3. ÚROVEŇ MIKROPOČÍTAČE

V této kapitole se budeme zabývat tím, jak se realizuje úroveň virtuálního k-počítače pomocí úrovně základního μ-počítače. K tomuto účelu doporučujeme zopakování podkapitol 1.1 a 1.2 o struktuře víceúrovňového počítačového systému.

Naší úlohou nechť je realizace nějakého k-počítače, který budeme nazývat cílový počítač (cílový proto, že naším cílem bude jeho programové řízení pomocí programů napsaných v k-jazyce). Předpokládejme, že pro náš k-počítač neexistuje hardware, který by realizoval jeho programy. Z kapitoly 1 víme, že k-počítač je možno realizovat pomocí počítače nižší úrovně - μ-počítače (existuje-li nějaký). μ-počítač, o kterém předpokládáme, že hardwarově existuje a na kterém chceme cílový počítač realizovat, budeme nazývat hostitelský počítač (hostitelský proto, že bude "hostit" cílový počítač). Pozor! Úroveň k-počítače (cílového počítače) a úroveň μ-počítače (hostitelského počítače) jsou dvě úrovně jednoho dvojúrovňového počítačového systému.

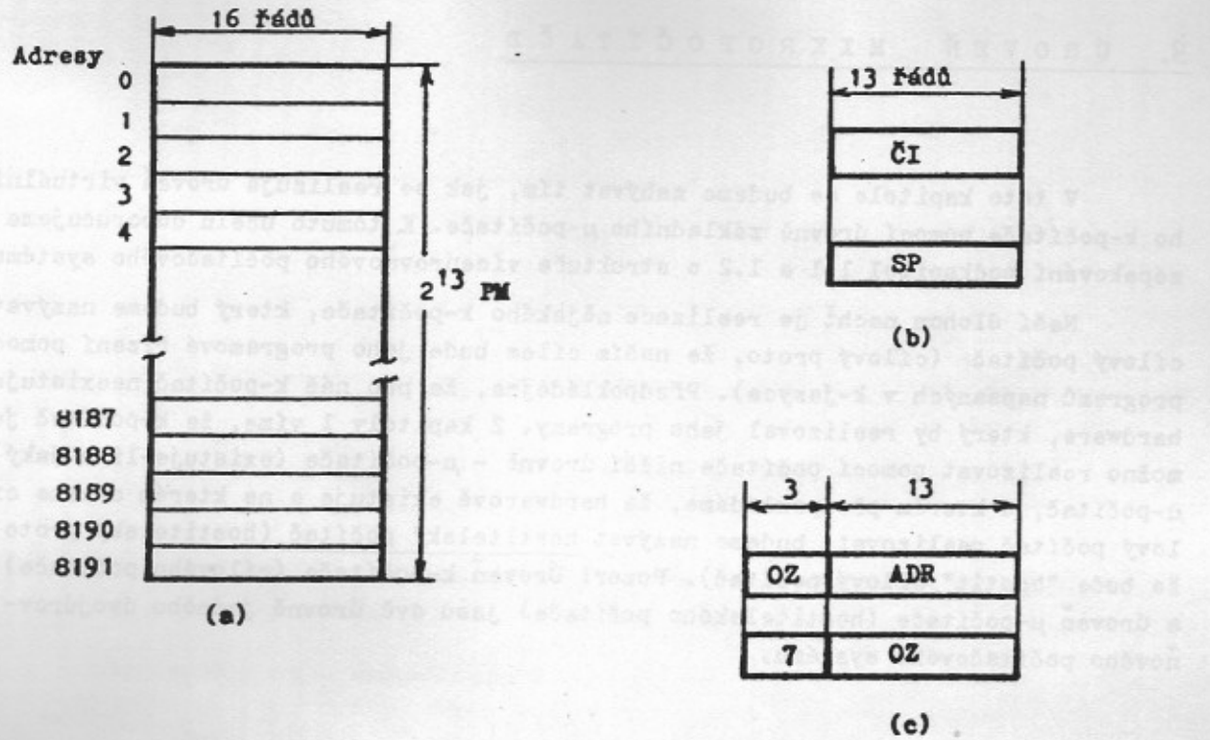
3.1 HYPOTETICKÝ CÍLOVÝ k-počítač (CIKAP)

3.1.1 Cílovým počítačem nechť je minipočítač s délkou slova 16 dvojkových řádů. Nejmenší adresovatelnou jednotkou je jedno slovo. Kapacita HP nechť je $2^{13} = 8192$ slov; PM budou mít adresy od 0 do $8191_{(10)}$, to je $0 - 17777_{(8)}$. Adresa bude tedy vyžadovat 13 bitů; AR bude mít 13 řádů a DR bude mít 16 řádů (obr. 4.1.1).

Nechť je CIKAP zásobníkový počítač s aritmetickými operacemi, které vyhodnocují operandy ze zásobníku a vkládají do něho výsledky. Zásobník je realizován v HP. Aritmetické operace používají operandy v doplňkovém kódu. Data jsou v HP uložena v doplňkovém kódu.

k-jazyk má dva formáty instrukcí: jednoadresový a bezadresový. Seznam instrukcí obsahuje 12 instrukcí (obr. 4.1.2):

- dvě jednoadresové instrukce PUSH a POP (odst. 3.4.8),
- jednoadresovou instrukci nepodmíněný skok,
- tři jednoadresové instrukce podmíněného skoku, které vyjímají a testují obsah vrcholu zásobníku T ($T < 0$, $T = 0$, $T \geq 0$); je-li podmínka splněna, skáče se na adresu specifikovanou v ADR instrukcí; na konci operace se testované T ztrácí,
- jednoadresovou instrukci skoku do procedury (volání procedury), která ukládá návratovou adresu (ČI + 1) do zásobníku, a potom se skočí na adresu specifikovanou v ADR instrukci,
- bezadresovou instrukci návratu z procedury (návrat do "hlavního" programu), která vybírá z vrcholu zásobníku adresu následující instrukce a ukládá ji do ČI,
- 4 bezadresové aritmetické instrukce (+, -, ×, :), které nejprve vybírají ze zásobníku 1. operand ("pravý") a potom druhý operand ("levý"); po provedení příslušné operace nad operandy se výsledek ukládá na vrchol zásobníku (na místo



Obr. 4.1.1 Programátorovi programujícímu v k-jazyce CIKAP jsou zřídi-telné: (a) HP o 8 K PM, (b) registry, (c) formáty k-in-strukcí strojového jazyka

000 ADR	PUSH:	Vložení obsahu HP(ADR) do zásob-níku	
001 ADR	POP:	Vyjmutí vrcholu zásobníku T a zá-pis do HP (ADR)	
010 ADR	SKO:	Skok na adresu ADR	
011 ADR	SZA:	Vyjmutí vrcholu zásobníku T a skok na ADR je-li T < 0	
100 ADR	SNU:	Vyjmutí vrcholu zásobníku T a skok je-li T = 0	
101 ADR	SNZ	Vyjmutí vrcholu zásobníku T a skok je-li T ≥ 0	
110 ADR	SPR:	Vložit návrato-vou adresu na vrchol zásobníku a skok do pod-programu	
111 0000000000000	PLU:	POP T, POP S, PUSH S+T	
111 0000000000001	MIN:	POP T, POP S, PUSH S-T	
111 0000000000010	NAS:	POP T, POP S, PUSH S*T	
111 0000000000011	DIV:	POP T, POP S, PUSH S:T	
111 0000000000100	SNA:	Vyjmout návrato-vou adresu z vr-cholu zásobníku a skok na ni	

Obr. 4.1.2 Seznam instrukcí k-jazyka CIKAP; T je slovo uchované na vrcholu zásobníku, S je slovo hned pod ním

bývalého 2. operandu). Jestliže T je okamžitý vrchol zásobníku a S je slovo bezprostředně pod ním, pak aritmetické instrukce vypočítávají hodnoty výrazů S + T, S - T, S * T a S : T.

P o z n á m k a : Na pořadí T a S při sčítání a násobení přirozeně nezáleží (jsou komutativní). Při odčítání a dělení je pořadí operandů T a S podstatné.

3.1.2 OZ instrukcí je možno rozšířit. Potenciálně by strojový jazyk CIKAP mohl mít 8192 bezadresových instrukcí se sedmičkou v nejvyšších třech řádech; jazyk CIKAP využívá jen 5 možností.

