu je to výhodné. Násobení se záporným násobitelem v doplňkovém kódu by znamenalo změnu algoritmu. Proto, jestliže μ-počítač "zjistí", že násobitel není kladný (řádek 63), automaticky jej převede na kladný (INVERZE + 1) - řádky 64 a 65. Podle pravidel 2´ a 4´ je tedy třeba změnit i znaménko násobence - řádky 66 a 67. Protože žádný výstup ze sčítačky není přímým vstupem do registru C, musíme obsah DR dočasně uschovat do D, provést změnu znaménka C a přes registr DR vrátit

do C; až potom se vrátí obsah D do DR - řádky 66-69.

V násobícím cyklu NCYK se potom postupuje

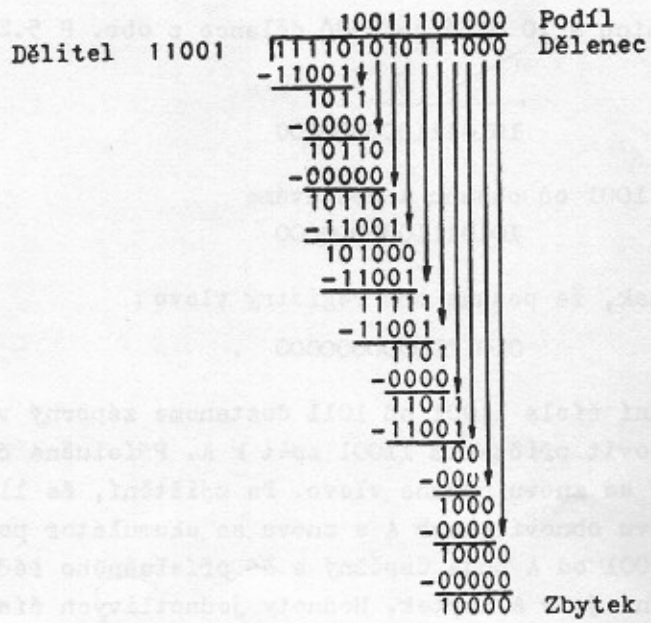
- buď podle 1. pravidla - kladné parciální součiny - výsledek kladný,
- nebo podle 2. pravidla - záporné parciální součiny - výsledek záporný.

Protože znaménko v C je vždy znaménkem parciálního součinu, je i znaménkem výsledného součinu. To je využito v řádcích 82-84. Při posuvu A doprava se do A(15) automaticky zapíše 0:

- má-li být parciální součet součinů kladný, pak je hodnota v A(15) již správná,
- má-li být parciální součet součinů záporný, pak se musí hodnota v A(15) změnit na 1_L - řádek 84; této operaci říkáme rozšiřování znaménka (doprava).

I n t e r p r e t a c e D I V

P 5.6 Vyjděme opět z ručního dělení, obr. P 5.2. Podobně jako se násobení skládá ze sčítání a posouvání, dělení se interpretuje jako odčítání a posouvání. Jak jsme v odstavci 2.5.3 konstatovali, hlavním problémem dělení je odhad parciálního podílu q_i. Dalším problémem je určení počtu "odčítání" dělitele od



Obr. P 5.2 Dělení 31416 : 25 v dvojkové soustavě

dělence. Např. při dělení čísla 1749598 číslem 1944 je třeba v prvním kroku odhadnout "kolikrát je 1944 obsaženo v 17495". Pro správný odhad používá člověk jisté počtářské zkušenosti, kterou počítač nemá.

Po odhadu první (nejvyšší) číslice podílu se součin číslice podílu a dělitele odčítá od dělence a tím se kontroluje správnost odhadnuté číslice. Jestliže

(nesprávně) odhadneme, že "1944 je v 17495 obsaženo" 9 krát a odečteme 9 x 1944 od 17495, bude rozdíl záporný. To nám signalizuje, že odhad (9) byl příliš vysoký. Dalším krokem je obnova (parciálního) dělence na původní hodnotu. Potom odhadujeme číslo podílu znovu a procedura kontroly se opakuje.

Ve dvojkové soustavě je uvedený proces poměrně jednoduchý, protože jediné možné odhady jsou 0 nebo 1; je-li 1 příliš velká, potom 0 je správná.

Dalším problémem je určit, odkud se má začít dělit; máme se ptát, zda 1944 je obsaženo v 1 nebo v 17 nebo v 174 nebo v 1749, 17495, 174959 nebo v 1749598? Správná odpověď je přirozeně 17495, ale jak to zjistí počítač? Nezbývá mu, než se postupně pokoušet odčítat dělitele od nejvyššího řádu, potom od nejvyšších dvou řádů, od tří nejvyšších řádů atd; do nejvyšších řádů podílu přitom vkládá 0, dokud není "pokusný" dělenec dostatečně velký.

Na obr. P 5.2 je uvedeno dělení čísla 31416 číslem 25. První pokusy v odčítání 11001 od 1, 11, 111 a 1111, pro které je dělitel velký, nejsou naznačeny; dělitel v nich není obsažen. Uvedeno je až odčítání dělitele od 11110. Tento pokus je úspěšný, protože 11001 je v 11110 obsaženo jednou se zbytkem 101. Dalším krokem je "připsání další číslice" (jedničky).

P 5.7 Ukážeme, jak se na počítači "připisuje další číslice dělence". Nechť registry A a D opět tvoří 32řádový akumulátor. V tomto okamžiku nechť je jejich stav:

5 vyšších a 10 nižších řádů dělence z obr. P 5.2

A D

0000000000011110 1010111000000000

Po odečtení čísla 11001 od obsahu A dostáváme

0000000000000101 1010111000000000

Počítač "připíše" tak, že posune oba registry vlevo:

0000000000001011 0101110000000000 .

Při pokusu o odčítání čísla 11001 od 1011 dostaneme záporný výsledek; obsah registru A musíme obnovit přičtením 11001 zpět k A. Příslušná číslice podílu bude tedy 0 a akumulátor se znovu posune vlevo. Po zjištění, že 11001 není obsaženo ani v 10110, se znovu obnoví obsah A a znovu se akumulátor posune vlevo. Další pokus o odčítání 11001 od A bude úspěšný a do příslušného řádu podílu se zapíše 1. Na konci dělení je v A zbytek. Hodnoty jednotlivých číslic podílu je možno ukládat do nejnižšího řádu registru D. Posunováním obsahu D vlevo se vždy uprázdní místo pro další číslici podílu. Konečný podíl bude v 16 řádech registru D.

