Princípy počítačov
Letný semester
2010 / 2011

Doc. RNDr. Daniel Olejár, CSc.

RNDr. Richard Ostertág, PhD.

Obmedzenia na použitie týchto prezentácií

Tieto prezentácie vytvorili doc. RNDr. Daniel Olejár, PhD., a RNDr. Richard Ostertág, PhD. na základe informácií uvedených v zozname literatúry a internetových zdrojov pre prednášku z Princípov počítačov pre 1. ročník odboru Informatika na Fakulte matematiky, fyziky a informatiky Univerzity Komenského v Bratislave. Prezentácie sú vystavené na webovskej stránke Katedry informatiky FMFI UK, priebežne upravované a aktualizované. Študenti si môžu prezentácie stiahnuť a vytlačiť pre vlastnú potrebu. Akékoľvek šírenie, zverejňovanie týchto prezentácií alebo ich častí, ich používanie na iné ako na študijné účely, neautorizovaná modifikácia a iné manipulácie s textami a obrazovým materiálom, ktoré prezentácie obsahujú, sú zakázané a autori v prípade porušenia týchto pravidiel odstránia prezentácie z voľne dostupnej webovej stránky a budú uplatňovať svoje práva v zmysle autorského zákona.

Cieľ prednášky

- Oboznámiť poslucháčov s tým, ako principiálne funguje počítač
- Na čo to je dobré:
  - Poznanie možností a obmedzení súčasných počítačov a vybudovanie vedomostí potrebných pre pochopenie budúcich počítačov
  - Pochopenie zmyslu a vzájomných vzťahov ďalších predmetov (operačné systémy, siete, programovanie,...)
  - Doplnenie neúplných poznatkov a neznámych pojmov (všeobecné informatické vzdelanie)

Stručný obsah prednášky 1/2

1. Úvod (čo je počítač, počítač ako systém pozostávajúci z niekoľkých virtuálnych strojov, vzťahy medzi jednotlivými virtuálnymi strojmi)
2. Základná organizácia počítača von Neumannovského typu a princípy jeho činnosti (CPU, pamäť, I/O, zbernice, inštrukcie a operácie, vykonávanie programov, RTL)
3. Zjednodušený model počítača (Simplified Instructional Computer, SIC)
4. Central Processing Unit, CPU, jej štruktúra a funkcie
5. Mikroprogramovanie (princíp, mikroprogramovaná CLU, formáty mikroinštrukcií, nanoprogramovanie)
6. RISC a CISC (zdôvodnenie, princípy a porovnanie)

Stručný obsah prednášky 2/2

7. Spracovanie vstupu/výstupu (I/O systém, riadenie I/O, prístup k I/O portom, DMA, princípy operácií prenosu údajov, pripojenie periférnych zariadení, rozhrania, I/O procesory)
8. Pamäť (funkcia pamäte, operačná a pomocná pamäť, parametre pamätí, zásobníková pamäť, modulárna pamäť, asociatívna pamäť, cache, virtuálna pamäť, technológie realizácie pamätí)
9. Zbernice (funkcia, princíp fungovania, typy, prideľovanie zbernice)
10. Paralelné počítače (dôvody pre paralelizmus, klasifikácia paralelených počítačov, aplikácie paralelizmu: pipelining, vektorové počítače, multiprogramming, multiprocesoring)
11. Futuristické koncepty

Ako študovať princípy počítačov?

- Na prednáškach
  - Podstatné informácie (princípy, základné pojmy)
  - Doplňujúce a ilustrujúce informácie (konkrétne systémy, technické parametre)
  - Jednoduché (ale podstatné) poznatky budú v textoch, ale nebudú sa prednášať
  - Explicitne povieme, čo je potrebné doštudovať (a odkiaľ)
- Texty prednášok budú na webe (a časom možno aj doplňujúce materiály k jednotlivým témam)
- Rozširujúce informácie (literatúra, Internet)
- Skúšajú sa základné (nie rozširujúce) znalosti

Literatúra

1. Langholz
2. L.H.Pollard *Computer Design and Architecture*, Prentice Hall 1990
3. W.D.Murray *Computer and Digital System Architecture,* Prentice Hall,1990
4. J.L.Hennessy, D.A.Patterson *Computer Architecture a Quantitative Approach,* Morgan Kaufann Publishers, 1990
5. A.S.Tanenbaum *Structured Computer Organization,* 3-rd ed.,Prentice Hall, 1990
6. F.G.Soltis *Systém AS/400 zevnitř,* Computer Press 1997
7. Internetové zdroje a firemná literatúra

1. Úvod

- Počítač = zariadenie na spracovanie informácie
  - Informácia je zapísaná v podobe „textu“ nad nejakou abecedou (symbolicky kódovaná)
- Spracovanie prebieha na základe presne stanoveného postupu (programu) a dá sa popísať ako postupnosť transformácií vstupnej informácie na výstupnú
- V súčasných počítačoch transformácie v konečnom dôsledku realizujú elektrické (logické) obvody
- Zadať vstup tak, aby mohol byť priamo spracovaný výkonnými obvodmi je náročné (človek a logický obvod používajú veľmi rozdielne jazyky)
- Preto sa medzi používateľa a výkonné obvody „vkladajú“ ďalšie úrovne, umožňujúce postupne transformovať úlohu do podoby zrozumiteľnej výkonným obvodom

1.1. Počítač ako systém virtuálnych strojov 1/2

- Formulácia problému: človek A formuluje úlohu v jazyku L_1. Ak systém B „rozumie“ jazyku L_1, môže úlohu priamo riešiť. Ak však B „rozumie“ len jazyku L_2, úlohu v jazyku L_1 treba preformulovať do jazyka L_2
- Dva prístupy: preklad (celý program v L_1 sa preloží do jazyka L_2 a potom vykoná) a interpretácia (jednotlivé inštrukcie programu v L_1 sa priebežne prekladajú do L_2 a vykonávajú)
- Medzistupňov môže byť viac: L_1 - L_2 - L_3 - ... - L_n
- Na ľubovoľný stupeň (k) sa možno pozerať ako na samostatný virtuálny stroj, ktorý transformuje vstup v jazyku L_k na výstup v jazyku L_{k+1}
- Tento prístup umožňuje sústrediť sa na riešenie na danom stupni a abstrahovať od vyšších a nižších stupňov: problém sa dá lepšie štrukturalizovať a (často potom aj) jednoduchšie riešiť

1.1. Počítač ako systém virtuálnych strojov 2/2

- Pôvodné počítače – 2 úrovne
  (človek, výkonný hardware)
- Súčasné počítače majú (Tanenbaum) 6 a viac úrovní:

1. Digital logic level (úroveň logických obvodov)
2. Microprogramming level (mikroprogramová úroveň)
3. Conventional machine level
4. Operating system machine level (úroveň operačného systému)
5. Assembly language level
   (úroveň jazyka symbolických adries, assemblera)
6. Problem-oriented language level
   (úroveň vyšších programovacích jazykov)

1.2. Digital logic level (úroveň logických obvodov)

- Logické obvody sú skonštruované z hradiel (gates), fyzikálnych systémov schopných realizovať elementárne logické funkcie
- Vstupy a výstupy logických obvodov sú reprezentované pomocou elektrických signálov
- Priamo vykonateľný program v podobe textovo zapísanej postupnosti príkazov pre logické obvody neexistuje
- Logické obvody však pomocou hardvérovo realizovanej riadiacej jednotky môžu vykonávať aj postupnosť transformácií (informácie) – napr. násobenie, delenie;
- Štúdium logických obvodov bolo náplňou prvej polovice semestra
- Bolo by možné ísť ešte o úroveň nižšie – na úroveň fyzikálnych procesov prebiehajúcich v logických obvodoch
- Budeme sa zaoberať virtuálnymi strojmi nad úrovňou logických obvodov