Ú l o h y

a) O které bezadresové instrukce byste doplnili CIKAP a proč?

b) Jaké datové typy se mohou realizovat v CIKAP na existujícím seznamu instrukcí? O které datové typy se rozšířil CIKAP splněním úlohy a) ?

c) Vytvořte všeobecný algoritmus volání procedury v jazyce CIKAP.

3.2 HYPOTETICKÝ HOSTITELSKÝ μ-počítač (HOMIP)

3.2.1 Interpret k-programů CIKAP poběží na HOMIP, jehož vnitřní organizace je znázorněna na obr. 4.2.1. Počítač hostitelské úrovně se skládá kromě jiného z 15 registrů, 39 datových cest řízených příslušnými řídícími body, 16řádové dvojkové sčítačky a posouvače o 1 řád vlevo.

3.2.2 Registry HOMIP je možno rozdělit do dvou skupin:

a) registry s proměnným obsahem,
b) registry s konstantním obsahem.

a) Registry mají následující funkce:

ČI - čítač instrukcí CIKAP (13 řádů) - přístupný programátorům jak na k-úrovni, tak i na μ-úrovni.

SP - směrník vrcholu zásobníku CIKAP (13 řádů) - přístupný programátorům jak na k-úrovni, tak i na μ-úrovni,

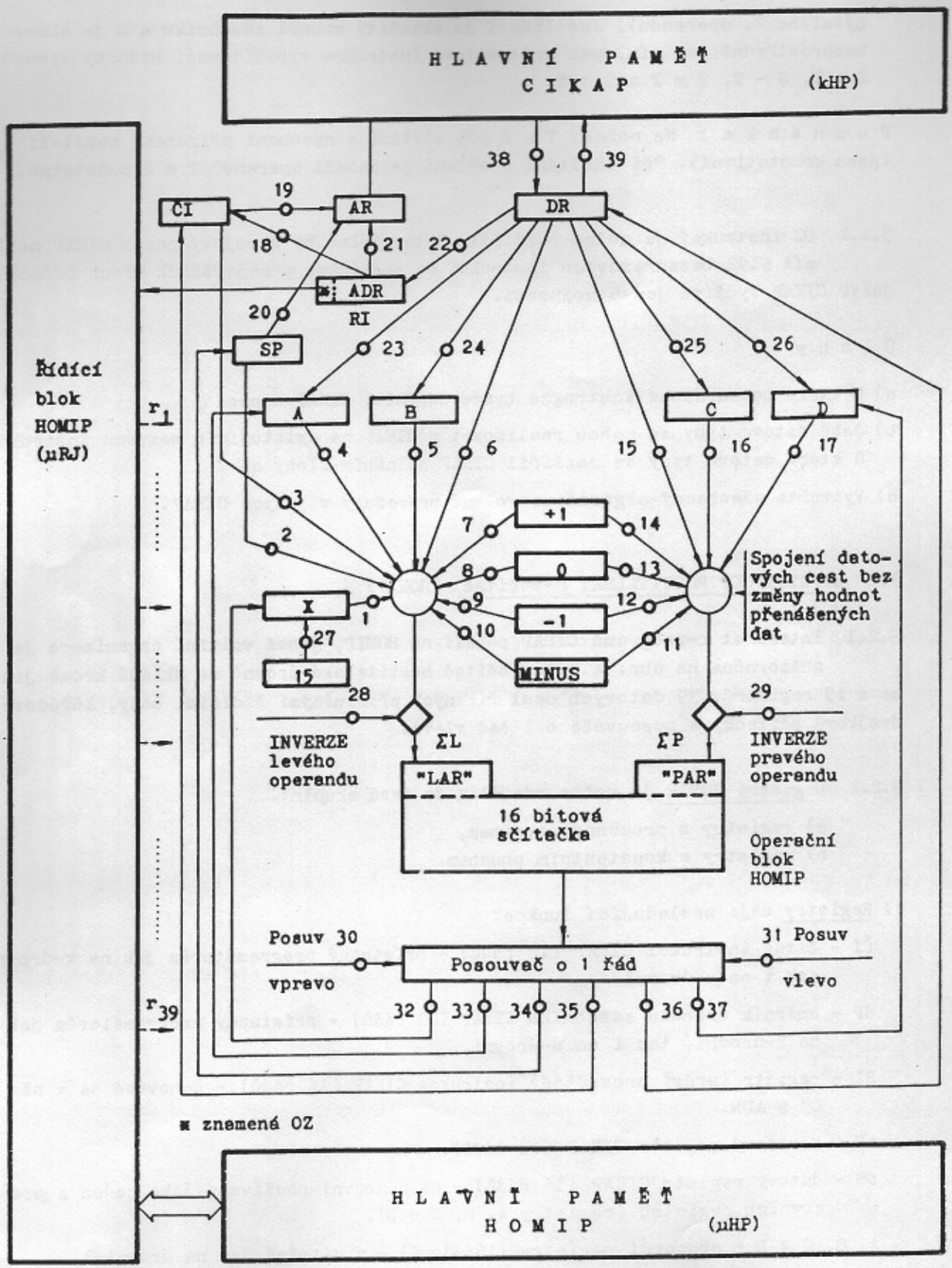
RI - registr (právě prováděné) instrukce CIKAP (16 řádů) - uchovává se v něm OZ a ADR.

AR - adresový registr CIKAP (13 řádů).

DR - datový registr CIKAP (16 řádů) - na μ-úrovni používaný jako jeden z pracovních registrů (registry A, B, C a D).

A, B, C a D - pracovní registry (16řádové) - viditelné jen na úrovni 1.

X - univerzální čítač (16 řádů), používaný hlavně jako čítač cyklů v μ-programech.



Obr. 4.2.1 Organizace a datové cesty hardwaru hostitelského počítače

Opakujeme, že programátor na k-úrovni "vidí" jen dva registry (ČI a SP); všechny registry (tedy včetně ČI a SP) vidí programátor na μ-úrovni.

b) Registry s konstantním obsahem uchovávají následující konstanty:

+1 - plus jeden,

0 - nula,

-1 - minus jeden,

15 - patnáct - nastavení čítače při zpracovávání slov po jednotlivých bitech,

MINUS - uchovává zápornou konstantu $-32\ 768_{(10)} = 1\ 000\ 000\ 000\ 000\ 000_{(2)}$ v doplňkovém kódu.

3.2.3 Operační jednotka HOMIP obsahuje 16řádovou sčítačku, která předpokládá operandy v doplňkovém kódu; výsledky jsou také v doplňkovém kódu. Přetečení se nezjišťuje. Výstup ze sčítačky jde vždy přes posouvač. Posouvač posouvá výstup ze sčítačky jen tehdy, je-li přiveden řídící signál 30 (posuv vpravo) nebo řídící signál 31 (posuv vlevo). Není-li přiveden ani jeden z uvedených signálů, posouvač výstup ze sčítačky kopíruje neposunutý.

Sčítačka má dva vstupy: pravý a levý. Můžeme si představit, že sčítačka má dva fiktivní registry "LAR" a "PAR". Podle toho pak budeme mluvit o pravém a levém operandu, o pravém a levém registru apod. Symbolicky budeme levý vstup do sčítačky označovat Σ L a pravý vstup Σ P.

3.2.4 Datové cesty a řídící body. HOMIP má 39 řídících bodů; některé z nich řídí přesuny mezi registry, jiné otevírají cesty do sčítačky a z ní. Řídící body a signály 28-31 a 38 a 39 ovládají provádění zvláštních funkcí.

Následuje popis významu jednotlivých řídících bodů (resp. řídících signálů); čísla se shodují s čísly na obr. 4.2.1.

a) Řízení vstupů do sčítačky (RB1 - RB10 řídí levé vstupy a RB11 - RB17 řídí pravé vstupy):

1. X do sčítačky - řídí odečtení jedničky z X při řízení cyklů v μ-programech.
2. SP do sčítačky - řídí přičítání jedničky do SP, resp. odčítání jedničky z něho.
3. ČI do sčítačky - řídí přičítání jedničky do ČI nebo přesun ČI do HP přes DR.
4. A do sčítačky - řídí sčítání obsahu A s pravým operandem nebo přesun obsahu A do HP přes DR.
5. B do sčítačky - řídí sčítání obsahu B s pravým operandem nebo přesun obsahu B do HP přes DR.
6. DR do sčítačky - řídí sčítání obsahu DR s pravým operandem; v DR je obvykle uložen druhý operand.
7. +1 do sčítačky - řídí přičítání jedničky k obsahu některého pravého registru; např. D := 1 + D, ale i DR := 1 + DR.