P 5.8 Část μ-programu interpretující instrukci DIV (začíná na řádku 92) používá uvedený algoritmus. Jeho analýzu začneme opět předpokladem, že dělenec i dělitel jsou celá, kladná čísla. Až potom vysvětlíme, jak se řeší zbývající 3 případy se zápornými čísly celými (řádky 97-107, 126-128).

Mikroprogram začíná přečtením operandů ze zásobníku v HP cílového počíta-

če v pořadí dělitel (T) a dělenec (S); operace dělení se uskutečňuje podle výrazu S : T. Dělitel se ukládá do registru C (řádek 94) a dělenec do D (řádek 96). Čítač cyklů se nastaví na hodnotu 15 - řádek 92. O registru B zatím neuvažujeme.

Cyklus dělení začíná návěštím DCYK - řádek 108. Výchozí stav pro něj bude:

I D obsahuje (kladný) dělenec (DC) - řádek 96.

II DR obsahuje záporný dělitel (DL), který se bude používat k odčítání DL od parciálního dělence (zbytku) v A - řádky 115 a 116.

III C obsahuje kladný dělitel; bude se používat k obnově parciálního zbytku při neúspěšném "odhadu" příslušného bitu podílu - jde o přičítání C do A - řádky 121 a 122.

IV A obsahuje nulu - řádky 94 a 95.

Dělící cyklus DCYK začíná posunem levé části akumulátoru, tj. obsahu registru A vlevo; je-li obsah D(15) = 0, není třeba už nulu do A "připisovat" (skok na NEPIS); Jestliže D(15) = 1, jednička se "připíše" přičtením hodnoty 1 do A - řádky 111 a 112. Potom se posune obsah D vlevo - řádky 113 a 114. Tím je v D(15) připravena další číslice dělence, která se bude "připisovat" v následujícím cyklu a současně se řád D(0) uvolnil pro číslici podílu.

V řádku 116 se odčítá DL od obsahu A a v řádku 117 následuje test znaménka parciálního zbytku:

- jestliže A(15) = 0, do D(0) se zapíše jednička přičtením jedničky k D - řádky 118 a 119;
- jestliže A(15) = 1, tzn., že je parciální zbytek záporný, je třeba obsah A obnovit - řádky 121 a 122; v D(0) již nula je.

V řádcích 123 a 124 se odčítá jednička od obsahu X (čítač cyklů). V řádku 125 se testuje, zda se již provedlo 16 cyklů:

- jestliže ne, zahájí se znovu cyklus DCYK;
- jestliže ano, dělení je ukončeno a podíl je v D; obsah D se přes DR vloží do zásobníku v HP - řádky 129 - 131.

P 5.9 Probereme nyní, jak se interpretuje instrukce DIV strojového jazyka CIKAP pomocí mikroprogramu počítače HOMIP v případech, kdy jeden nebo oba operandy jsou záporné. Před začátkem dělícího cyklu se analyzují znaménka DC a DL. Na základě testu se do registru B uloží znaménko výsledku. Na začátku algoritmu se předpokládá, že DL a DC jsou kladné a dělící cyklus s těmito předpoklady pracuje.

Zopakujeme si všeobecná pravidla práce se znaménky při dělení:

1 (+) : (+) = (+)
2 (-) : (+) = (-)
3 (+) : (-) = (-)
4 (-) : (-) = (+)

Probereme nejdříve metodu určení znaménka podílu (výsledku). V řádku 94 se do B uloží skutečný DL (z něho nás zajímá pouze znaménko). V řádku 97 se analyzuje znaménko DC:

- je-li DC kladný, skáče se na řádek 103 (DOKLAD) a s obsahem B se již nic neprovádí; tzn. je takové, jako je znaménko DL (pravidla 1 a 2);

- je-li DC záporný, vytvoří se (mimo jiné) doplněk obsahu B neboli změní se znaménko v B - řádky 100-102. Jestliže tedy byl v B kladný dělitel (a DC je záporný), výsledek bude kladný (č. pravidlo).

Tím máme určeno znaménko výsledku. Obsah (znaménko) registru B se použije až po ukončení dělícího cyklu - řádek 125 a případně 126-128.

Jak jsme již uvedli, dělící cyklus pracuje s předpokladem, že DC a DL jsou kladné, bez ohledu na jejich skutečná znaménka. To znamená, že pro dosažení stavů I a IV z odstavce P 5.8 je třeba:

I Je-li původní DC v registru D

- kladný (test na řádku 97), obsah D se nemění a zůstává kladný,
- záporný, vytvoří se v D jeho doplněk (řádky 98 a 99), po provedení řádku 100 bude DC v D kladný.

II a III Na začátku se předpokládá, že původní DL je v registru C kladný, a proto se v řádcích 103 a 104 vytváří jeho doplněk, který se uchovává v DR. Jestliže se testem na řádku 105 dodatečně zjistí, že je původní DL záporný, přesune se doplněk z D (dělitel "změněný na kladný") do registru C (řádek 106) a v DR se vytvoří další doplněk - řádky 106 a 107. Ve všech čtyřech případech se tedy nakonec dosáhne požadovaný stav: kladný DL v registru C a záporný dělitel v registru DR.

IV Obsah registru A se nuluje vždy na řádcích 94 a 95.

Po 16. opakování dělícího cyklu (řádky 108 a 125) se podíl upraví podle znaménka v B(15).

Zbytek v registru A je pro programátora v strojovém jazyce našeho hypotetického cílového k-počítače nepřístupný, a tedy ztracený.

- - x - -

U p o z o r n ě n í : Uvedené algoritmy interpretace strojových instrukcí NAS a DIV pomocí μ-instrukcí OTEVRI_RB a TEST_A_PODMINENY_SKOK není možno považovat za jediné ani za nejefektivnější. Jsou však snadno pochopitelné. Algoritmus DIV by bylo možno vylepšit např. tak, že by se, v případě záporného parciálního zbytku, místo obnovy parciálního dělence připočítáním dělitele v následujícím průchodu dělícím cyklem dělitel neodečítal, ale přičítal.

Poznamenejme ještě, že HOMIP je navržen jako relativně jednoduché číslicové zařízení, určené k interpretaci různých strojových jazyků, např. pro experimentální účely při navrhování počítačů. Pokud jde o implementaci strojového jazyka sériově vyráběných počítačů, bylo by třeba brát mnohem větší ohled např. na rychlost a cenu zařízení, a řešení by byla odlišná. Příklad uvedený v této kapitole slouží tedy jako úvod do problematiky.