1.3. Microprogramming level (mikroprogramová úroveň)

- Program (mikroprogram), ktorého úlohou je interpretovať inštrukcie 3. úrovne
- Mikroinštrukcie sú priamo vykonateľné pomocou logických obvodov (mikroinštrukcii zodpovedá mikrooperácia)
- Súbor mikroinštrukcií = „strojový kód“ počítača
- Existujú počítače, ktoré nemajú mikroprogramovú úroveň (RISC)

1.4. Conventional machine level

- Zle definovaná úroveň
- Hybridná úroveň (inštrukcie jazyka tejto úrovne môžu byť aj na úrovni mikroprogramu aj operačného systému)
- Aj nové inštrukcie
- Odlišná organizácia pamäte
- Možnosť súčasného behu viacerých programov
- Inštrukcie tejto úrovne sa interpretujú pomocou mikroprogramu

1.5. Operating system machine level (úroveň operačného systému)

- Principiálny rozdiel oproti predchádzajúcim trom úrovniam
- Tri najnižšie úrovne nie sú určené pre programátorov, ale pre systémových programátorov (špecialisti na implementáciu virtuálnych strojov)
- Jazyky predchádzajúcich úrovní boli binárne
- Pôvodne mal operačný systém nahradiť činnosť operátora, neskôr sa ukázalo, že na tejto úrovni sa dajú riešiť aj iné problémy (time sharing, práca s pamäťou a inými zdrojmi)

1.6. úroveň jazyka assemblera

- Na tejto úrovni sa používa jazyk symbolických adries
- Program v jazyku symbolických adries sa preloží pomocou assemblera do jazyka nižšej úrovne

1.7. Problem-oriented language level
(úroveň vyšších programovacích jazykov)

- Programovacie jazyky vyššej úrovne ako napríklad:
  - C++, Pascal, Basic, Java, Smalltalk
  - C#, Lisp, Haskell, APL
- Prekladajú sa do jazykov nižšej úrovne alebo interpretujú

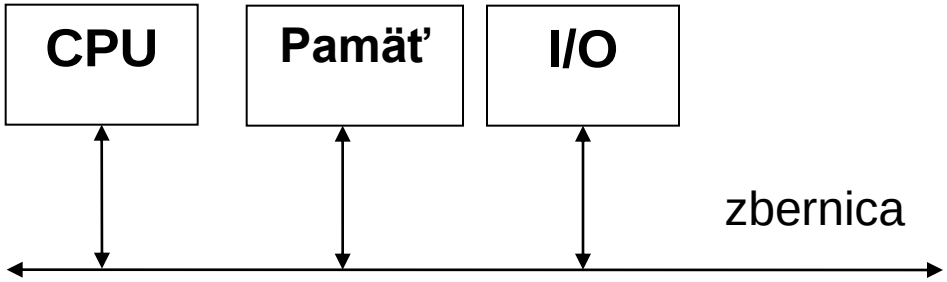
Ktorými úrovňami sa budeme zaoberať?

- Na tejto prednáške sa budem zaoberať dvoma úrovňami:
  - Mikroprogramovou
  - Úrovňou konvenčného stroja
- Úroveň digitálnych obvodov sme preberali v prvom semestri
- Operačným systémom, programovaniu a aplikačnému programovaniu sú venované samostatné prednášky
- Úroveň jazyka assemblera – na prednáške z kompilátorov alebo operačných systémov

2. Základná organizácia počítača von Neumannovského typu

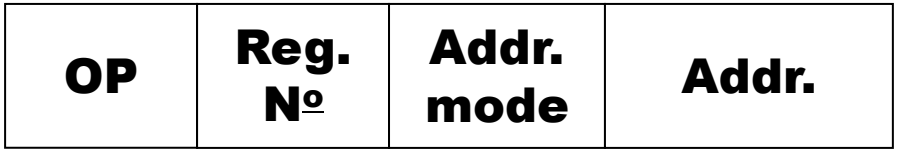
- Digitálny počítač s uloženým programom spracováva údaje na základe usporiadanej postupnosti inštrukcií
  - Hardvér
  - Softvér
- Základné časti počítača (Hardvér)
  - Centrálny procesor, *central processing unit*, *CPU* (riadi činnosť počítača a vykonáva inštrukcie)
  - Pamäť, *memory* (ukladanie a uchovávanie informácie – aj programu aj údajov)
  - vstupno/výstupné zariadenia, *Input/output, I/O* (umožňujú počítaču komunikovať s prídavnými zariadeniami a prostredníctvom nich s okolím – klávesnica, monitor, tlačiareň, ...)
  - Zbernice, *buses* (spájajú jednotlivé subsystémy počítača)

2. Základná organizácia počítača von Neumannovského typu - schéma



2.1. Inštrukcie počítača

- Budeme skúmať počítač na úrovni mikroprogramovej a konvenčného stroja
- Organizácia počítača je určená množinou inštrukcií, ktoré počítač vykonáva
- Inštrukcia = binárny vektor, ktorý slúži na označenie operácie
- Všetky inštrukcie, ktoré je schopný počítač vykonať = inštrukčný súbor počítača
- Rozdelenie inštrukcie na časti rozličného významu = formát inštrukcie
- Časti inštrukcie, ktoré majú istý význam = polia
  - Operačné pole (opcode): určuje operáciu, ktorá sa ma vykonať
  - Explicitný, implicitný operand
  - Adresné pole, adresný spôsob



2.1. Spracovanie inštrukcie (1)

- Inštrukcia je uložená v pamäti počítača
- Vykonanie operácie, ktorú inštrukcia určuje, sa nedá uskutočniť v jednom takte
  - Napr. sa musí preniesť z pamäte do registra
- Počítač vykonáva inštrukcie pomocou postupnosti elementárnych operácií, ktoré sa nazývajú mikrooperácie
- Mikrooperácie sú určené mikroinštrukciami
- Mikrooperácie sa dajú vykonať v jednom takte
- Postupnosti mikroinštrukcií sa nazývajú mikroprogramami (alebo CPU cyklami)
- Pri spracovaní inštrukcie sa uplatňujú nasledujúce CPU cykly:

2.1. Spracovanie inštrukcie (1)

- Pri spracovaní inštrukcie sa uplatňujú nasledujúce CPU cykly:
  - Fetch (získavanie inštrukcie z pamäte)
  - Address (dekódovanie adresy operandu)
  - Translation
  - Execute (vykonanie inštrukcie)
  - Interrupt (ošetrenie prerušenia)
- Mikrooperácie spôsobujú prenos údajov medzi registrami, preto ich výhodne možno popisovať pomocou RTL (register transfer language), ktorý sme používali pri návrhu digitálnych systémov

2.2. Register Transfer Language (1/4)

- Register = postupnosť pamäťových členov + obvody umožňujúce uložiť, preniesť a posunúť informáciu uloženú v pamäťových členoch
- Číslovanie bitov v registri je typu Big Endian: 0,1,...,n-1
- Zápis do registra

  B := (A)

  B_i := (A_i); i= 0,1,...,n-1

- Môžu sa prenášať aj časti registrov (polia)

  PC := IR[AD]

- Ak časť registra nemá meno, tak

  R1[0..3] := (X)