8. 0 do sčítačky - řídí přesun obsahu některého pravého registru nebo některého levého registru; např. A := 0 + C nebo D:= 0 + DR.

9. -1 do sčítačky - řídí odčítání jedničky od obsahu pravého registru.

10. MINUS do sčítačky - řídí zaznamenání záporné konstanty MINUS do některého pravého registru; např. při současném posuvu dvou registrů A, D, když A(0) = 1 se má posunout do D(7) - viz i AC a MQ na obr. 2.5.1.

11. MINUS do sčítačky - jako 10 se změněnými stranami sčítačky; MINUS je pravý operand.

12. -1 do sčítaňky - jako 9 se změněnými stranami sčítačky; -1 je pravý operand.

13. 0 do sčítačky - jako 8 se změněnými stranami sčítačky; 0 je pravý operand.

14. +1 do sčítačky - jako 7 se změněnými stranami sčítačky; +1 je pravý operand.

15. DR do sčítačky - jako 6 se změněnými stranami sčítačky; obsah DR je pravý operand.

16. C do sčítačky - řídí sčítání obsahu D s levým operandem nebo přesun D do jiného registru; např. A := 0 + D.

b) Přesuny mezi registry

18. RI do ČI - řídí přesun ADR instrukce do ČI při instrukci skoku; datová cesta má 13 řádů.

19. ČI do AR - řídí čtení následující instrukce z HP cílového počítače (13řádová cesta).

20. SP do AR - řídí vybírání položky ze zásobníku nebo ukládání položky do zásobníku (13řádová cesta).

21. RI do AR - řídí čtení obsahu ADR z HP nebo zápis na adresu ADR v HP.

22. DR do RI - řídí přesun instrukce, která se má provádět z DR do RI po jejím přečtení z HP.

23. DR do A - řídí přesun obsahu DR (obvykle jako levého operandu) do A.

24. DR do B - řídí přesun obsahu DR (obvykle jako levého operandu) do B.

25. DR do C - řídí přesun obsahu DR (obvykle jako levého operandu) do C.

26. DR do D - řídí přesun obsahu DR (obvykle jako levého operandu) do D.

27. 15 do X - řídí nastavení čítače cyklů na maximální hodnotu 15.

c) Inverze a posuvy

28. Působí-li na řídící bod 28 řídící signál, vstupuje do levého vstupu sčítačky inverzní kód levého operandu - viz operaci NEG z podkapitoly 2.3.

29. Působí-li na řídící bod 29 řídící signál, vstupuje do pravého vstupu sčítačky inverzní kód pravého operandu.

Jestliže na řídící body 28 a 29 nepůsobí žádný řídící signál, přesuny do sčítačky se uskutečňují bez inverze operandů.

30. Působí-li na řídící bod 30 řídící signál, posouvá se výstup ze sčítačky o 1 řád napravo (logický posuv).

31. Působí-li na řídící bod 31 řídící signál, posouvá se výstup ze sčítačky o 1 řád doleva (logický posuv).

Jestliže na řídící bod 31 a zároveň na RB31 nepůsobí řídící signál, posuv se neuskuteční.

d) Výstupy ze sčítačky (přes posouvač)

32. Výstup ze sčítačky do X - řídí odčítání jedničky z čítače cyklů X.

33. Výstup ze sčítačky do A - řídí přesun výsledku sčítání do A.

34. Výstup ze sčítačky do SP - řídí přesun výsledku sčítání do SP (obvykle přičtení nebo odečtení jedničky).

35. Výstup ze sčítačky do ČI - řídí přesun výsledku sčítání do ČI (ovykle přičtení jedničky).

36. Výstup ze sčítačky do DR - řídí přesun výsledku sčítání do DR.

37. Výstup ze sčítačky do D - řídí přesun výsledku sčítání do D.

Výstup ze sčítačky může být před vstupem do libovolného uvedeného registru posunut o 1 řád vpravo nebo vlevo (RB30, resp. RB31).

e) Čtení a zápis do HP cílového počítače

38. HP do DR - řídí přesun při čtení obsahu PM do DR,

39. DR do HP - řídí přesun při zápise obsahu DR do PM.

HP vybírá adresu příslušného PM podle obsahu AR.

4.2.5 Obsah některého registru je možno přesouvat do více registrů současně. Např. obsah DR je možno z nějakého důvodu přesunout najednou do registrů A, B, C a D. V tomto případě se řídící body 23-26 otevřou současně. Na druhé straně však není účelné ani dovolené přesunovat obsahy dvou nebo více registrů do jednoho registru. Kdyby se např. současně otevřely řídící body 19-21, obsah AR by byl nedefinován.

Při sčítání dvou operandů musí být do sčítačky otevřen současně právě jeden levý a jeden pravý řídící bod. Posouvač může posunovat buď napravo (RB30), nebo nalevo (RB31), ale ne v obou směrech najednou; RB30 a RB31 mohou ovšem být současně oba zavřeny.

Ú l o h y

a) Napište posloupnost otevírání řídících obvodů počítače HOMIP při provádění instrukcí PUSH, POP, PLU, MIN počítače CIKAP. Kolika způsoby se může každá z nich provádět?

b) Doplňte obrázek 4.2.1 tak, aby se mohly provádět logické operace definované zásobníkovými instrukcemi AND a OR.

3.3 POSLOUPNOST PROVÁDĚNÍ μ-operací

Víme, že mikrooperace se řídí otevřením datových cest:

- mezi registry (přesun údajů z registru do registru beze změny jeho hodnoty, významu, ...),
- mezi registrem a funkční jednotkou nebo mezi funkční jednotkou a registrem (zpracování údaje a jeho uchování v registru).

Připomínáme, že funkční jednotka (sčítačka, posouvač apod. – obr. 2.6.4) obvykle neobsahuje paměťové prvky, a proto musí být výsledek po zpracování údaje (údajů) okamžitě uložen (obvykle do registru).

Mikrooperace se tedy provádějí tak, že se otvírají a zavírají některé řídící body. Otvírání a zavírání ovládá řídící signál přicházející z RJ.

3.3.2 Jak se programy, a tedy i instrukce CIKAP interpretují pomocí posloupnosti mikrooperací definovaným mikroinstrukcemi jazyka HOMIP, probereme v dalších odstavcích na příkladu (k-)instrukce PLU (na zásobníku).

Etapu provádění instrukce PLU je možno rozdělit do čtyř kroků (to ještě nejsou μ-operace):

1. Vybrání prvního operandu ze zásobníku a jeho uložení do některého registru HOMIP.
2. Vybrání druhého operandu ze zásobníku a jeho uložení do některého jiného registru HOMIP.
3. Vytvoření součtu obou operandů a jeho uložení v registru.
4. Vložení součtu do zásobníku.

Algoritmy těchto kroků je třeba realizovat pomocí otevírání a zavírání řídících bodů. Je třeba si uvědomit, že v případě, kdy je na datové cestě více řídících bodů (prochází více registry nebo funkčními bloky), nemohou být některé za sebou jdoucí řídící body otevřeny najednou (- generátor fází). Kdyby se například RB4, RB13, RB36 a RB39 otevřely současně, nevěděli bychom, co si HP zapamatuje; byl by to obsah registru A, nebo původní obsah D, nebo něco jiného? Odpověď není jednoznačná.

Aby byly přesuny jednoznačné, je třeba každý krok rozdělit na několik fází. Bude-li to možné, dovolíme, aby se fáze jednotlivých kroků překrývaly - ušetří se tím čas. Pro jednoduchost předpokládejme, že každá μ-operace "čtení z HP" a "zápis do HP" se provádí v jedné fázi. (Doporučujeme zopakovat si algoritmy čtení a zápisu do HP.)