4. ÚROVEŇ POČÍTAČE S OPERAČNÍM SYSTÉMEM

4.1 ÚVOD

4.1.1 V soudobých výpočetních systémech je pouze relativně malá část programů používána ve strojovém jazyce. V kapitole 1 jsme tuto vrstvu programů nazvali k-programy. Tyto programy přejímají některé úkoly při obsluze počítače a bývají nazývány operační systém, exekutiva, monitor, supervizor apod. Uživateli se konvenční počítač s těmito programy jeví opět jako virtuální počítač. Podle číslování úrovní z kapitoly 1 je to virtuální počítač úrovně 3 a budeme jej nazývat o-počítač, obslužný počítač, operační počítač nebo rozšířený počítač (žádný stručný termín není ustálen).

4.1.2 Jazykem o-počítače (o-jazykem) jsme se orientačně zabývali již v odstavci 1.2.7. Jak je tam uvedeno, většina instrukcí strojového jazyka je uživateli o-počítače přístupná, tj. je součástí jazyka o-počítače.

Jazyk o-počítače je tedy rozšířením (podmnožiny) strojového jazyka. K-program přitom slouží k interpretaci právě těch příkazů o-jazyka, které nepatří zároveň do k-jazyka.

4.1.3 V kapitole 1 jsme se seznámili s jednotlivými jazykovými úrovněmi počítačového systému a v dalších kapitolách této učebnice se hovoří o způsobech realizace jednotlivých jazyků pomocí jazyků odpovídajících nižší úrovni. Například v kapitole 4 jsme se seznámili s jedním způsobem realizace jednoho k-jazyka pomocí jednoho μ-jazyka. V této kapitole budeme hovořit o způsobech realizace o-jazyků pomocí k-jazyků.

Namísto termínu realizace někdy používáme termín implementace. Často budeme hovořit nikoli o implementaci celého o-jazyka, ale jen o implementaci některých jeho operací (například operací s hlavní pamětí).

.1.4 Počítač s operačním systémem (o-počítač) poskytuje svému uživateli dokonalejší služby než "počítač bez operačního systému" (k-počítač). Pokud bychom nechtěli operační systém použít, museli bychom mnohé jeho funkce ve svém programu naprogramovat sami. Operační systém pomáhá uživateli řešit následující typické úkoly:

1. ovládání vnějších zařízení,
2. organizaci dat na vnějších mediích,
3. efektivní využívání počítače,
4. zadávání úkolů počítači.

Základními poznatky o řešení těchto úkolů se budeme v této kapitole zabývat.

4.1.5 Operační systém patří k tzv. systémovému programovému vybavení počítače. Tyto programy vytváří dodavatel počítače (patří sem též překladače, slu-

žební programy apod.). Bez systémového programového vybavení není počítačový systém kompletní a jeho používání je pro uživatele nepřiměřeně nákladné.

Vytvoření kvalitního systémového programového vybavení dodavatelem zamezuje duplicitě programátorských prací u jednotlivých uživatelů počítačového systému.

4.1.6 Termíny operační systém, exekutiva (prováděcí program), monitor (sledovací program), supervizor (dohlížecí program) a podobné jsou jednotlivými firmami dodávajícími počítače používány zcela volně pro různě dokonalé a různě specializované programové vybavení. Samotný termín tedy neříká nic více, než že se jedná o programové vybavení rozšiřující funkce konvenčního počítače a napomáhající obsluze počítače. Většinou se jedná o programy pro implementaci o-jazyka, některé firmy však za součást operačního systému považují i překladače, služební programy apod.

P o z n á m k a : Vzhledem ke konkurenčnímu boji nemají firmy na sjednocení terminologie zájem; naopak vymýšlení nových názvů pro staré věci má někdy vzbudit u zákazníka dojem, že kupuje inovované zařízení, nebo ztížit porovnávání zařízení.

4.1.7 Vzhledem k výše uvedené situaci a velkému množství počítačů a operačních systémů je nereálné snažit se o jejich podrobnější porovnávání nebo vypracování přehledu. Rovněž není vhodné na začátku studia této problematiky vycházet z některého firemního operačního systému, neboť to zpravidla vede ke zkreslenému pohledu a k "firemní zaujatosti".

V této kapitole se proto zaměřujeme na všeobecný úvodní výklad, který by měl usnadnit pozdější studium a používání o-počítače. Všeobecný výklad přitom zároveň napomáhá k pochopení, čím jsou jednotlivé firemní systémy zvláštní.

P o z n á m k a : Z praktických příčin je třeba výklad chápat jako přípravu na bližší seznámení s konkrétním firemním operačním systémem. Méně se uplatňuje hledisko, jak by operační systém měl (podle současných znalostí) vypadat.

4.1.8 V "dávné" historii počítačů (asi v letech 1946 až 1953) nebyly operační systémy používány. O elektronických počítačích vyrobených v tomto období hovoříme jako o počítačích prvé generace. Programátor na takovém počítači také operoval, tj. zaváděl program do paměti, spouštěl a zastavoval počítač, zobrazoval si obsahy různých registrů a paměťových míst (při ladění programů) apod. Taková práce byla velice zajímavá, ale časté opakování uvedených úkonů bylo nepříjemné. U drahých počítačů (levné tehdy ani neexistovaly) jsou ovšem nežádoucí ztráty času, způsobené pomalostí a chybováním člověka při těchto manipulacích. Základním úkolem prvotních operačních systémů byla automatizace rutinních prací při obsluze počítače a při ladění programů. Konvenční počítače od druhé generace počínaje jsou většinou navrhovány tak, že jsou vhodné pro určité druhy operačních systémů (podkapitola 5.2), zatímco jiné druhy není možné nebo účelné pro ně vytvářet.

4.1.9 Rozdělování počítačů do jednotlivých generací má pouze orientační význam, neexistuje všeobecně přijatá definice generací. Výrobci počítačů zpravidla "upravují" výklad tohoto pojmu ve svůj prospěch. Pokud je třeba o generacích hovořit, je nejlépe vycházet z porovnání se špičkovými výrobky vzniklými v určitém ob-

dobí. Porovnání by se nemělo omezovat na dílčí aspekty (jako je technologie výroby obvodů, míra integrace obvodů, operační systém, vyšší programovací jazyky apod.), neboť uživatele zajímá počítač především jako celek.

P ř í k l a d

V tabulce 5.1.1 je uvedeno jedno z možných rozdělení počítačů na generace.