- Registrom sa dajú priraďovať konštanty

  L := 5

2.2. Register Transfer Language (2/4)

- Aritmetické operácie (kvôli jednoduchosti výsledok je uložený v registri, ktorého obsah je aj operandom operácie)

A3 := (A1) + (A2);	súčet
A := (A) + 1;	inkrementovanie obsahu
A := (A) - 1;	dekrementovanie
A := (~A);	logický doplnok
A := (~A) + 1;	binárny doplnok
A := (A) +(~B) + 1;	odčítanie A-B

- Aby sme ošetrili pretečenie použijeme 1-bitový register V:

VA3 := (A1) + (A2);

2.2. Register Transfer Language (3/4)

- Poznámky.
  - Operácie násobenia a delenia sa nedajú vykonať v jednom takte, a preto sa medzi základné mikroinštrukcie RTL nezaraďujú
  - Kvôli zvýšeniu čitateľnosti mikroprogramu sa pridávajú komentáre, oddelené bodkočiarkou od príslušnej mikroinštrukcie

- Logické operácie

C := (A) AND (B);	logický súčin
C := (A) OR (B);	logický súčet

- Operácie posunu

A := SL(A);	posun doľava
A := SR(A);	posun doprava
A := LCIR(A);	cyklický posun doľava
A := RCIR(A);	cyklický posun doprava

2.2. Register Transfer Language (4/4)

- Operácie presunu informácie medzi registrami a pamäťou (i-te pamäťové miesto označíme M[i])
- Čítanie z pamäte

  B := (M[(A)]); obsah pamäťového miesta, ktorého adresa je v registri A sa zapíše do registra B
- Zápis do pamäte

  M[(A)] := (B); do pamäťového miesta, ktorého adresa je v registri A sa zapíše obsah registra B
- Vykonávanie operácií sa niekedy viaže na splnenie nejakých podmienok:
  - Logické podmienky: **IF ... THEN ...**
  - Riadiace podmienky = logické funkcie definované na Booleovských premenných, ktoré riadia vstupy do registrov, napr.

    $t_0(c_1+c_2)$: X := (~A) + 1

3. Zjednodušený model počítača

- Fungovanie počítača demonštrujeme na zjednodušenom príklade (modeli) počítača

- Čím je určený model počítača:
  - Veľkosť pamäte
  - Veľkosť slova
  - Registre
  - Formát údajov
  - Formát inštrukcií
  - Spôsoby adresovania
  - Inštrukčný súbor
  - Obmedzenia na I/O

3.1. SIC (Simplified Instructional Computer)

- Štruktúra SIC:
  - CPU
  - Pamäť
  - Aspoň jedno I/O zariadenie
- Pamäť:
  - 2^{15} slov
  - Slovo dĺžky 24 bitov
  - Prístup do pamäte cez registre MAR a MBR
  - Zápis:

    M[(MAR)]:=(MBR)
  - čítanie:

    MBR:=(M[(MAR)])

3.1. Registre SIC

Register	Bitov	Popis
A	24	accumulator
X	15	index register
PC	15	program counter (adresa naled. inštrukcie)
L	15	linkage register (návratová adresa podprog.)
IR	24	instruction register
MBR	24	buffer pamäte
MAR	15	register adries pamäte
SW	11	status word
C	2	counter (generuje signály: t_0, t_1, t_2, t_3)
INT	1	interrupt flag
E	1	execute cycle flag
F	1	fetch cycle flag
S	1	start/stop flag (spúšťa C)

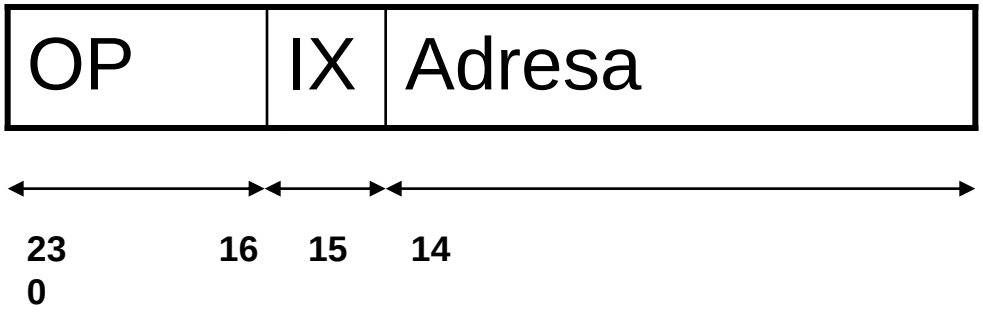
3.1.Pamäť a registre SIC

- Registre, do ktorých sa ukladá obsah pamäťového miesta majú dĺžku 24 bitov (A, IR, MBR)
- Registre, ktoré môžu obsahovať adresu pamäťového miesta, majú dĺžku 15 bitov (MAR, X, PC, L)
- Na zápis stavu SIC po vykonaní inštrukcie stačí 11 bitový SW
- C generuje hodinové signály, ktoré určujú takty SIC
- Pamäť je asynchrónna RAM (hodnoty MAR a MBR musia byť počas zápisu stabilné)
- Zápis do pamäte a čítanie z pamäte trvá kratšie ako jeden takt
- Vstup a výstup do/z CPU: 1 byte/takt;
  výnimka – prenos údajov z/do pamäte
- Prenos z/do 8 najpravejších bitov akumulátora A na/z I/O zariadenia
- Každé zariadenie SIC má 8 bitový kód

3.1. Údaje a inštrukcie SIC

- Údaje SIC:
  - Celé čísla - 24 bitové, binárny doplnkový kód
  - Znaky - 8 bitové, ASCII kód

- Formát inštrukcií SIC-u: (číslovanie bitov)



3.1. Inštrukcie SIC

- OP = operačný kód určujúci inštrukciu, dĺžka poľa = 8 bitov
- IX = flag (príznak) spôsobu adresovania
- Adresa = adresa operandu, 15 bitov
  - Ak IX = 0, priame adresovanie, operand je uložený v M[AD]
  - Ak IX = 1, nepriame (indexové) adresovanie,
    operand je uložený v M[AD+(X)]

- Uvažuje sa len jeden operand, druhým je implicitne obsah akumulátora A

3.2. Súbor inštrukcií SIC (1/4)

Inštrukcia	Skratka	OP	Popis (skoro v RTL)
Add	ADD m	00	A:=(A)+(m)
And	AND m	01	A:=(A) AND (m)
Compare	COMP m	02	(A):(m); CC:=výsledok
Jump	J m	03	PC:=m
Jump Equal	JEQ m	04	PC:=m if CC = “=“
Jump Greater T.	JGT m	05	PC:=m if CC = “>“
Jump Less T.	JLT m	06	PC:=m if CC = “<“
Jump Subrout.	JSUB m	07	L:=(PC), PC:=m
Load A	LDA m	08	A:=(m)
Load L	LDL m	09	L:=(m)
Load X	LDX m	0A	X:=(m)
Or	OR m	0B	A:=(A) OR (m)

3.2. Súbor inštrukcií SIC (2/4)

Inštrukcia	Skratka	OP	Popis (skoro v RTL)
Read device	RD m	0C	A[0..7]:=byte from device (m)
Return Subroutine	RSUB m	0D	PC:=(L)
Store A	STA m	0E	m:= (A)
Store L	STL m	0F	m:= (L)
Store SW	STSW m	10	m:= (SW)
Store X	STX m	11	m:= (X)
Subtract	SUB m	12	A:=(A)-(m)
Test device	TD m	13	test device (m) CC:=result
Write device	WD m	14	device(m):=(A[0..7])
Interrupt Return	IRT	15	PC:=(M[0]) SW[MASK]:=0

3.2. Súbor inštrukcií SIC (3/4)

Poznámky:

- m predstavuje adresu v pamäti
- (m) je obsah pamäťového miesta číslo m
- M[0] je prvé pamäťové miesto
- **JSUB**: prechod na vykonávanie podprogramu
  - Ukladá sa návratová adresa L:=(PC)
  - Nastavuje sa začiatok podprogramu PC:= m
- **RSUB**: návrat z podprogramu
  - Nastaví sa pôvodná hodnota PC PC:=(L)
- **TD** sa používa na testovanie I/O zariadenia predtým, ako sa z neho bude čítať, alebo sa naň bude zapisovať. Výsledok testovania sa zapíše do dvojbitového poľa CC registra SW.