3.3.3 Předpokládejme, že etapa čtení instrukce PLU z HP již skončila a že instrukce PLU je již v RI (a že je dekódována). Etapa provádění instrukce PLU se může skládat z následujících kroků a fází (sledujte obr. 4.2.1):

1. krok - přečtení 1. operandu (ze zásobníku) a jeho uložení do registru A neboli A := HP(SP):

ø1: otevření RB20 - adresa vrcholu zásobníku do AR; AR := SP
ø2: otevření RB38 - přečtení obsahu vrcholu zásobníku (1. operand) do DR; DR := HP(AR)
ø3: otevření RB23 - přesun 1. operandu z DR do A; A := DR

2. krok - přečtení 2. operandu ze zásobníku a jeho uložení do DR neboli DR := HP(SP):

ø4: otevření RB2 - přesun obsahu SP do Σ; SP → ΣL a současně otevření RB12 - přesun "obsahu" - 1 do Σ; -1 → ΣP a současně sčítání obsahu SP a -1 v Σ; (?):= SP + (-1)
(nechť přesun obsahů registrů do sčítačky a vlastní sčítání dohromady trvá právě jednu fázi)
ø5: otevření RB34 - přesun součtu do SP; SP := SP + (-1)
ø6: otevření RB20 - přesun nového obsahu SP do AR; AR := SP
ø7: otevření RB38 - přečtení obsahu nového vrcholu zásobníku (2. operand) do DR; DR := HP(AR)

3. krok - sčítání 1. a 2. operandu a uložení součtu v DR neboli DR := A + DR:

ø8: otevření RB4 - přesun obsahu A do Σ; A → ΣL a současně:
otevření RB15 - přesun obsahu DR do Σ; DR → ΣP a současně:
sčítání obsahu A a DR v Σ; (?) := A + DR
ø9: otevření RB36 - přesun součtu do DR; DR := A + DR

4. krok - uložení součtu (výsledku operace PLU) do zásobníku (na místo bývalého 2. operandu) neboli HP(SP) := DR

ø10: otevření RB20 - přesun obsahu SP do AR; AR := SP
ø11: otevření RB29 - zápis obsahu DR do zásobníku v HP; HP(AR) := DR

Právě jsme ukázali, jak se dá operace definovaná instrukcí na úrovni 2 rozložit na μ-operace počítače na úrovni 1. U každé fáze je ze středníkem uvedena μ-instrukce, která definuje příslušnou μ-operaci.

4.3.4 Podívejme se nyní pozorně na posloupnost μ-operací z předcházejícího odstavce. Je možno daný algoritmus provádění instrukce PLU zkrátit (např. na 8 fází)? Sledujme obr. 4.2.1, 4.3.1 a 4.3.2 a uvažujme:

Původní obsah SP potřebujeme jen jednou ve ø1. V následující fázi ø2 můžeme tedy do SP uložit již nový obsah SP (původní minus jedna). K tomu účelu můžeme ve ø1 paralelně s otevřením RB20 otevřít i RB2 a současně RB12. Mikrooperace z fází ø1 a ø4 se tedy mohou provádět současně v nové fázi F1 (fáze F budou stejně dlouhé jako původní fáze ø):

(F1) : otevřít RB20 a současně RB2 a současně RB12:

do fáze F1 jsme tedy sloučili přípravu čtení 1. a část přípravy čtení 2. operandu ze zásobníku.

Uvažujme dále:

Adresu 1. operandu máme připravenou v AR, a je tedy možno vybrat 1. operand z vrcholu zásobníku (zásobník je v HP); jedná se o původní μ-operaci z ø2 - otevře-

ní RB38. Současně s tím je možno přesunout součet původního obsahu SP a -1 do SP neboli provést μ-operaci z ø5 - otevření RB34. Nová fáze F2 se tedy bude skládat z μ-operací z fází ø2 a ø5:

(F2) : otevřít RB38 a současně RB34;

to znamená, že 1. operand je už v DR a adresa 2. operandu v SP.

Nový obsah SP můžeme přesunout do AR (dokončení přípravy 2. operandu) a 1. operand musíme z DR přesunout do jiného registru (vybrali jsme si registr A), abychom v následující fázi nepřepsali obsah DR druhým operandem. Fáze F3 se tedy bude skládat z otevření RB20 (původně ve ø6)a RB23 (původně ve ø3):

(F3) : otevřít RB20 a současně RB23;

stav procesu provádění instrukce PLU po F3 bude:

- SP obsahuje adresu 2. operandu (nesmíme jej přepsat, protože je to jediný ukazatel na vrchol zásobníku),
- AR obsahuje také adresu 2. operandu (budeme jej potřebovat k vložení součtu do zásobníku),
- DR obsahuje 1. operand, ale již nás nezajímá, protože i
- A obsahuje 1. operand.

Pro vytvoření součtu nám ještě chybí přečtení 2. operandu ze zásobníku, a tedy ve fázi F4 se uskuteční μ-operace z původní fáze ø7 :

(F4) : otevřít RB38;

ve fázi F5 se již může sčítat obsah registru A (1. operand) s obsahem registru DR (2. operand); jde o μ-operaci fáze ø8 (přesun obou operandů do sčítačky a sečtení):

(F5) : otevřít RB4 a současně RB15;

ve fázi F6 se může provést μ-operace z fáze ø9 (součet ze sčítačky do DR):

(F6) : Otevřít RB36;

protože μ-operace ve ø10 je zbytečná (v AR je stále ještě adresa 2. operandu, na kterou se má součet zapsat), může se ve fázi F7 již zapsat výsledek operace na vrchol zásobníku:

(F7) : otevřít RB39;

z původních 11 fází ø se nám podařilo provádění instrukce PLU zkrátit na 7 fází F.

Všimněme si, že :

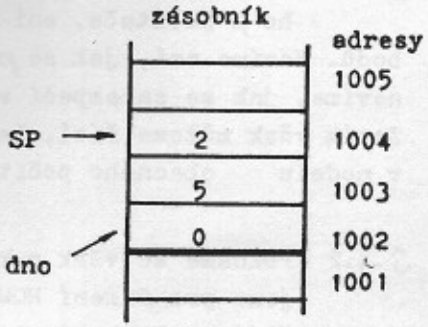
- po přečtení 2. operandu z HP jsme neodečítali jednu od obsahu SP (výsledek se ukládá na místo 2. operandu),
- 1. a 2. krok algoritmu sčítání se navzájem v čase překrývají; ušetřili jsme tím 3 fáze a zkrátili tím dobu provádění operace PLU o více než o čtvrtinu.

Ú l o h a

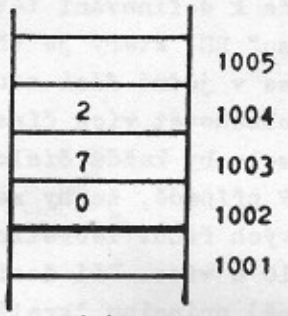
a) Upravte úlohy 4.2.5 a) a b) podle vědomostí z podkapitoly 3.3.

Stav před fází	SP	AR	DR	A
F1	1004	?	?	?
F2	1004	1004	?	?
F3	1003	1004	2	?
F4	1003	1003	2	2
F5	1003	1003	5	2
F6	1003	1003	5	2
F7	1003	1003	7	2
Konečný stav	1003	1003	7	2

(b)