Orientační tabulka generací počítačů — Tabulka 5.1.1

Generace	1	2	3	4
Přibližné období vzniku	1946 - 1953	1954 - 1962	1963 - 1970	1971 - 1978
Stavební prvky logických obvodů	elektronky	tranzistory	integrované obvody	obvody střední a velké integrace
Průměrná doba operací a paměťového cyklu [s]	10^{-3} - 10^{-4}	10^{-5}	10^{-6}	10^{-7} - 10^{-8}
Programové vybavení	sestavovací program (assembler); knihovny programů	kompilátory: FORTRAN, COBOL, ALGOL; monoprogramní operační systém	kompilátory dalších jazyků; multiprogramní operační systém	rozšiřitelné mikroprogramy, rozšiřitelné operační systémy, specializované programové vybavení
Přístup k počítači	místní periférie*	vzdálené periférie*	účastnický režim	v rámci počítačové sítě

* Vnější zařízení (vnější jednotky) nazýváme též stručně periférie.

4.2 ORGANIZACE VYUŽITÍ POČÍTAČE

4.2.1 Počítačový systém může být určen k řešení různých úkolů a jeho vlastnosti jsou tomuto účelu přizpůsobeny. Organizace využití počítače je dosti odlišná např. u počítače vestavěného ve výrobní lince a u počítače pracujícího souběžně pro mnoho uživatelů. Jednotlivé druhy operačních systémů jsou vhodné pro určitou organizaci využití počítače. V této stati uvedenou otázku vysvětlíme podrobněji.

4.2.2 U některých operačních systémů je zvykem o-program nazývat práce (anglicky - job) a posloupnost o-programů dávka (anglicky - batch). o-programy některých počítačů mohou předepsat souběžné provádění určených částí o-programu, na-

zývaných úloha (task). Tato možnost je nazývána "multitasking" (víceúlohová práce). Nedává-li operační systém tuto možnost, lze termíny "práce" a "úloha" ztotožnit.

P o z n á m k a : Provádění jedné úlohy je tedy proces úrovně 3. Provádění jedné práce je jeden či více kooperujících procesů úrovně 3. Kromě možnosti víceúlohové práce jednoho o-počítače může operační systém implementovat více o-počítačů na jednom k-počítači. Procesy z různých o-počítačů však jsou nezávislé (nekooperují).

4.2.3 Je-li počítač použit jako vestavěný systém, je často používán k jedinému účelu (tj. není sdílen). Počet takových počítačů v poslední době rychle roste (zejména díky zlevnění výroby jednodušších počítačů).

Pro starší nebo méně výkonné počítače jsou často užívány tzv. jednoprogramové (monoprogramní) operační systémy. Tyto systémy umožňují postupné (sériové) sdílení počítače. To znamená, že jsou zpracovávány různé o-programy (různých uživatelů) postupně jeden za druhým a počítač tedy neslouží jedinému účelu.

Operační systém však může dovolit i souběžné (paralelní) sdílení počítače. To znamená, že v daném okamžiku může být rozpracován více než jeden o-program. Jinak řečeno: lze spustit další o-programy dříve, než byl předchozí o-program ukončen. Takové operační systémy nazýváme víceprogramové (multiprogramní). Souběžné sdílení počítače má odlišné formy zejména v závislosti na lhůtách, ve kterých je nutné jednotlivé o-programy či části o-programů zpracovat.

1. Multiprogramní operační systémy pro zpracování dávek. Hlavním kritériem pro činnost systému je co nejlepší využití počítače; lhůty pro provedení prací jsou tomuto požadavku pokud možno přizpůsobeny.
2. Účastnické operační systémy. Hlavním kritériem pro činnost systému je plnění požadavků jednotlivých účastníků (například uživatelů pracujících u terminálů); lhůty pro zpracování požadavků účastníků mají pokud možno odpovídat zvyklostem při konverzaci.
3. Operační systémy pro řízení či sledování externích procesů (řízení "v reálném čase"). Hlavním kritériem pro činnost systému je dodržení určitých stanovených lhůt pro zpracování jednotlivých údajů (externí proces generující a přijímající údaje probíhá určitou rychlostí, kterou počítač zpravidla neovlivňuje).

4.2.4 Nepřímý přístup uživatele k počítači je zpravidla spojen s používáním operačního systému pro zpracování dávek. Je-li tento operační systém monoprogramní, provádí obvykle obsluha počítače seřazení práce do dávek tak, aby ztrátové časy při zpracování (způsobené např. nasazováním médií potřebných pro zpracování úloh) byly minimální. U multiprogramních operačních systémů pro zpracování dávek toto plánování pořadí zpracování a rozhodování o souběžnosti zpracování zpravidla provádí operační systém. Postačuje tedy, aby uživatel práci předal obsluze na místě nazývaném "příjem zakázek" a později si ji i s výsledky vyzvedl na místě nazývaném "výdej zakázek". Obsluha zajišťuje, aby zakázka (práce nebo skupina prací) uživatele byla za určitou dobu zpracována. Tuto lhůtu nazýváme doba obrátky a bývá dlouhá několik hodin až několik dnů.

4.2.5 O přímém přístupu uživatele k počítači hovoříme v případě, že uživatel není od počítače oddělen mezičlánkem, tj. obsluhou počítače. Pokud je při přímém přístupu použito postupné sdílení počítače, má každý uživatel k dispozici po stanovenou dobu celý počítač. Počítač pak uživatel ovládá sám nebo využívá pomoci operátorů. Počítač má zpravidla operační systém, který takovou práci usnadňuje, tj. uživatel komunikuje s o-počítačem. Sdílení počítače je řízeno administrativně provozovatelem počítače (plánování a přidělování "strojového času") nebo dohodou uživatelů. Tato organizace využívání počítače je vhodná zejména pro levnější počítače a pro laboratorní či experimentální účely.

4.2.6 Při přímém přístupu a souběžném sdílení počítače je přístupových zařízení připojeno k počítači více a jejich uživatelé mohou pracovat současně. Každý z těchto uživatelů má k dispozici "svůj" o-počítač; je to ovšem virtuální (zdánlivý) počítač. Úkolem operačního systému v tomto případě je tedy implementovat ne jeden, ale několik o-počítačů. Tyto o-počítače mohou (ale také nemusí) být úplně stejné a pracují paralelně. Souběžnost jejich činnosti je ovšem pouze jevovou stránkou věci, projevující se při pozorování systému na úrovni 3. Při podrobnějším zkoumání (při pozorování činnosti na úrovni 2, tj. při pozorování na úrovni konvenčního počítače) se pochopitelně ukáže, že počítač je pouze jeden a pracuje střídavě na úlohách jednotlivých uživatelů. Jeden uživatel tedy nemá v daném okamžiku k dispozici celý počítačový systém, ale pouze tu jeho část, kterou mu operační systém přidělí k využití.