3.2. Súbor inštrukcií SIC (4/4)

- **CC (condition code)** sa používa aj pri porovnaniach čísel (inštrukcia COMP), preto jeho 2 bity reprezentujú 3 hodnoty (<,=,>). Výsledok testovania zariadenia je:
  - Zariadenie je pripravené **<**
  - Zariadenie je obsadené **=**
  - Zariadenie nie je v prevádzke **>**
- **IRT (interrupt return)** túto inštrukciu využíva program na ošetrenie prerušení (interrupt handler) na návrat na tú inštrukciu programu, ktorú CPU spracovával v okamihu, keď došlo k prerušeniu. Okrem obnovenia pôvodného obsahu PC sa vynuluje príznakový bit prerušenia (MASK) v SW.
- Inštrukcia IRT je prístupná len pre systémový softvér a nie pre programy v jazyku assembler

3.2. Súbor inštrukcií SIC použitie TD

Nasledujúci program demonštruje použitie TD. Program

- Neustále testuje, či je zariadenie x pripravené.
- Ak je, prečíta z neho 1 byte a skončí,
- Ak je zariadenie x vypnuté alebo nefunkčné, vyhlási chybu

```
TEST    TD      “x”         ; testuje zariadenie x
        JEQ     TEST        ; ak je obsadené, testuj    znova
        JGT     ERROR       ; chyba – ohlási to
        RD      “x”         ; načítaj
        J       END         ; ukončí činnosť
ERROR   JSUB    EROUTINE    ; vyhlási chybu
END     STOP
```

3.3. SIC – Timing & Control (1/9)

- Všetky činnosti prebiehajúce v SIC trvajú nejaký čas. Predpokladáme, že čas je diskrétny a najmenšia jednotka času je 1 takt.
- CPU SIC vykonáva činnosť na základe spracovania postupnosti inštrukcií
- Spracovanie inštrukcie pozostáva z nasledujúcich činností:
  - Prečítanie inštrukcie z pamäte a uloženie do registra (**fetch**)
  - Vykonanie inštrukcie (**execute**)
- Počas spracovávania inštrukcie môže dôjsť k **žiadosti o prerušenie**: vtedy CPU musí
  - Zaregistrovať žiadosť o prerušenie
  - Dokončiť začatú činnosť
  - Uchovať informácie potrebné pre pokračovanie v činnosti po návrate z prerušenia
  - Odovzdať riadenie interrupt handleru

3.3. SIC – Timing & Control (2/9)

- Činnosť počítača prebieha v cykloch
- Cyklus je štandardný mikroprogram
- **Cykly SIC**
  - Fetch
  - Execute
  - Interrupt
- SIC má jednoduché adresovanie a inštrukčný súbor a preto nepotrebuje cykly address a translate
- CPU obsahuje 2-bitový register **C (counter),** ktorý sa v každom takte inkrementuje (mod 4)
- Counter C generuje **postupnosť riadiacich signálov**
  t_0,t_1,t_2,t_3: 00,01,10,11
- Riadiace signály countra C sa používajú na zaistenie správneho poradia vykonávania mikroinštrukcií
- Counter je ovládaný S-flagom. Ak S=1, C sa v každom takte inkrementuje, ak S=0, C je zablokovaný.

3.3. SIC – Timing & Control (3/9) Fetch cyklus

- Všetky cykly SIC sa dajú riadiť pomocou riadiacich signálov C
- Na rozlíšenie jednotlivých cyklov sa používajú E, F flagy:

F E	cyklus
0 1	execute
1 0	fetch
0 0	interrupt
1 1	nepoužitý

- **Cyklus FETCH**

FE't_0:	MAR:=(PC)
FE't_1:	MBR:=(M[(MAR)])
	PC:=(PC)+1
FE't_2:	IR:=(MBR)
FE't_3:	If IR[IX]=1 then MAR:=(IR[AD])+(X)
FE't_3:	If IR[IX]=0 then MAR:=(IR[AD])
FE't_3:	E:=1
	F:=0

3.3. SIC – Timing & Control (4/9) Interrupt cyklus

- Počas vykonávania programu môže dôjsť k udalosti, ktorá si vyžiada jeho prerušenie
- Čo sa bude diať:
  - Žiadosť o prerušenie (CPU, I/O zariadenie) signalizuje nastavením registra INT:=1
  - Uloží sa obsah PC na stanovené miesto v pamäti (M[0])
  - Odovzdá sa riadenie interrupt handleru (PC:= (M[1]))
  - Nastavia sa príznakové bity v SW
    - MASK:=1 (aby nedochádzalo k vnoreným prerušeniam)
    - Zariadenie, ktoré iniciovalo prerušenie nastaví pole ICODE na hodnotu, ktorá špecifikuje dôvod prerušenia

3.3. SIC – Timing & Control (5/9)
Interrupt cyklus

Cyklus INTERRUPT

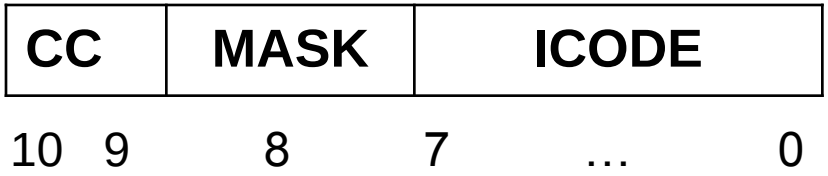
F'E't$_0$: MBR[AD]:=(PC)
INT:=0
SW[MASK]:=1

F'E't$_1$: MAR:=0
PC:=1

F'E't$_2$: M[(MAR)]:=(MBR)

F'E't$_3$: F:=1

Status Word (SW) register (11 bitový)



3.3. SIC – Timing & Control (6/9)
Execute cyklus

- Každý výkonný cyklus je jedinečný, lebo sa spracovávajú rozličné inštrukcie
- K riadiacim signálom pridáme identifikátor inštrukcie, ktorá sa v danom execute cykle vykonáva: $\mathbf{P_i}$
- Výkonný cyklus
  - Na začiatku má k dispozícii adresu operandu
  - Musí vykonať príslušnú inštrukciu
  - V poslednom kroku nastaví buď fetch alebo interrupt cyklus:

$F'Et_3$: E:=0

If (SW[MASK] OR INT=0) then F:=1

3.3. SIC – Timing & Control (7/9)
Execute cycles ADD, J

ADD

$P_0F'Et_0$: MBR:=(M[MAR])

$P_0F'Et_1$: A:=(A)+(MBR)

$P_0F'Et_2$:

$P_0F'Et_3$: E:=0

If (SW[MASK] OR INT = 0) then F:=1

J

$P_3F'Et_0$: PC:=(IR[AD])

$P_3F'Et_1$:

$P_3F'Et_2$:

$P_3F'Et_3$: E:=0

If (SW[MASK] OR INT = 0) then F:=1

3.3. SIC – Timing & Control (8/9)
Execute cycles LDL, JSUB

LDL

$P_9F'Et_0$: MBR:=(M[(MAR)])

$P_9F'Et_1$: L:=(MBR[AD])

$P_9F'Et_2$:

$P_9F'Et_3$: E:=0

If (SW[MASK] OR INT = 0) then F:=1

JSUB

$P_7F'Et_0$: MBR:=(M[(MAR)])

$P_7F'Et_1$: L:=(PC)

$P_7F'Et_2$: PC:=(MBR[AD])

$P_7F'Et_3$: E:=0

If (SW[MASK] OR INT = 0) then F:=1

3.3. SIC – Timing & Control (9/9)
Execute cycle RSUB

RSUB

$P_{13}F'Et_0$: PC:=(L)

$P_{13}F'Et_1$:

$P_{13}F'Et_2$

$P_{13}F'Et_3$: E:=0

If (SW[MASK] OR INT = 0) then F:=1

3.4. Funkcionálne jednotky SIC

- Slúžia na vykonávanie operácií potrebných na vykonávanie niektorých inštrukcií:
  - Sčítačka pre ADD
  - Sčítačka pre indexové adresovanie
  - Porovnávací obvod
  - AND
  - OR
- Vstupy funkcionálnych jednotiek: MBR, A
- Výstupy funkcionálnych jednotiek: MAR, A

3.5. Štart systému SIC

Startup SIC

- S:=1, F:=1, E:=0, C:=0
- SW[ICODE]=“startup”
- PC:=1

Manuálny start/stop switch:

- Nastavenie 1: spustí sa startup
- Nastavenie 0: S sa nastaví na 0

3.6. Trasovanie inštrukcií SIC počiatočný stav (1/4)

- Popis CPU cyklov, riadiaceho a časového mechanizmu, výkonných obvodov určuje architektúru počítača SIC
- Pozrieme sa na vykonávanie fragmentu programu

  STA 0 19, ADD 1 20, ...
- Stav pamäte a registrov pred vykonaním **STA 0 19**:

9	STA 0 19	(C)=11	(F)=1	(E)=0
10	ADD 1 20	(PC)=9	(X)=5	(INT)=0
	…	(A)=101	(MAR)=19	(MBR)= 101
19	+101	(IR) = STA 1 10		
20	+100			
	…			
25	+50			

3.6. Trasovanie inštrukcií SIC
Fetch STA 0 19 (2/4)

C	F	E	PC	X	INT	A	MAR	IR	MBR
			9	5	0	101	19	STA 1 10	101
00	1	0	9	5	0	101	**9**	STA 1 10	101
01	1	0	**10**	5	0	101	9	STA 1 10	**STA 0 19**
10	1	0	10	5	0	101	9	**STA 0 19**	STA 0 19
11	**0**	**1**	10	5	0	101	**19**	STA 0 19	STA 0 19

Fetch STA 0 19

…

Execute STA 0 19

3.6. Trasovanie inštrukcií SIC
Fetch, Execute ADD 1 20 (3/4)

C	F	E	PC	X	INT	A	MAR	IR	MBR
	1	0	10	5	0	101	19	STA 0 19	STA 0 19
00	1	0	10	5	0	101	**10**	STA 0 19	STA 0 19
01	1	0	**11**	5	0	101	10	STA 0 19	**ADD 1 20**
10	1	0	11	5	0	101	10	**ADD 1 20**	ADD 1 20
11	**0**	**1**	11	5	0	101	**25**	ADD 1 20	ADD 1 20
00	0	1	11	5	0	101	25	ADD 1 20	**50**
01	0	1	11	5	0	**151**	25	ADD 1 20	50
10	0	1	11	5	0	151	25	ADD 1 20	50
11	**1**	**0**	11	5	0	151	25	ADD 1 20	50

Fetch ADD

Execute ADD

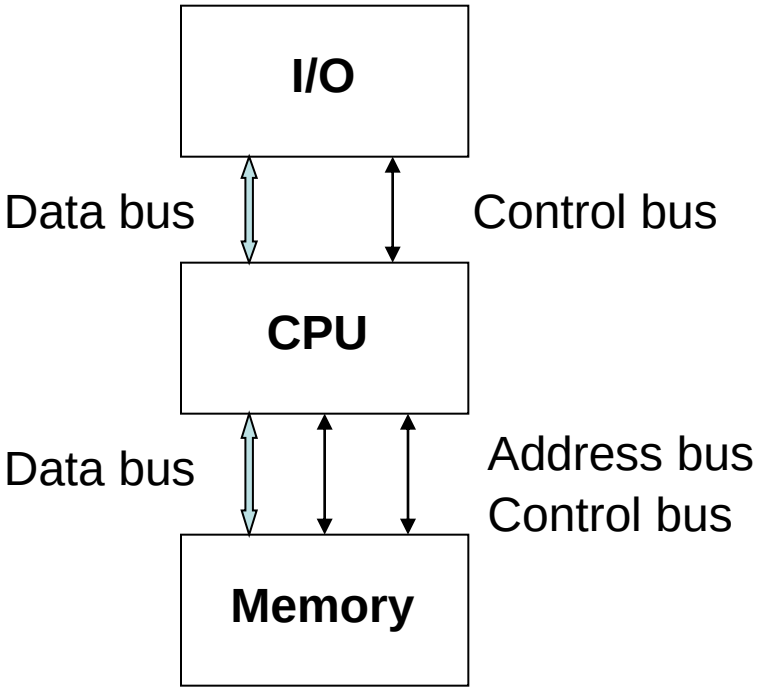
3.6. Trasovanie inštrukcií SIC ošetrenie prerušenia (4/4)

- Žiadosť o prerušenie sa signalizuje nastavením INT:=1
- Najprv sa dokončí fetch a execute cyklus
- Po ukončení execute cyklu inštrukcie ADD sa začne spracovávať prerušenie
- Predpokladáme, že prerušenie nie je maskované; t.j. (SW[MASK])=0

C	F	E	PC	X	INT	A	MAR	IR	MBR	SW [MASK]
11	0	0	11	5	1	151	25	AD 1 20	50	0
00	0	0	11	5	**0**	151	25	AD 1 20	**11**	**1**
01	0	0	**1**	5	0	151	**0**	AD 1 20	11	1
10	0	0	1	5	0	151	0	AD 1 20	11	1
11	**1**	0	1	5	0	151	0	AD 1 20	11	1

INTERRUPT CYCLE

4. Central Processing Unit, CPU



- Jedna zo základných častí počítača
- Program je uložený v hlavnej pamäti
- CPU číta z pamäte inštrukcie a údaje
- Zapisuje do pamäte
- Podobne komunikuje CPU s I/O, synchronizácia je zabezpečená pomocou riadiacej zbernice