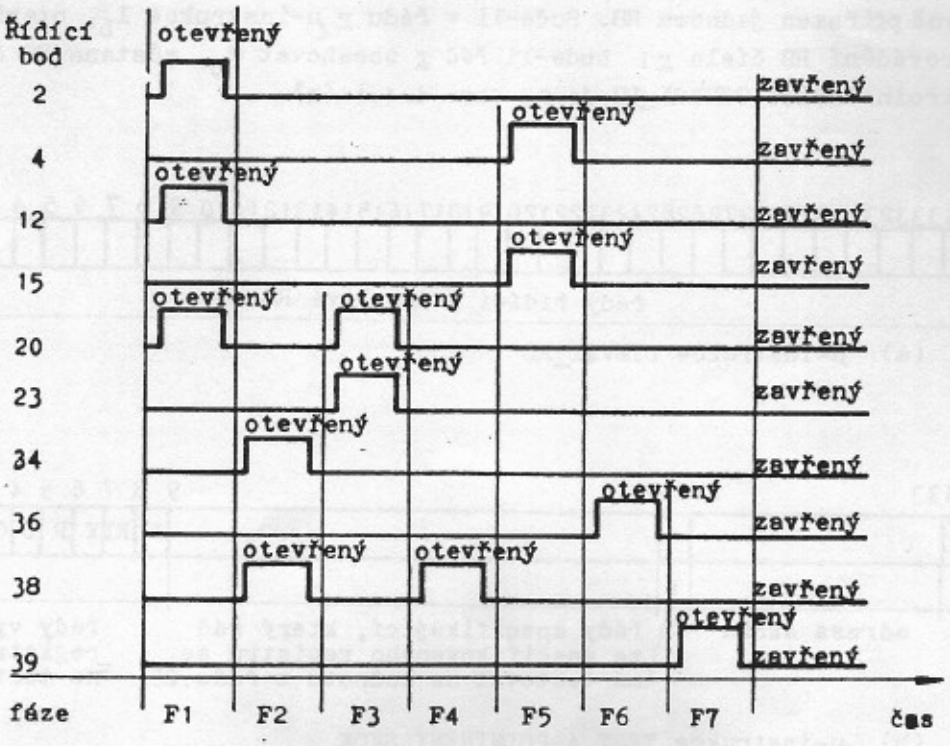


(a)



(c)

Obr. 4.3.1 a) Stav zásobníku před prováděním instrukce PLU; b) stav registrů po dobu provádění μ-operací před začátkem každé fáze c) stav zásobníku po provedení μ-programu instrukce PLU



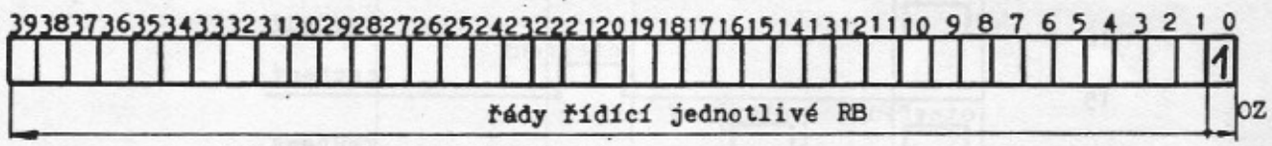
Obr. 4.3.2 Etapa provádění instrukce PLU pomocí μ-operací ve fázích F1 až F7

3.4 μ-jazyk HOSTITELSKÉHO POČÍTAČE

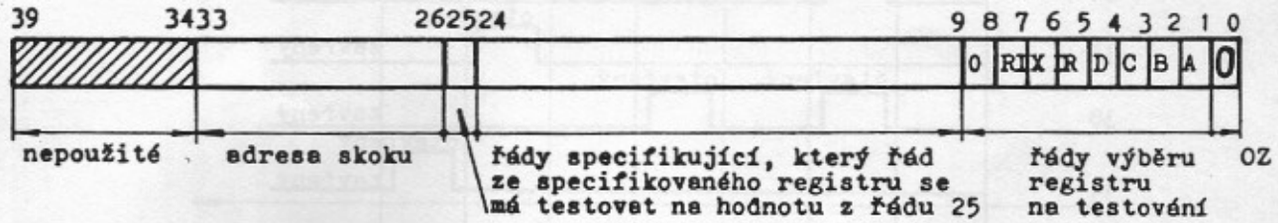
3.4.1 Dosud jsme nespecifikovali ani instrukce našeho hypotetického hostitelského μ-počítače, ani způsob, jakým se řídí posloupnosti otevírání řídících bodů. Nevíme ani, jak se μ-programy uchovávají a jak se provádějí. Jinými slovy nevíme, jak se zabezpečí vytvoření řídících signálů a jejich správná posloupnost. Zatím však můžeme říci, že všeobecnou odpověď na uvedené otázky je třeba hledat v modelu obecného počítače.

3.4.2 Pokusme se však o konkrétnější odpovědi na otázky: Jaké instrukce potřebujeme pro řízení HOMIP? Co by měla umět jeho řídící jednotka? Po krátké úvaze přijdeme lehko na to, že RJ hostitelského počítače musí umět otevírat a zavírat řídící body v jistých, přesně definovaných okamžicích. Můžeme si představit, že k definování této činnosti stačí jedna instrukce, která bude mít v ADR "adresu" RB, který je třeba právě otevřít. Instrukce nechť se jmenuje OTEVRI_RB. Má-li se v jedné fázi současně otevřít více řídících bodů, musela by ADR μ-instrukce obsahovat více čísel RB. Vzhledem k tomu, že hostitelský μ-počítač má 39 RB, muselo by každé číslo RB v dvojkové soustavě mít 6 bitů ($2^5 - 1 < 39 < 2^6 - 1$). V případě, že by se mělo otevřít např. 6 RB, musela by ADR obsahovat 36 dvojkových řádů. Teoreticky by bylo možné otevřít i víc RB současně; dá se ukázat, že 10 a více. Při desíti bychom potřebovali již 60 bitů. To odporuje (na jedné straně) principu "kratší instrukce je lepší než dlouhá" a (na druhé straně) by obvody byly nepřijatelně složité.

Počet řádů v ADR μ-instrukce je možno zredukovat na 39 tak, že každý řád ADR bude pevně přiřazen jednomu RB. Bude-li v řádu r μ-instrukce 1_L, otevře se při jejím provádění RB číslo r; bude-li řád r obsahovat 0_L, zůstane RB číslo r zavřený. Mikroinstrukce OTEVRI_RB je na obr. 4.4.1 (a).



(a) μ-instrukce OTEVRI_RB



(b) μ-instrukce TEST_A_PODMÍNĚNÝ SKOK

Obr. 4.4.1 Seznam μ-instrukcí hypotetického hostitelského μ-počítače má jen dvě μ-instrukce

3.4.3 Z programátorské praxe již dobře víme, že se mnohé programy nedají realizovat bez podmíněných skoků; některé algoritmy se bez nich realizují velmi obtížně. Intuitivně tedy cítíme, že i náš μ-jazyk by měl obsahovat μ-instrukci podmíněného skoku. μ-instrukce podmíněného skoku by měla obsahovat podmínku a adresu, kam se má v μ-programu skočit, je-li podmínka splněna.

Nechť má μ-instrukce podmíněného skoku formát podle obr. 4.4.1 (b) a nechť se jmenuje TEST_A_PODMINENY_SKOK. Je to poměrně univerzální μ-instrukce, umožňuje testování hodnoty libovolného bitu sedmi registrů IR, X, DR, D, C, B, A a konstantního registru 0 (pro nepodmíněný skok). Pomocí μ-instrukce testování jednotlivých bitů je možno naprogramovat i testování složitějších relací jako $<$, $\neq$, $\geq$, ... , testování nuly, obsahu celého registru apod. V μ-instrukci TEST_A_PODMINENY_SKOK je též možno volit, zda se má daný řád určeného registru testovat na 1_L nebo 0_L (řád 25).

Je-li splněna podmínka určená řády 1-24 a hodnotou řádu 25, změní se posloupnost provádění μ-instrukcí skokem na μ-instrukci na adrese určené řády 26 až 33. Z toho plyne, že paměť μ-programu má kapacitu maximálně 256 (μ-) slov.

Shrnutí:

1. Do jednoho z řádů 1-8 μ-instrukce TEST_A_PODMINENY_SKOK se zapíše 1_L podle toho, který registr se má testovat.
2. Do jednoho z řádů 9-24 μ-instrukce se zapíše 1_L podle toho, který řád registru určeného v bodě 1 se má testovat.
3. Do řádu 25 μ-instrukce se zapíše, na jakou hodnotu se mají řády určené v bodě 2 testovat (1_L nebo 0_L).

   V řádech 1-8, resp. 9-24 μ-instrukce TEST_A_PODMINENY_SKOK může být vždy jen po jedné jedničce.

Jazyk HOMIP tedy vystačí s dvěma μ-instrukcemi. Na OZ μ-instrukce pak stačí pouze 1 řád. Bude jím řád 0 μ-instrukce; bude-li řád 0 μ-instrukce obsahovat:

- 1_L, jedná se o μ-instrukci OTEVRI_RB,
- 0_L, jedná se o μ-instrukci TEST_A_PODMINENY_SKOK.