4.2.7 Sdílení velkého počítače tedy uživatelům umožňuje zpracovat úlohy, které jsou pro menší počítače příliš rozsáhlé či složité. Velký, a tedy drahý počítač však musí být dobře vytížen, a proto je obvykle sdílen souběžně.

Při organizaci souběžného sdílení vznikají určité ztrátové časy. Jde o dobu, kdy operační systém neslouží určitému o-programu, ale řeší programy sdílení (třeba rozvrhování, který o-program kdy spustit). Tato doba bývá nazývána režie operačního systému.

Při účtování strojového času hradí uživatel sdíleného počítače jen částečnou cenu strojového času, vypočtenou obvykle podle doby, kdy jeho program skutečně "běžel", a podle prostředků, které mu byly přitom k dispozici (části hlavní paměti, vnější zařízení apod.). Navíc je připočítávána alikvotní část režie. To lze chápat jako daň uživatele za to, že může velký počítač použít a přitom sám neplatí celý strojový čas.

4.2.8 Jednou z forem přímého přístupu k počítači je přímé zadávání prací. Uživatel má přístup k terminálu, do kterého vloží svou zakázku. Výsledky má po době obrátky k dispozici ve výdeji zakázek. Oproti nepřímému přístupu je tedy uživatel méně závislý na práci obsluhy počítače.

Terminál pro přímé zadávání úloh může být místní nebo vzdálený. V druhém případě hovoříme o vzdáleném zadávání prací (remote job entry). Výsledky si ovšem uživatel musí vyzvednout ve výdeji zakázek nebo mu musí být poslány. Většinou je vhodnější vzdálený terminál a operační systém řešit tak, aby výsledky prací vystupovaly na tomto terminálu. To se nazývá vzdálené provádění prací (někdy se hovoří i v tomto případě jen o "vzdáleném zadávání prací").

Pro tyto způsoby je charakteristické, že práce je zadána celá najednou a že výsledek také obdržíme celý najednou. Operační systém je orientován na zpracování dávek obdobně jako při nepřímém přístupu. Obvykle je možné provádět zadávání prací souběžně z různých terminálů a přitom ještě souběžně probíhá zpracování jednoho či více o-programů.

4.2.9 Jinou formou přímého přístupu uživatele k počítači je konverzační (nebo též interakční) přístup. Název zdůrazňuje skutečnost, že řešení problému může mít formu postupného vyměňování zpráv mezi uživatelem a výpočetním systémem, tedy formu konverzace. Uživatel dostává výsledky vždy na terminálu.

4.3 JAZYK O-POČÍTAČE A JEHO IMPLEMENTACE

4.3.1 o-program (práce), resp. jeho úlohy (odst. 4.2.2) jsou složeny z kroků (job step).

P ř í k l a d

```
# JOB    JMENO = 'KARLICEK' ,  KONTO = 134;
  }
  } kroky
  }
# EOJ
```

Příklad ukazuje příkaz začátku a konce práce, mezi nimiž jsou jednotlivé kroky. Znak # v tomto příkladu odlišuje příkazy jazyka řízení prací (odst. 1.2.7 a)) od jiných údajů.

Při programování kroku je třeba zejména určit:

- výkonný program, který má být proveden,
- parametry, s nimiž má být proveden,
- vstupní data,
- výstupní data (místa pro jejich uložení).

P ř í k l a d

```
# FETCH            'FORTRAN VERSION H', TAPE = 2 ;  }
# FILE       IN1 = 'ZDROJOVÝ PROGRAM 5';           }
# FILE      OUT1 =  STANDARD ;                     } krok
# PARAM     LIST =  NO;                            }
# RUN   PRIORITY =  10;                            }
```

Jak je vidět ve výše uvedeném příkladě, má činnost o-počítače řízená o-jazykem dvě složky:

- provádění řídících příkazů,
- provádění výkonných programů.

Jedním krokem rozumíme tedy volání a provedení výkonného programu.

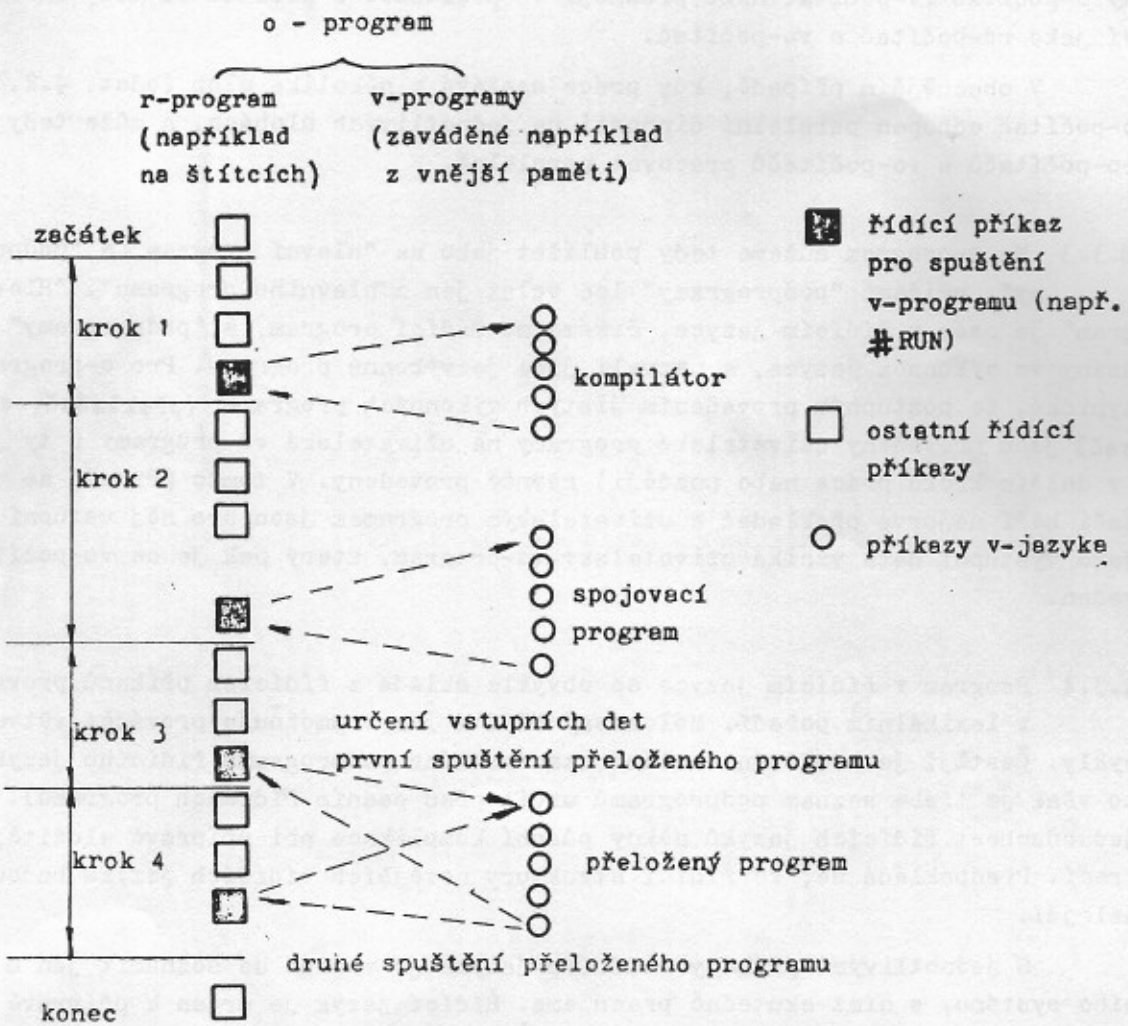
Výkonné programy považujeme za část o-programu. Většinou platí, že příkazy o-jazyka (odst. 1.2.7), které lze používat ve výkonných programech (1.2.7.b), 1.2.7.c)), nelze používat jako řídící příkazy (1.2.7.a)), a naopak řídící příkazy nelze používat ve výkonných programech (obr. 5.3.1).