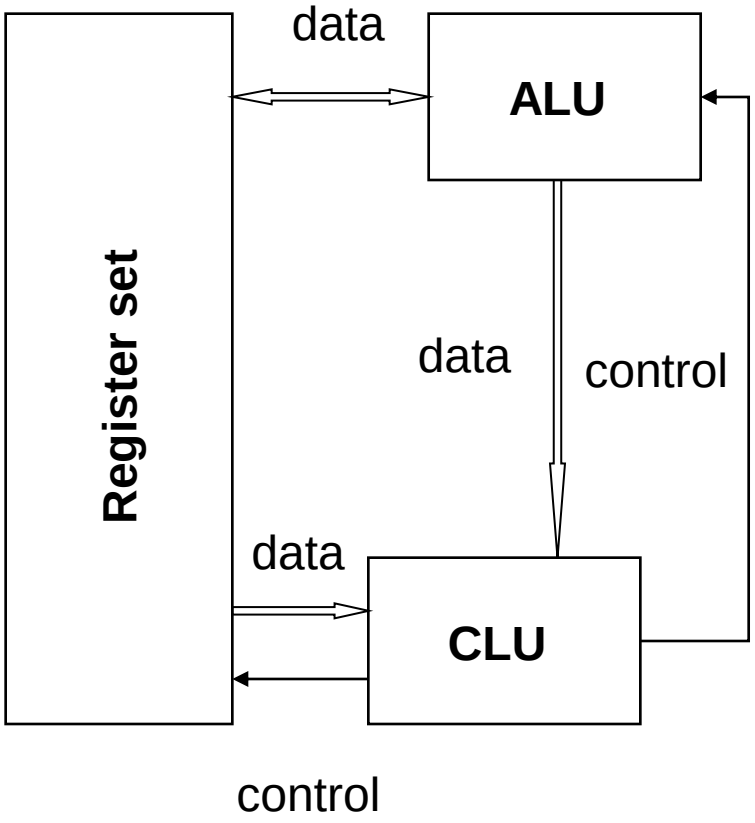
4. Central Processing Unit, CPU vykonávanie programu

1. CPU získava informácie z pamäte (fetch)
2. CPU dekóduje inštrukciu
3. Podľa toho, o akú inštrukciu ide, CPU prípadne generuje riadiace impulzy na získanie (fetch) ďalšieho operandu a potom na vykonanie
   (a) Aritmetickej alebo logickej operácie
   (b) Uloženie výsledku do pamäte
   (c) Čítanie alebo zápis info do/zo I/O
4. CPU sa vracia do bodu 1 a opakuje postupnosť 1-4 dovtedy, kým sa program neukončí

4. Central Processing Unit, CPU poznámky k vykonávaniu programov

- CPU je schopný vykonávať istú množinu elementárnych činností, ktoré sa nazývajú mikrooperácie
- Mikrooperácie sú realizované pomocou hardvérových zariadení (*)
- Mikrooperácie popisujeme pomocou mikroinštrukcií
- Zmysluplné postupnosti mikroinštrukcií tvoria mikroprogramy
- Používateľ programuje vo vyššom programovacom jazyku, jeho program sa prekladá do tzv. strojového kódu, inštrukcie strojového kódu (machine instruction) sa realizujú pomocou mikroprogramov (*)
- Mikroprogramy na vykonávanie strojových inštrukcií sa píšu počas návrhu počítača a závisia od interpretácie strojových inštrukcií

4.1. Hlavné časti CPU



1. Register set
2. Arithmetic and logic unit (ALU)
3. Control and logic unit (CLU)

4.1. Základné funkcie hlavných častí CPU

- Registre slúžia na uchovávanie informácie, s ktorou bezprostredne pracuje CPU (údaje, inštrukcie, adresy, riadiace údaje, hodnoty operendov a výsledkov operácií, príznaky,...)
- ALU používa hodnoty uložené v registroch na vykonávanie aritmetických a logických operácií
- CLU riadi systém dvoma spôsobmi
  - Usmerňovaním prenosov informácie medzi ALU, registrami, pamäťou a I/O
  - Inštruovaním ALU, ktoré operácie má vykonať

4.2. Register set (množina registrov) (1)

- Množina registrov závisí od konkrétneho počítača
- Registre môžu mať rozličnú veľkosť (spomeň si na SIC)
- konštrukčne: registre s paralelným čítaním a zápisom, posuvné registre
- Niektoré registre sa štandardne používajú v skoro všetkých univerálnych počítačoch
- **Program counter register** (PC, CI)
  - Slúži na ukladanie adresy nasledujúcej inštrukcie, ktorá sa má vykonávať
  - Musí sa dať inkrementovať o +1
  - Musí sa doň dať uložiť adresa (číslo) pri operácii skoku
  - PC je binárny counter s paralelným ukladaním stavu

4.2. Register set (množina registrov) (2)

- Instruction register (IR, register inštrukcií)
  - Slúži na uchovávanie inštrukcie, ktorá sa práve vykonáva
  - Dôvody:
    - na vykonanie inštrukcie bude možno potrebné čítať niekoľkokrát z pamäte; ak by vykonávaná inštrukcia bola uložená v MBR, pri ďalšom čítaní z pamäte by sa stratila
    - Spracovanie inštrukcie si vyžaduje špeciálny hardvér (napr. na dekódovanie operačného kódu)
- **Program Status Word (PSW)**
  - Obsahuje informáciu o stave CPU počas posledného execute cycle
  - Aritmetické operácie (prenos, pretečenie, znamienko)
  - Interrupt flags
  - Pri prerušení sa môže použiť na ukladanie stavu bežiaceho programu

4.2. Register set (množina registrov) (3)

- **Memory address register** (MAR, register adries pamäte)
  - Slúži na ukladanie adresy pamäťového miesta, s ktorým sa má pracovať
  - Jeho veľkosť je určená veľkosťou pamäte, ktorú má adresovať
  - Register s paralelným zápisom a čítaním
- **Memory buffer register** (MBR, vyrovnávací register pamäte)
  - Prostredníctvom neho sa čítajú údaje z pamäte a zapisujú sa do pamäte
  - Jeho veľkosť je rovnaká ako veľkosť pamäťového miesta
  - Register s paralelným zápisom a čítaním

4.2. Register set (množina registrov) (4)

- **Accumulator** (ACC, akumulátor)
  - Jeden z operandov aritmetickej a logickej operácie je (skoro vždy) obsah akumulátora
  - Výsledok operácie sa ukladá do akumulátora
  - Takéto riešenie zjednodušuje výkonné aj riadiace obvody
  - Niektoré počítače používajú ako akumulátor jeden z univerzálnych registrov (pseudo-ACC)
- **Flag registers**
  - Obsahujú informácie potrebné na riadenie počítača
  - V SIC to boli E,F,C,S
  - Flag registers má každý počítač,ale každý svoje špecifické

4.3. Formáty inštrukcií (1)

- inštrukcia = binárny vektor, ktorý pre počítač umožňuje definovať
  - Operáciu, ktorú má vykonať
  - Operandy (hodnoty, s ktorými sa má operácia vykonať)
- Množina všetkých inštrukcií = instruction set (množina inštrukcií počítača) – určuje vnútorné usporiadanie počítača
- Inštrukcie podľa typu:
  - Aritmetické
  - Logické
  - Prenos údajov
  - I/O
  - Riadiace
  - iné

4.3. Formáty inštrukcií (2)

- inštrukcia=binárny vektor rozdelený na časti, ktoré sa nazývajú polia
- Formát inštrukcie = konvencia, ako interpretovať jednotlivé polia:
  - Operačný kód (čo sa má robiť)
  - Operandy (s čím sa to má robiť):
    - Explicitné (konštanty)
    - Registre
    - Pamäťové miesta
- Spôsob interpretácie operandu závisí od spôsobu adresovania
- V SIC mali všetky inštrukcie rovnakú dĺžku a uvažovali sa dva spôsoby adresácie (priama a indexová adresácia)
- V skutočných počítačoch existujú inštrukcie rozličných dĺžok a používa sa viacero spôsobov adresovania