Z obsahu předcházejícího a tohoto odstavce vyplývá, že délka (μ-) slova paměti μ-programu HOMIP bude 40 dvojkových řádů.

Ú l o h a

a) Jakým způsobem by se daly využít nepoužité řády μ-instrukce TEST_A_PODMINENY_SKOK?

3.5 ŘÍZENÍ u-operací V HOSTITELSKÉM POŘÍTAČI

3.5.1 V této kapitole jsme zatím postupovali takto (porovnej s obsahem podkapitol 1.1 a 1.2):

- v podkapitole 3.1 jsme definovali hardware cílového počítače úrovně 2 (k-počítač a jeho k-jazyk),

- v podkapitole 4.2 jsme definovali hardware hostitelského počítače úrovně 1, který by měl interpretovat instrukce úrovně 2,
- v podkapitole 4.3 jsme ukázali, jak by měl podle našeho přání fungovat hardware z podkapitoly 4.2 při interpretaci instrukcí úrovně 2,
- v podkapitole 4.4 jsme na základě vědomostí o činnosti hardwaru μ-počítače definovali μ-jazyk; μ-program sestavený z jeho μ-instrukcí bude interpretovat k-instrukce.

Zbývali jsme se tedy tím, co zatím "vidí" programátor:

1. Programátor v strojovém jazyce vidí: HP cílového počítače úrovně 2, ČI, SP a IR (jeho programy se pro něho vykonávají jen v HP a existenci OJ a RJ může jen tušit).

   P o z n á m k a 1 : Programátor ve strojovém jazyce tedy nepotřebuje znát způsob implementace strojového jazyka a zařízení, které je pro implementaci použito. V zásadě je možno uskutečnit jiné implementace téhož strojového jazyka na jiném (dokonce i na stejném) zařízení. Často je strojový jazyk staršího počítače implementován na novějším počítači (jako "druhý" strojový jazyk novějšího počítače); je tím umožněno využívání programů pro starší počítač i na novějším počítači. Obvykle se pak hovoří o emulaci (napodobení) staršího počítače na novějším (odstavec 2.1.7 - počítače třídy SM-51).

   P o z n á m k a 2 : Další možností je implementace jednoho strojového jazyka na několika zařízeních s různou rychlostí a cenou. Příkladem jsou počítače JSEP-2.

2. Programátor v μ-jazyce vidí: všechny registry a funkční jednotky hostitelského počítače na úrovni 1 (včetně ČI, SP a IR) a HP cílového počítače; zatím neví, jak je sestrojena RJ, která řídí datové cesty podle μ-instrukcí.

Mohli bychom se současně pokusit vyjmenovat, co zatím "patří" hardwaru:

A. Cílového počítače: HP, ČI, SP, RI; programy v strojovém jazyce nemají přístup do jiných částí hardwaru; cílovému počítači chybí RJ.

B. Hostitelského počítače: všechny registry a funkční jednotky z obr. 4.2.1, HP cílového počítače; intuitivně cítíme, že hostitelskému počítači zatím chybí jeho vlastní RJ a HP.

Z bodů 2 a B plyne, že bychom měli náš HOMIP doplnit o RJ a HP. V HP hostitelského počítače na úrovni 1 budou uchovány μ-programy. Jeho řídící jednotka (RJ) bude číst μ-instrukce z HP hostitelského počítače a provádět je (otevírat a zavírat RB pomocí řídících signálů).

Je třeba si uvědomit, že HOMIP je pro CIKAP jak operační, tak i řídící jednotka (biz body 1 a A).

3.5.2 Hlavní paměť hostitelského počítače HOMIP obsahuje μ-programy interpretující instrukce strojového jazyka CIKAP. Vzhledem k tomu, že HOMIP plní funkci řídící jednotky CIKAP, říká se hlavní paměti μ-počítače i řídící paměť. Z pochopitelných důvodů se nazývá i mikropaměť. Pro naše potřeby ji budeme označovat μHP.

μ-HP se logicky odlišuje od (k) HP. Fyzicky mohou být a někdy i jsou v jedné paměťové jednotce počítače. V HP počítače úrovně 2 se uchovávají programy napsané ve strojovém jazyce a data jimi zpracovávaná. V μHP našeho HOMIP na úrovni 1 se uchovávají μ-prograny bez dat; konstanty využívané μ-programy jsou uložené v registrech konstant. Mikroprogramem zpracovávaná data jsou v registrech.

Slovo μHP počítače HOMIP bude mít délku 40 dvojkových řádů, což je délka μ-instrukce. Její kapacita bude 256 PM; je dána počtem řádů rezervovaných pro adresu skoku v instrukci TEST_A_PODMINENY_SKOK. Principiálně by adresa μHP mohla mít až 14 řádů a kapacita by tedy mohla být až 2^{14} slov. 256 PM však úplně stačí pro uložení interpretu instrukcí strojového nazyka.

3.5.3 Řídící jednotku hostitelského počítače HOMIP budeme označovat μRJ. V souladu s dosud získanými znalostmi o řídící jednotce, musí μRJ obsahovat vlastní ČI, neboli čítač μ-instrukcí - ČμI. Vzhledem k tomu, že v μHP jsou jen μ-instrukce (nejsou v ní uloženy operandy), nepotřebujeme zvláštní AR; ČμI bude zastávat funkci AR i ČI.

Každá μ-instrukce se po přečtení z μHP obvykle uchovává v registru μ-instrukcí, který budeme označovat RμI. Jednotlivé řády RμI se interpretují podle obsahu jeho řádu 0 (instrukce OTEVRI_RB nebo TEST_A_PODMINENY_SKOK). Bude-li RμI(0) = 1, může zbývajících 39 řádů přímo ovládat jednotlivé RB. Bude-li RμI(0) = 0, provede se test specifikovaného řádu specifikovaného registru a po jeho vyhodnocení se buď přepíše obsah ČμI adresou skoku - řády RμI (33-26), nebo se v μ-programu pokračuje v původním pořadí μ-instrukcí. Testovacími obvody se zabývat nebudeme; jsou relativně složité. Při provádění testování nejsou řídící body řízeny (fáze je využita jen pro test). Při konstrukci rychlého počítače by to však bylo nevýhodné.

Vzhledem k tomu, že v μHP nejsou data, nepotřebujeme žádný datový registr; RμI zastává funkci DR a RI.