Proto je účelné o-jazyk rozdělit na dvě disjunktní části:

- řídící jazyk (r-jazyk, přesněji ro-jazyk),
- výkonný jazyk (v-jazyk, přesněji vo-jazyk).

Program v řídícím jazyku nazýváme též řídící program (r-program, přesněji ro-program), obdobně jako program ve výkonném jazyce nazýváme výkonný program (v-program, přesněji vo-program).

Řídící jazyk je také často nazýván jazyk řízení prací (Job Control Language, JCL). V řídícím jazyce je vyjádřeno, které výkonné programy mají být spuštěny, s jakými daty, v jakém pořadí a za jakých podmínek.



r-program a v-programy tvoří logický celek zvaný o-program, fyzicky však nemusí být na stejném médiu.

Obr. 5.3.1 Příklad struktury o-programu

Výkonný jazyk je také často nazýván jazyk prováděného programu (run time language, doslova jazyk doby běhu), neboť programy psané ve vyšších programových jazycích nebo v asembleru jsou většinou překládány do tohoto jazyka a v něm pak "běží", tj. jsou interpretovány (odst. 1.2.12).

4.3.2 Činnost o-počítače má tedy dvě složky:

- řízení prací (činnost dle r-programu),
- provádění výkonných programů (činnost dle v-programu).

Tyto dvě činnosti jsou vzájemně výlučné. Činnost o-počítače při provádění práce má formu cyklického střídání uvedených činností. To můžeme chápat tak, že nejprve zpracuje (virtuální) počítač řídící práci a pak pracuje (virtuální) počítač vykonávající připravenou práci.

Opět můžeme počítač řídící práce nazvat řídící o-počítač (r-počítač nebo přesněji ro-počítač) a počítač vykonávající připravené práce můžeme nazvat výkonný o-počítač (v-počítač nebo přesněji vo-počítač). o-počítač se tedy střídavě jeví jako ro-počítač a vo-počítač.

V obecnějším případě, kdy práce sestává z několika úloh (odst. 4.2.2), je o-počítač schopen paralelní činnosti na jednotlivých úlohách, a může tedy několik ro-počítačů a vo-počítačů pracovat paralelně.

4.3.3 Na o-program můžeme tedy pohlížet jako na "hlavní program" s "podprogramy", přičemž "podprogramy" lze volat jen z "hlavního programu". "Hlavní program" je psán v řídícím jazyce, říkáme mu řídící program, a "podprogramy" jsou psány ve výkonném jazyce, a nazvali jsme je výkonné programy. Pro o-programy je typické, že postupným provedením jistých výkonných programů (překladač, spojovač) jsou převáděny uživatelské programy na uživatelské vo-programy a ty jsou (v dalším kroku práce nebo později) rovněž provedeny. V tomto případě na vo-počítači běží nejprve překladač a uživatelským programem jsou pro něj vstupní data; jako výstupní data vzniká uživatelský vo-program, který pak je na vo-počítači proveden.

4.3.4 Program v řídícím jazyce se obvykle skládá z řídících příkazů prováděných v lexikálním pořadí. Málokterý řídící jazyk umožňuje provádět větvení a cykly. Častěji je používáno volání standardních podprogramů řídícího jazyka (často však je třeba seznam podprogramů určit před psaním řídících programů). Tato jednoduchost řídících jazyků někdy působí komplikace při přípravě složitějších prací. Předpokládá se, že řídící struktury novějších řídících jazyků budou dokonalejší.

S jednotlivými příkazy řídícího jazyka je vhodné se seznámit jen u operačního systému, s nímž skutečně pracujeme. Řídící jazyk je určen k přípravě činnosti pro výkonný počítač a pro pochopení jeho sémantiky je tedy třeba znát možnosti výkonného počítače. To je u některých počítačů dost složité, a právě to činí při studiu řídících jazyků obtíže.

4.3.5 Při povrchním pohledu lze výkonný program zaměnit s k-programem, neboť oba mají formu posloupnosti instrukcí strojového jazyka a jsou vyjádřeny v binárním tvaru (instrukce jsou posloupnosti nul a jedniček). Základní odlišnost je však v tom, že některé instrukce strojového jazyka nepatří do výkonného jazyka (tzv. privilegované instrukce) - odstavec 2.6.3.

Další odlišnost je sémantická. Instrukce nazývané příkaz přerušení (odst. 1.2.7), resp. programové přerušení (pro EC instrukce SVC a MC, pro SM instrukce TRAP a EMT; přílohy 1 a 2), z hlediska činnosti k-počítače způsobují akci zvanou přerušení. Použití téže instrukce ve výkonném programu má však jiný (bohatší) význam: dojde totiž k provedení k-programu určeného zmíněným přerušením (který obvykle vykonává nějakou řídící činnost) a k návratu do v-programu.

Výkonný jazyk má tedy dva druhy příkazů:

a) příkazy pro volání k-programů (často jsou nazývány "volání supervizoru" nebo "volání monitoru");

b) podmnožinu instrukcí strojového jazyka.

Příkazy pro volání k-programů slouží zejména k řízení činnosti operačního systému a k využívání jeho služeb.