4.3. Formáty inštrukcií (3)

- Adresové pole možno rozdeliť na viacero častí
- Prvé počítače mali 4 adresový formát:
  - Operand 1
  - Operand 2
  - Výsledok (adresa, na ktorú sa ukladá výsledok)
  - Adresa, na ktorej sa hľadala nasledujúca inštrukcia
- Súčasné počítače majú inštrukcie s menším počtom adries
- Od počtu adries v inštrukcii závisí veľkosť adresového priestoru
- Uvažujme 48 bitové adresové pole

Počet adries	Veľkosť adresového priestoru	
4	48:4=12	4 kB
3	48:3=16	64 kB
2	48:2=24	16 MB
1	48	256 TB

4.4. Spôsoby adresovania (1) (addressing modes)

- Určujú, ako treba interpretovať adresové pole inštrukcie; odkiaľ brať operandy
- operand:
  - Konštanta
  - Obsah registra
  - Obsah pamäťového miesta

4.4. Spôsoby adresovania (2) (addressing modes)

Mód	Hodnota operandu	Príklad prenosu
Implicitný	Žiadna	
Bezprostredný	konštanta	OPR:=číslo
Priamy	Pamäť na adrese	OPR:=M[ADR]
Nepriamy	Pamäť na adrese adresy	OPR:=M[M[ADR]]
Register	Obsah registra	OPR:=(R1)
Register (nepriamy)	Pamäť na adrese z registra	OPR:=M[(R1)]
Autoinkrement	Register, inkrement reg.	OPR:=(R1); ++R1
Relatívny	Pamäť na zloženej adrese	OPR:=M[(PC)+ADR]
Indexový	Pamäť na zloženej adrese	OPR:=M[(IX)+ADR]

4.4. Spôsoby adresovania (3)

Implicitný mód (implied mode)

- Operandy sú špecifikované priamo v operačnom kóde, napr. CLEAR_ACC
- Iný prípad: implicitným operandom aritmetickej operácie je obsah akumulátora
- Implicitnú adresáciu využíva zásobník: operandami sú najvyššie dva registre zásobníka, výsledok sa ukladá na vrch zásobníka
- existujú operácie na prenos údajov medzi pamäťou a zásobníkom

Bezprostredný spôsob adresovania (immediate addressing mode)

- V adresovom poli sú uložené konštanty (tento spôsob adresovania využíva napríklad operácia SHIFT)

Priame a nepriame adresovanie (direct and indirect addressing modes)

- Jeden bit slúži na odlíšenie, či ide o priamu alebo nepriamu adresáciu (napr. I=0 nepriame adresovanie, I=1 priame adresovanie)

4.4. Spôsoby adresovania (4)

Register v adresovom poli (adresové pole špecifikuje register)

- V priamom móde je register operandom
- V nepriamom móde je v danom registri adresa pamäťového miesta, ktorého hodnota je operandom
- autoinkrement/autodekrement mód sa dá používať aj v priamej aj v nepriamej adresácii
  - Takýto spôsob adresovania sa používa pri čítačoch, alebo indexových premenných, alebo pri zásobníkovej pamäti

Zložené adresovanie Adresa operandu sa vypočíta na základe údajov z adresového poľa a obsahov niektorých registrov

- *Relative adressing mode*: hodnota adresového poľa sa pripočítava k obsahu PC
- *Index adressing mode*: jeden register sa využíva ako indexový, adresa=obsah adresového poľa + (IX)
- Použitie - cykly

4.4. Spôsoby adresovania (5)

Zložené adresovanie-pokračovanie

- *Bázový register*: adresa operandu sa počíta ako súčet obsahu bázového registra (základná adresa) a obsahu adresového poľa (offset)
- *Augmented addressing:* namiesto sčítania obsahu registra a offsetu sa tieto hodnoty zreťazujú:

  adresa= (R)IR[AD]
- použitie: virtuálna pamäť; obsah registra= číslo stránky

Block addressing: využíva adresu na určenie pozície/adresy prvého slova v bloku údajov (pásky a disky)

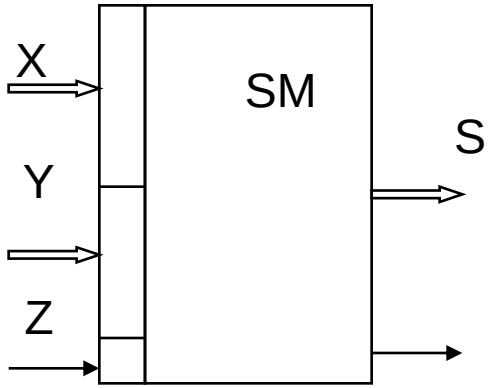
- Bloky údajov majú rovnakú alebo nerovnakú dĺžku
- Pri blokoch nerovnakej dĺžky potrebujeme určiť koniec bloku:
  - Adresa začiatku a konca bloku
  - Adresa začiatku a údaj o dĺžke bloku
  - EOB znak

4.5. Arithmetic and logic unit, ALU (Aritmetická a logická jednotka)

- Obvody vykonávajúce aritmetické a logické operácie
- V SIC
  - Register (akumulátor) ako implicitný argument aritmetickej a logickej operácie a miesto, kde sa ukladá výsledok
  - výhoda: jednoduchá štruktúra ALU
  - nevýhoda: pomalosť
- Reálne ALU môžu robiť operácie s ľubovoľným párom pracovných registrov a výsledok ukladať do ľubovoľného pracovného registra
- Spojenie registrov s ALU: multiplexory
- Aritmetické funkcie ALU závisia od typu údajov, reprezentácie čísel (pevná, pohyblivá rádová čiarka, záporné čísla)

4.5. ALU – aritmetické operácie

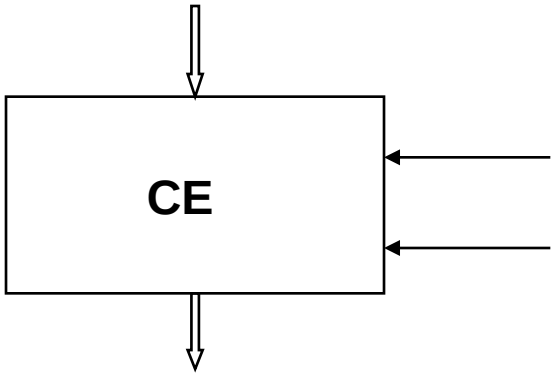
- ALU využíva binárny paralelný sumátor
- Predpokladá sa, že je podľa potreby možné negovať vstupy sumátora



X	Y	Z	Mikrooperácia
A	B	0	S:=A+B
A	B	1	S:=A+B+1
A	~B	0	S:=A+~B
A	~B	1	S:=A+~B+1
~A	B	0	S.=~A+B
~A	B	1	S.=~A+B+1
~A	~B	0	S.=~A+~B
~A	~B	1	S.=~A+~B+1
A	0	0	S:=A
A	0	1	S:=A+1
A	1	0	S:=A-1
A	1	1	S:=A

4.5. ALU – aritmetické operácie (2)

- Na vstup sumátora je dobré pripojiť *convert element,* ktorý transformuje vstup na požadovaný tvar (X,~X,1,0)