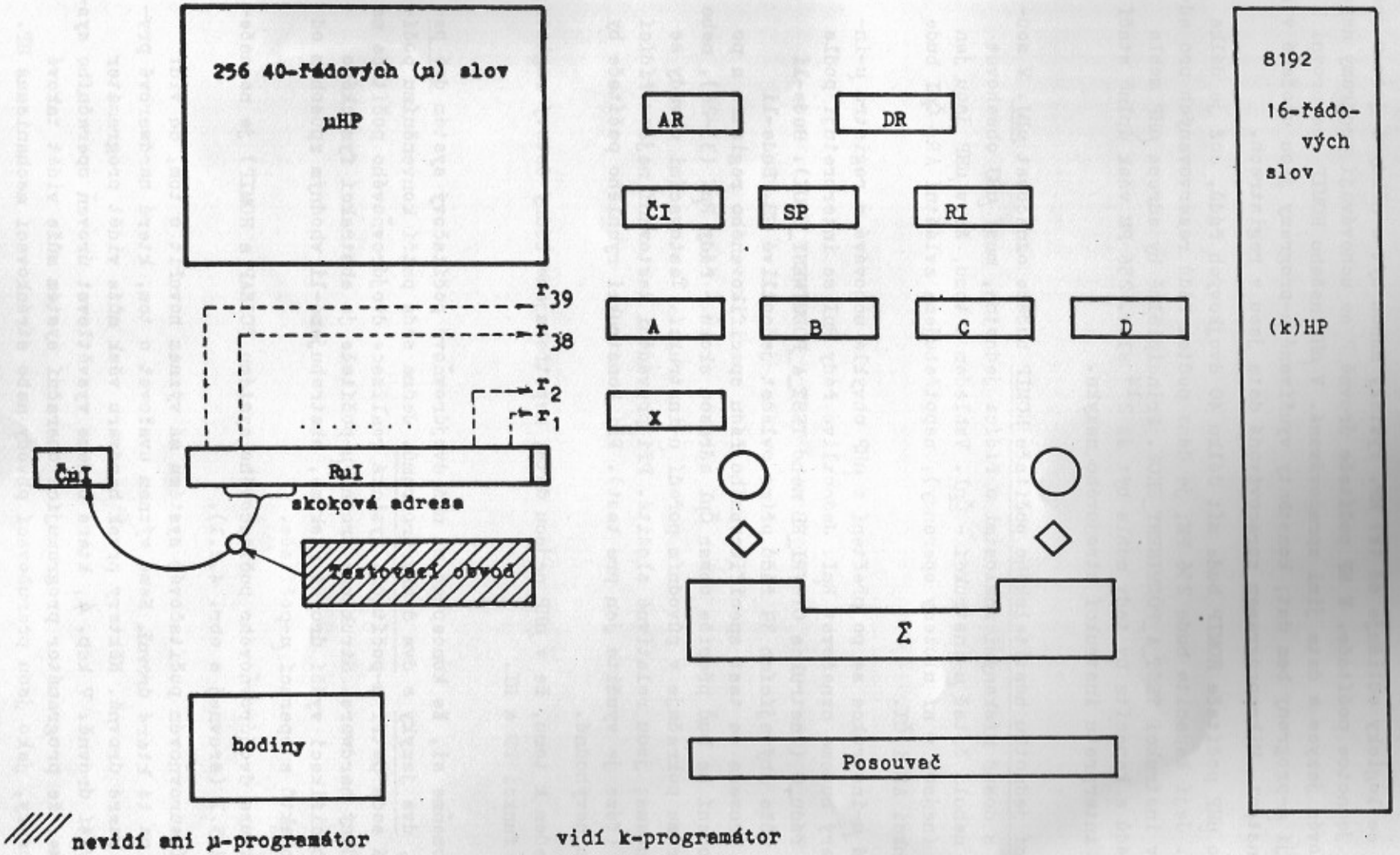
3.5.4 Připomeňme si, že koncepčně má náš dvojúrovňový počítačový systém dvě paměti, dva jazyky a dva druhy programů. Jedna sada patří konvenčnímu počítači a druhá sada patří μ-počítači. Fyzická realizace dvojúrovňového počítače má však jen jediný hardware. Struktura úrovně μ-počítače je abstrakcí fyzického hardwaru. Specifikaci vyšší úrovně získáme, abstrahujeme-li vhodným způsobem od některých objektů a operací μ-počítače.

Struktura dvojúrovňového počítačového systému (CIKAP a HOMIP) je naznačena na obr. 4.5.1 (srovnej s obr. 4.2.1).

Ve víceúrovňovém počítačovém systému má význam hovořit o tom, co vidí programátor na té které úrovni. Nemá význam uvažovat o tom, které hardwarové prvky patří do které úrovně. Některý prvek hardwaru však může vidět programátor z některé vyšší úrovně. V kap. 4, která budeme vysvětlovat úroveň operačního systému, uvidíme, že programátor programující operační systém může vidět takové hardwarové prvky, jako jsou přerušovací obvody nebo stránkovací mechanismus HP.

Ú l o h a

a) Pokuste se pojmy z HOMIP A CIKAP zařadit do schematického popisu počítačového systému z kap. 1.



Obr. 4.5.1 Prvky hardwaru hostitelského mikropočítače (bez spojů)

3.6 POMOCNÝ JAZYK PRO MIKROPROGRAMOVÁNÍ

3.6.1 Psát μ-programy ve tvaru posloupnosti 40bitových čísel není pohodlné a je obtížné vyvarovat se chyb. Z toho důvodu byly vymyšleny symbolické mikroprogramovací jazyky. Pro zápis μ-programu budeme používat jednoduchý symbolický mikroprogramovací jazyk, který budeme označovat μJ. Bude zkonstruován z podobných příkazů, jako se používají v jazycích ALGOL, PASCAL apod.

Mikroprogramovací jazyk se mimo jiné často liší od strojových jazyků tím, že umožňuje vyjádřit souběžnost provádění μ-operací a tedy paralelní otevírání řídících bodů. V jedné fázi F (nazývané také takt) může μ-instrukce definovat více než jeden přesun mezi registry nebo mezi registry a funkčními jednotkami. Proto bude jeden řádek μ-programu obsahovat vždy jednu μ-instrukci, která se provádí v jednom taktu. O etapě čtení μ-instrukce z μ-HP zatím nebudeme uvažovat. Všechny příkazy napsané v jednom řádku se budou provádět paralelně.

3.6.2 Zápis instrukce OTEVRI_RB. přesun mezi registry budeme zapisovat jako doposud pomocí přiřazovacího příkazu. Např.

A := DR; znamená, že se otevře RB23 a obsah DR se přesune do registru A.

Každý příkaz v μ-instrukci musí být ukončem středníkem. Pořadí příkazů v řádku je nepodstatné. Dva řádky:

```
A := DR;  AR := RI(ADR);  S := 15;
AR := RI(ADR);  X := 15;  A := DR;
```

definují stejnou μ-instrukci: otevření RB21, RB23 a RB27 současně. Avšak

```
A := DR;
AR := RI(ADR);
X := 15;
```

znamená postupné otevírání RB23, RB21 a RB27.

Použití sčítačky vyžaduje dva takty (sčítačka nemá paměť a vlastní sčítání trvá jistou dobu - např. 1/2 taktu):

- otevření vstupů do sčítačky,
- otevření výstupů ze sčítačky; to současně znamená otevření vstupů do registrů výsledku.

Předpokládáme, že sčítání se uskuteční na konci 1. taktu a na začátku 2. taktu. Přiřazovací příkaz s výrazem sčítání na pravé straně (např. DR := A + C;) nevyjadřuje, že se jedná o dvě μ-instrukce, které se mají zapsat do dvou řádků. Fázi vstupu do sčítačky budeme zapisovat výrazem registr $\rightarrow\sum$, např.:

```
A→∑L;  C→∑P;  v jednom řádku otevření RB4 a RB16 a
DR := A + C;  v druhém řádku (otevření RB36).
```

Druhý řádek je třeba číst: "přesun součtu A + C do DR".

V souladu s odst. 4.2.3 přiřazovací příkaz DR := A + C znamená, že jsme do formálně zapsaného příkazu

registr := ∑L + ∑P

dosadili skutečné parametry v pořadí DR, A a C.

Pořadí operandů ve výrazu na pravé straně je důležité: A je levý a C je pravý operand. Protože se mezery mezi příkazy v jednom řádku ignorují, budeme vždy použití sčítačky zapisovat pod sebou, např.

```
SP → Σ L;   -1 → Σ P;          /* RB2, RB12 */
    SP := SP + (-1);           /* RB34 */
A := DR;    ČI → Σ L;   +1 → Σ P;     /* RB23, RB3, RB14 */
                ČI := ČI + 1  ;   AR := RI(ADR)  /* RB35, RB21 */
```

Komentáře budou uzavřeny v závorkách /* a */.

Je-li třeba přesunout obsah jednoho registru do druhého registru, musíme přesun uskutečnit přes sčítačku. Např. přesun obsahu registru A do registru C zapíšeme:

```
A → Σ L;   0 → Σ P;
    C := A + 0 ;
```

Použití invertorů ovládaných RB28 a RB29 zapíšeme jako funkci; např. inverzi obsahu DR zapíšeme

```
INVERZE(DR) → Σ L; 0 → Σ P;  /* RB6, RB28, RB13 */
      DR := INVERZE(DR) + 0;      /*RB36 */
```

nebo vytvoření doplňkového kódu obsahu registru A a jeho přesun do D zapíšeme

```
INVERZE(A) → Σ L; + 1 → Σ P      /* RB4, RB28, RB14 */
     DR := INVERZE(A) + 1;       /* RB37 */
```

Použití posouvače ovládaného RB30 a RB31 zapíšeme také jako funkci. Je-li třeba z nějakého důvodu do DR zapsat např. celé číslo 4, zápis bude

```
1 → Σ L;   1 → Σ P;          /* RB7 a RB14 */
   DR := L_POSUV(1 + 1);     /* RB31 a RB36 */
```

nebo chceme-li obsah A dělit dvěma a výsledek vrátit do A

```
A → Σ L;   0 → Σ P                /* RB4 a RB13 */
    A := P_POSUV(A + 0)           /* RB 30 a RB33 */
```