4.3.6 Způsob implementace výkonného jazyka bývá na rozdíl od překladu či interpretace nazýván částečná interpretace (obr. 5.3.2). Tím se rozumí, že jsou interpretovány jen některé příkazy (bod 4.3.5a)), zatímco ostatní (bod 4.3.5b)) lze přímo provést. Není totiž důvod interpretovat příkazy strojového jazyka pomocí téhož jazyka.

Částečná interpretace je velmi podobná volání podprogramů. K-programy ovšem nejsou z v-programu volány instrukcí "volání podprogramů", ale "příkazem pro volání k-programu", a nejsou pochopitelně vyjádřeny ve v-jazyce, ale v k-jazyce. S analogickou situací jsme se již setkali v odst. 4.3.3, kde jsme se zabývali voláním výkonného programu z řídícího programu.

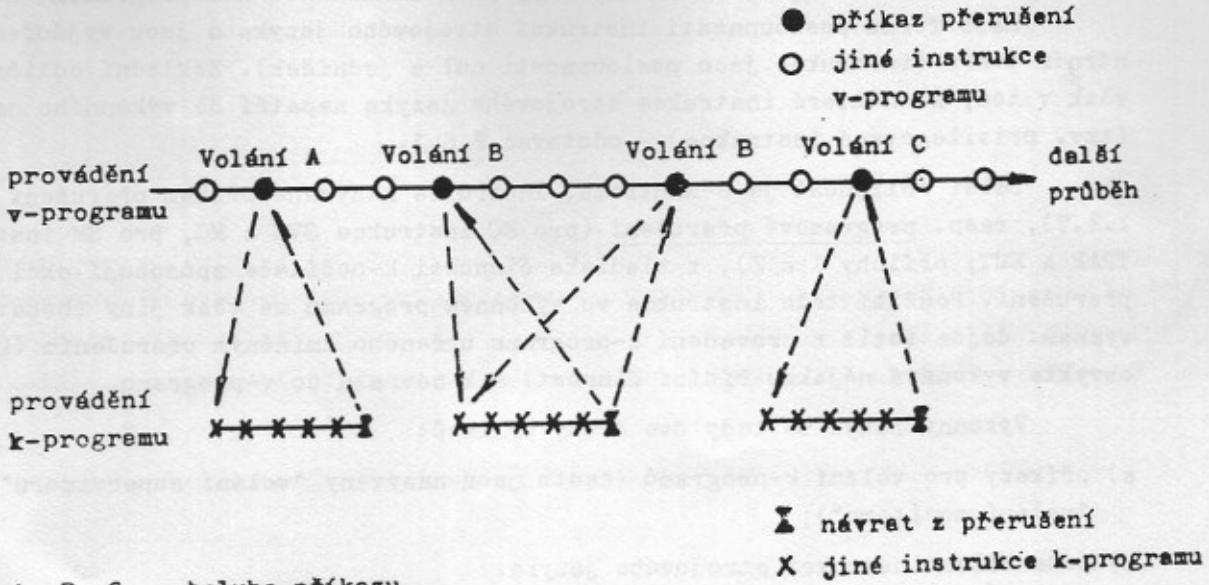
Odlišnost v-jazyka od k-jazyka se může zdát malá, má však své důsledky například při detekci chyb (co je chybné ve výkonném jazyce, nemusí být chybné ve strojovém jazyce). Při řešení návaznosti obou jazyků musí být též stanoven způsob předávání parametrů a další konvence.

4.3.7 V odstavci 1.2.5 jsme uvedli, že o-jazyk je interpretován k-programem. Nyní upřesníme, jak je to obvykle provedeno.

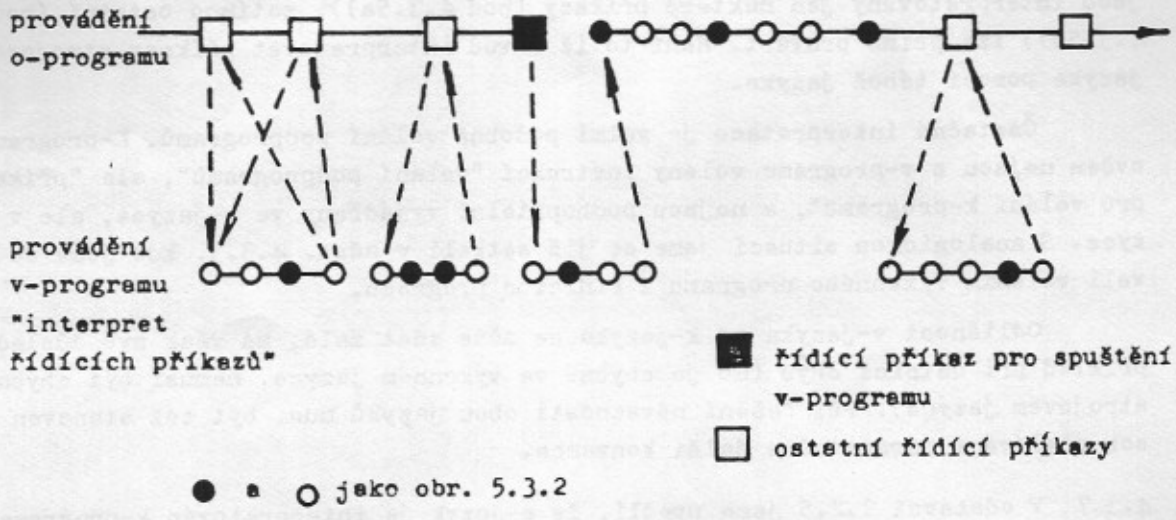
Příkazy výkonného jazyka jsou částečně interpretovány k-programem. Pro příkazy řídícího jazyka by bylo možné napsat interpret ve strojovém jazyce, je to však zbytečně programátorsky pracné. Výhodnější je interpret řídícího jazyka napsat ve výkonném jazyce. Předpokládejme tedy, že je to tak provedeno (obr. 5.3.3).

Tím jsme vyjádřili způsob implementace obou složek o-jazyka. Nyní můžeme způsob implementace o-jazyka vyjádřit souhrnně:

1. o-jazyk je částečně interpretován ve výkonném jazyce (tj. příkazy řídícího jazyka jsou interpretovány ve výkonném jazyce a příkazy výkonného jazyka není třeba ve výkonném jazyce interpretovat);



Obr. 5.3.2 Znázornění částečné interpretace v-programu



Obr. 5.3.3 Znázornění částečné interpretace o-programu

2. výkonný jazyk je částečně interpretován ve strojovém jazyce (odst. 4.3.3 a 4.3.6).