4.6.3 μ-instrukci TEST_A_PODMINENY_SKOK zapíšeme pomocí příkazu větvení (rozhodovacího příkazu) podle vzoru:

```
if reg(k) = b then goto   návěští;
```

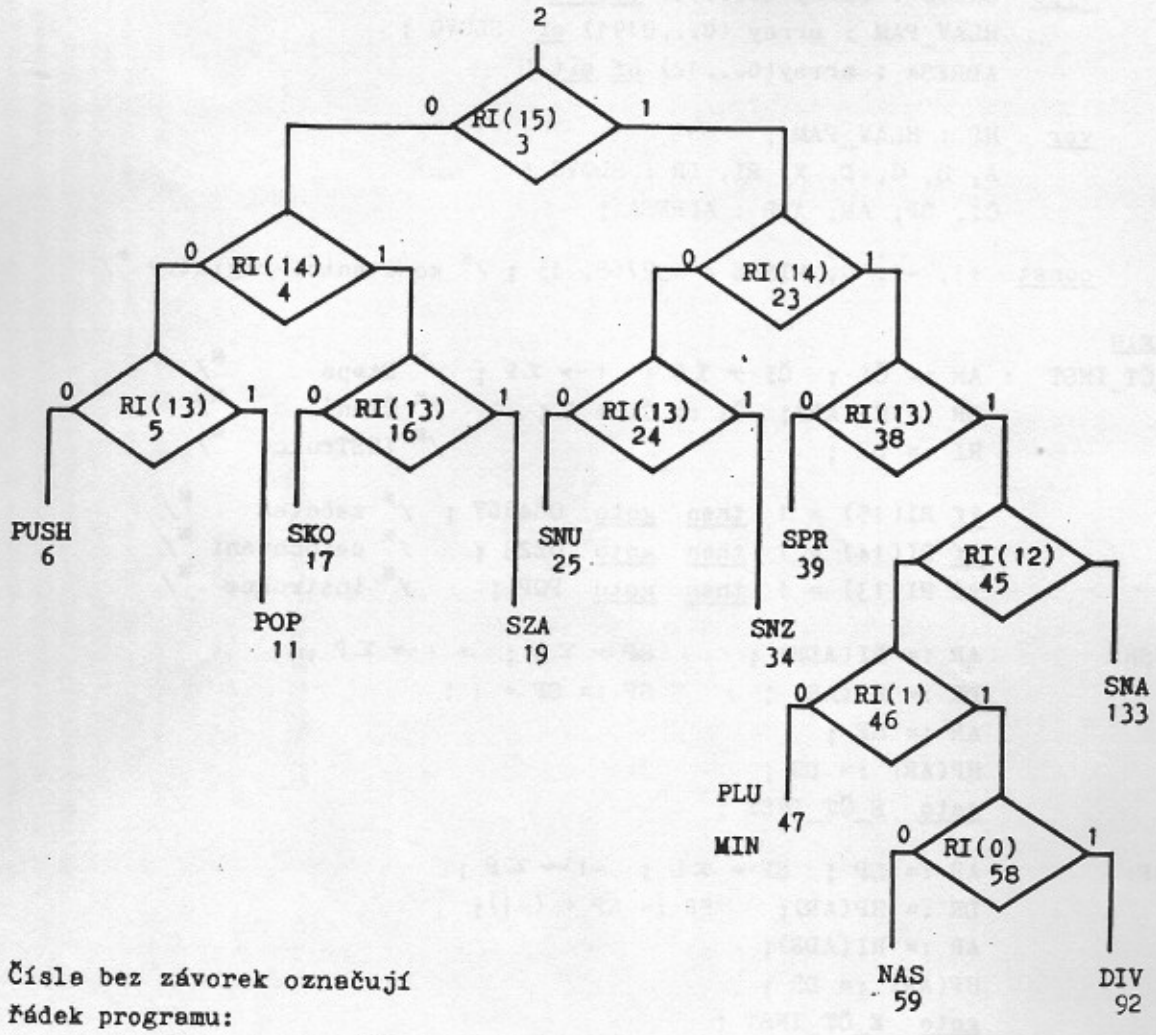
kde k je číslo řádu, který se má testovat v registru reg; hodnota b určuje hodnotu uloženou v řádě 25 μ-instrukce TEST_A_ ... . Každý řádek μ-programu může začínat návěštím ukončeným dvojtečkou ":". Např.

```
PQR: if  RI(13) = 1 then goto NUL;   /* test, zde obsah řádu 13 registru in- */
                                     /* strukce je =1; jestliže ano, skok    */
                                     /* na návěští NUL                       */
     if   X(15) = 0 then goto PQR;   /* test, zda obsah řádu 15 registru C   */
                                     /* je =0; jestliže ano, skok na návěští PQR */
```

Nepodmíněný skok je možno zapsat
goto návěstí; což je ekvivalentní zápisu: if 0(0) = 0 then goto návěstí;

3.7 INTERPRET PROGRAMU CÍLOVÉHO POČÍTAČE (CIKAP)

3.7.1 Na obrázku 4.7.1 je vývojový diagram dekódování OZ instrukcí strojového jazyka a na obr. 4.7.2 je úplný interpret programů napsaných v strojovém jazyce CIKAP. Dříve než budete pokračovat, prohlédněte si dobře oba obrázky. Při analýze interpretu programů použijte přílohu od str. 58.



Čísla bez závorek označují
řádek programu:
- buď adresu instrukce TEST_A_PODMINENY
- nebo začátek u-programu interpretujícího příslušný OZ jazyka CIKAP

Obr. 4.7.1 Vývojový diagram dekódování instrukce strojového jazyka cílového počítače

```
program  INTERPRET_CÍLOVÉHO_POČÍTAČE;
         /* Interpret programů nepsaných v strojovém jazyce nypotetického */
         /* cílového počítače úrovně 2 (CÍKAP). Interpretace se provádí   */
         /* na hostitelském mikropočítači úrovně 1 (HOMIP), který má 39   */
         /* řídících bodů, registry A, B, C, D, R... a jeho mikrojazyk má */
         /* dvě instrukce OTEVŘI_RB a TEST_A_PODMÍNENÝ_SKOK. Zápis algo-   */
         /* ritmu interpretu je v jazyce µJ.                               */

         type  SLOVO : array (0...15) of bit ;
               HLAV_PAM : array (0...8191) of  SLOVO ;
               ADRESA : array(0...12) of bit ;

         var   HP : HLAV_PAM ;
               A, B, C, D, X, RI, DR : SLOVO ;
               ČI, SP, AR, ADR : ADRESA ;

         const  +1, -1, 0, MINUS = -32768, 15 ; /* konstantní registry */

    begin
0   E_ČT_INST  : AR := ČI ;  ČI → ΣL ;  1 → ΣP ;  /* Etapa      */
1                DR := HP(AR);  ČI := ČI + 1 ;     /* ČTení      */
2                RI := DR ;                        /* INSTrukce  */

3                if RI(15) = 1  then  goto  OZ4567 ;  /* začátek    */
4                if RI(14) = 1  then  goto  OZ23 ;    /* dekódování */
5                if RI(13) = 1  then  goto  POP ;     /* instrukce  */

6   PUSH:        AR := RI(ADR) ;       SP → ΣL ;  + 1 → ΣP ;
7                DR := HP(AR) ;        SP := SP + 1 ;
8                AR := SP ;
9                HP(AR) := DR ;
10               goto  E_ČT_INST ;

11  POP:         AR := SP ;  SP → ΣL ;  -1 → ΣP ;
12               DR := HP(AR);    SP := SP + (-1);
13               AR := RI(ADR);
14               HP(AR) := DR ;
15               goto  E_ČT_INST ;

16  OZ23         if RI(13) = 1  then  goto  SZA ;

17  SKO:         ČI := RI(ADR);
18               goto  E_ČT_INST ;

19  SZA:         AR := SP ;  SP → ΣL ;  -1 → ΣP ;
20               DR := HP(AR);    SP := SP + (-1) ;
21               if DR(15) = 1  then  goto  SKO ;
22               goto  E_ČT_INST ;
```

Obr. 4.7.2 Interpret (začátek)