Celkově je tedy o-jazyk skutečně implementován ve strojovém jazyce, způsob implementace lze nazvat "dvojúrovňová hierarchie částečných interpretací", což je obecně vzato opět částečná interpretace.

P o z n á m k a : Výkonný program, interpretující příkazy řídícího jazyka, je nazýván též interpret řídícího jazyka nebo interpret JCL. Lze též říci, že interpret JCL je "systémový" výkonný program.

4.4 POŽADAVKY NA STROJOVÝ JAZYK

4.4.1 Na některých počítačích je volání supervizoru (odst. 4.3.5 a)) prováděno stejně jako volání podprogramů a ve výkonném jazyce lze používat všechny instrukce strojového jazyka. To však vede k různým problémům (např. nelze řešit souběžné sdílení počítače, jsou menší možnosti při detekci chyb apod.). Toto řešení je přijatelné pouze ve zvláštních případech (malé počítače, nesdílené počítače apod.). V ostatních případech je třeba rozlišovat, kdy počítač interpretuje strojový jazyk a kdy výkonný jazyk. Z tohoto důvodu jsou rozlišovány takzvané privilegované a neprivilegované instrukce strojového jazyka (odst. 4.6.1 a 4.6.3).

4.4.2 Ve výkonném jazyce lze používat neprivilegované instrukce a nelze používat privilegované instrukce (použití takové instrukce je třeba hlásit jako chybu). Při interpretaci instrukcí strojového jazyka je proto (zpravidla jedním bitem ve stavovém slově procesoru) rozlišována činnost procesoru:

a) v režimu "uživatel" (užívání o-počítače),
b) v režimu "systém" (užívání k-počítače, tedy činnost operačního systému).

Používají se i jiné názvy, např. a) režim "program" a b) režim "supervizor". Případ "a" znamená, že procesor považuje privilegované instrukce za chybné; v případě "b" provádí všechny instrukce strojového jazyka. Název "privilegovaná instrukce" vyjadřuje, že privilegium k jejímu provedení má pouze operační systém (přesněji privilegovaná instrukce smí být v k-programu a nesmí být v o-programu). To dovoluje vyhradit některé činnosti (třeba ovládání vnějších zařízení) pouze operačnímu systému a zabránit nežádoucím intervencím jiných programů (ať již úmyslným, či neúmyslným) do těchto činností. Pro realizaci uvedených činností pak o-programy využívají služeb operačního systému (odst. 4.3.2 a)).

4.4.3 Při interpretaci o-programu pracuje procesor v režimu "uživatel", tedy provádí jen neprivilegované instrukce. Připomeňme odstavec 4.3.3, kde jsme uvedli dva druhy příkazů výkonného jazyka. Nyní můžeme upřesnit:

a) příkazy, které mají být interpretovány k-programem, jsou programovány pomocí neprivilegovaných instrukcí typu "příkaz přerušení",
b) "podmnožinou instrukcí strojového jazyka" jsou míněny zbývající neprivilegované instrukce.

Instrukce "příkaz přerušení" je velmi podobná instrukci skoku do podprogramu. Odlišnost je v tom, že instrukce "příkaz přerušení" může měnit režim činnosti procesoru, např. z režimu "uživatel" do režimu "systém". Kromě neprivilegovaných instrukcí "příkaz přerušení" je na některých počítačích k dispozici zvláštní privilegovaná instrukce "návrat z přerušení". Od instrukce návrat z programu se odlišuje tím, že obnovuje režim činnosti procesoru předcházející odpovídajícímu "příkazu přerušení".

4.4.4 Provedení instrukce "příkaz přerušení" je zvláštním případem akce nazývané přerušení. Režim činnosti procesoru je totiž zpravidla měněn zapsáním nového stavového slova procesoru do registru stavového slova procesoru (PSW).

Přerušením tedy rozumím dující činnost k-počítače:

a) uschování návratové adresy;

b) uschovámí stavového slova procesoru;

c) uschování některých registrů;

d) zapsání údajů o příčině přerušení (například do určeného registru);

e) zapsání nového stavového slova procesoru;

f) zaspání nové adresy do čítače instrukcí.

Tyto činnosti jsou typické pro provádění přerušení; na jednotlivých počítačích však existují různé odlišnosti (například některé počítače při přerušení neukládají žádné registry).

Přerušení tedy má za následek, že procesor k-počítače přestane provádět instrukce v "normálním" pořadí určeném k-programem a provede odskok na novou adresu (bod "f"), určenou podle příčiny přerušení. Na této adrese začíná obslužný program (handler), nazývaný též "posloupnost (instrukcí pro) zpracování přerušení".

Při přerušení jsou uschovány potřebné údaje (body "a,b,c") tak, aby bylo možné v obslužném programu uschovat případné další potřebné údaje a naprogramovat návrat a pokračování v původní činnosti. K tomu slouží instrukce umožňující návrat z přerušení, která provádí:

g) obnovení původního obsahu registrů (uložených dle bodu "c"),

h) obnovení původního stavového slova procesoru (uloženého dle bodu "b");

i) obnovení původní adresy v čítači instrukcí (tj. zapsání návratové adresy, uložené dle bodu "a", do čítače instrukcí).

Opět se jedná o typické činnosti. Na některých počítačích je k provedení těchto akcí třeba více instrukcí, jinde je čítač instrukcí součástí stavového slova procesoru PSW atd. Pro příklad obr. 5.4.1.

4.4.5 Přerušení, obslužný program a návrat z přerušení dohromady zpravidla mají zachovat stav procesu "provádění programu". Programem zde míníme libovolný program, který byl přerušen. Pojem stav procesu byl vysvětlen v odstavci 1.1.1. Při přerušení a jeho obsluze je obvykle ukládána a obnovována jen ta část stavového vektoru procesu, kterou by provedení obslužného programu změnilo.

4.4.6 Příčiny přerušení jsou různé podle typu počítače; probereme opět pouze typické případy.

Zatím jsme uvedli, že přerušení je důsledkem instrukce "příkaz přerušení". V tom případě jde o programové přerušení a příkaz přerušení je jeho příčinou.

Podobným případem je přerušení, jehož příčinou je výjimečná situace při provádění nějaké instrukce. Např. při provádění instrukce sčítání může dojít k přeplnění. Na některých počítačích je třeba vznik této výjimečné situace testovat programem (instrukcí podmíněného skoku). To ovšem program prodlužuje. Výhodnější je, jestliže počítač v případě přeplnění provede přerušení. Tím je zavolán obslužný program, který provede např. vypsání chybové zprávy a návrat nebo ukončení programu anebo jinou činnost. Příčiny přerušení tohoto druhu budeme nazý-