4.5. ALU – aritmetické operácie (3)

Násobenie a delenie

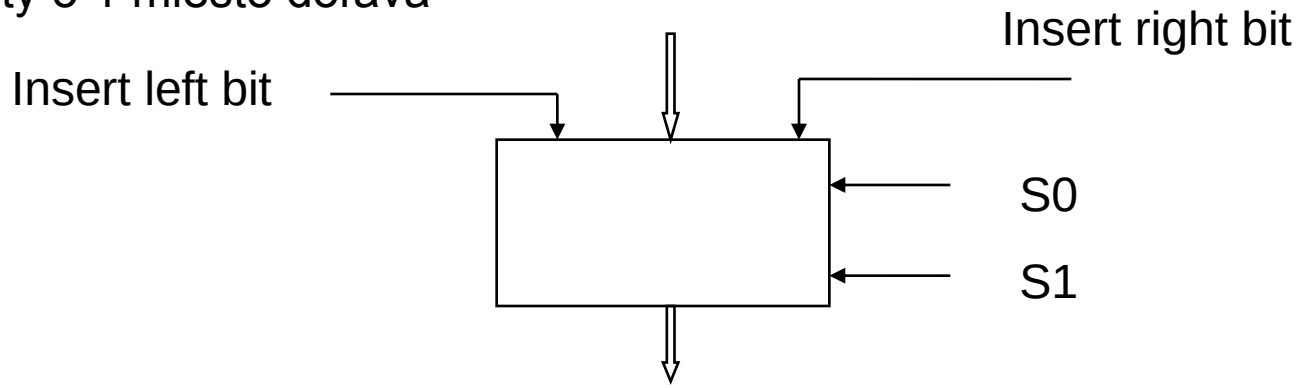
- Dá sa realizovať pomocou sčítania, odčítania, posunu a testovania
- Trvá dlho
- Čo sa s tým dá robiť
  - Čas vykonania mikroinštrukcie = čas vykonania časovo najnáročnejšej mikroinštrukcie
  - ALU s nerovnakými dĺžkami mikroinštrukcií
  - Nezaradiť násobenie a delenie medzi mikroinštrukcie
    - Realizácia násobenia a delenia pomocou mikroprogramu
    - Použitie aritmetického koprocesora

4.5. ALU – podmienkové bity

- Pri vykonávaní aritmetických operácií sa testuje výsledok a výsledky testovania sa ukladajú do podmienkových bitov (condition bits)
- Testuje sa (napr.)
  - Pretečenie (OF,overflow)
  - End carry (EC) prenos z posledného bitu
  - Sign (S) znamienko výsledku
  - Zero (Z) výsledok=0

4.5. ALU – logické funkcie

- Sú jednoduchšie ako aritmetické, lebo nie sú potrebné prenosy medzi rádmi
- Realizujú sa pomocou obvodov pozostávajúcich z rovnakých hradiel a pracujúcich paralelne
- AND, OR,NOR, XOR,NOT,...
- Na realizáciu posunu sa používa namiesto pomalého posuvného registra obvod *position scaler,* ktorý posunie vstup A na výstup B
  - Nezmenený
  - Posunutý o 1 miesto doprava
  - Posunutý o 1 miesto doľava



4.6. Control logic unit, CLU

- Generuje riadiace signály na vykonanie postupnosti mikroinštrukcií
- Riadi I/0 procesy, obslúženie prerušení
- CLU sa funkcionálne delí na
  - IP (instruction processor: fetch, address,interrupt cycles)
  - AP (arithmetic processor: execute cycles)

4.6. CLU – aritmetika v pevnej rádovej čiarke

- Operácie:
  - Aditívne (sčítanie, odčítanie),
  - multiplikatívne (násobenie a delenie)
- Čísla sú zapísané v tvare

znamienko|absolútna hodnota

- V prípade aditívnych operácií sa využíva binárny doplnkový kód
- V prípade multiplikatívnych operácií sa pracuje s absolútnymi hodnotami operandov a zvlášť sa vyhodnocuje znamienko
- Na realizáciu multiplikatívnych operácií sa používajú štandardné algoritmy

4.6. CLU – aritmetika v pohyblivej rádovej čiarke

- Exponent býva v excess kóde
- Mantisa má normalizovaný tvar
- Základom pre exponent bývajú mocniny čísla 2

4.6. CLU – aritmetika v pohyblivej rádovej čiarke

Aditívne operácie

1. Over, či sú operandy nenulové
2. Uprav menší z opernadov tak, aby oba operandy mali rovnaký exponent
3. Sčítaj/odčítaj mantisy
4. Normalizuj mantisu

Násobenie

1. Skontroluj nenulovosť operandov
2. Sčítaj exponenty
3. Vynásob mantisy
4. Normalizuj výsledok

Delenie

1. Skontroluj nenulovosť operandov
2. Uprav delenec (tak, aby po vydelení mantís bol výsledok v normálnom tvare, t.j. mantisa delenca musí byť ≥ mantisa deliteľa
3. Odčítaj exponenty
4. Vydeľ mantisy

4.6. CLU – aritmetika v desiatkovej sústave

- Tam, kde sa veľa počíta je výhodné previesť čísla z desiatkovej do dvojkovej sústavy, vykonať potrebné výpočty a výsledok zobraziť v desiatkovej sústave
- V jednoduchších zariadeniach (pokladne) je výhodnejšie používať BCD kódovanie a navrhnúť ALU pracujúcu s BCD číslami
- Ušetria sa prevody medzi desiatkovou a binárnou sústavou
- Či sa to robí naozaj – nevedno. Cena výkonných obvodov je taká nízka, že sa výkonné procesory môžu používať aj na elementárne účely.

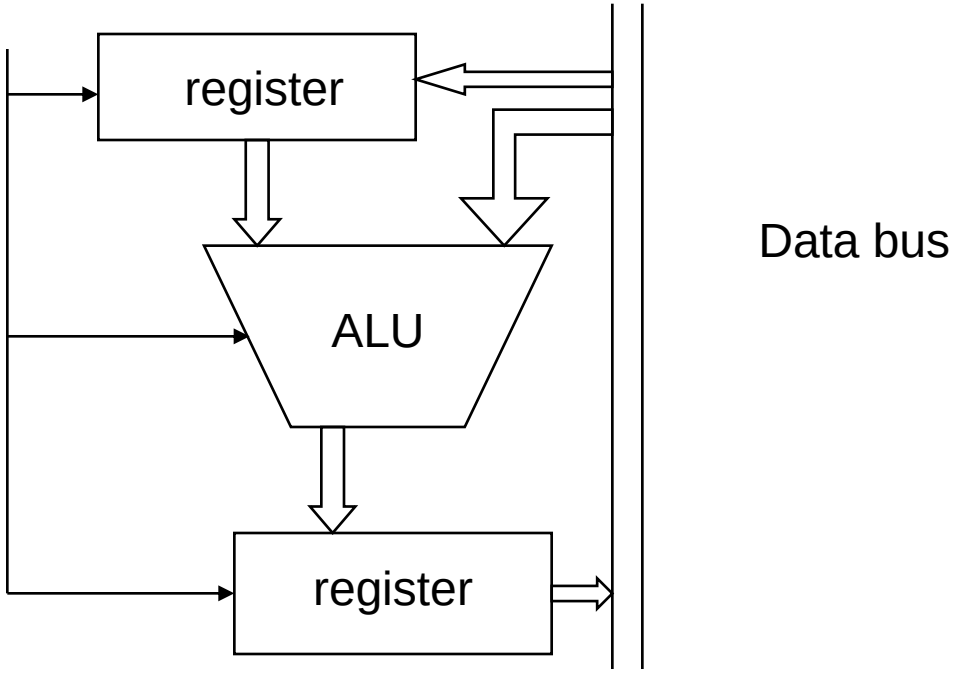
4.7. Konfigurácia CPU

- Časti CPU možno konfigurovať rozličným spôsobom
- Konfiguráciu CPU najviac ovplyvňuje počet vnútorných zberníc
- Uvedieme dva príklady

Jednozbernicová organizácia CPU

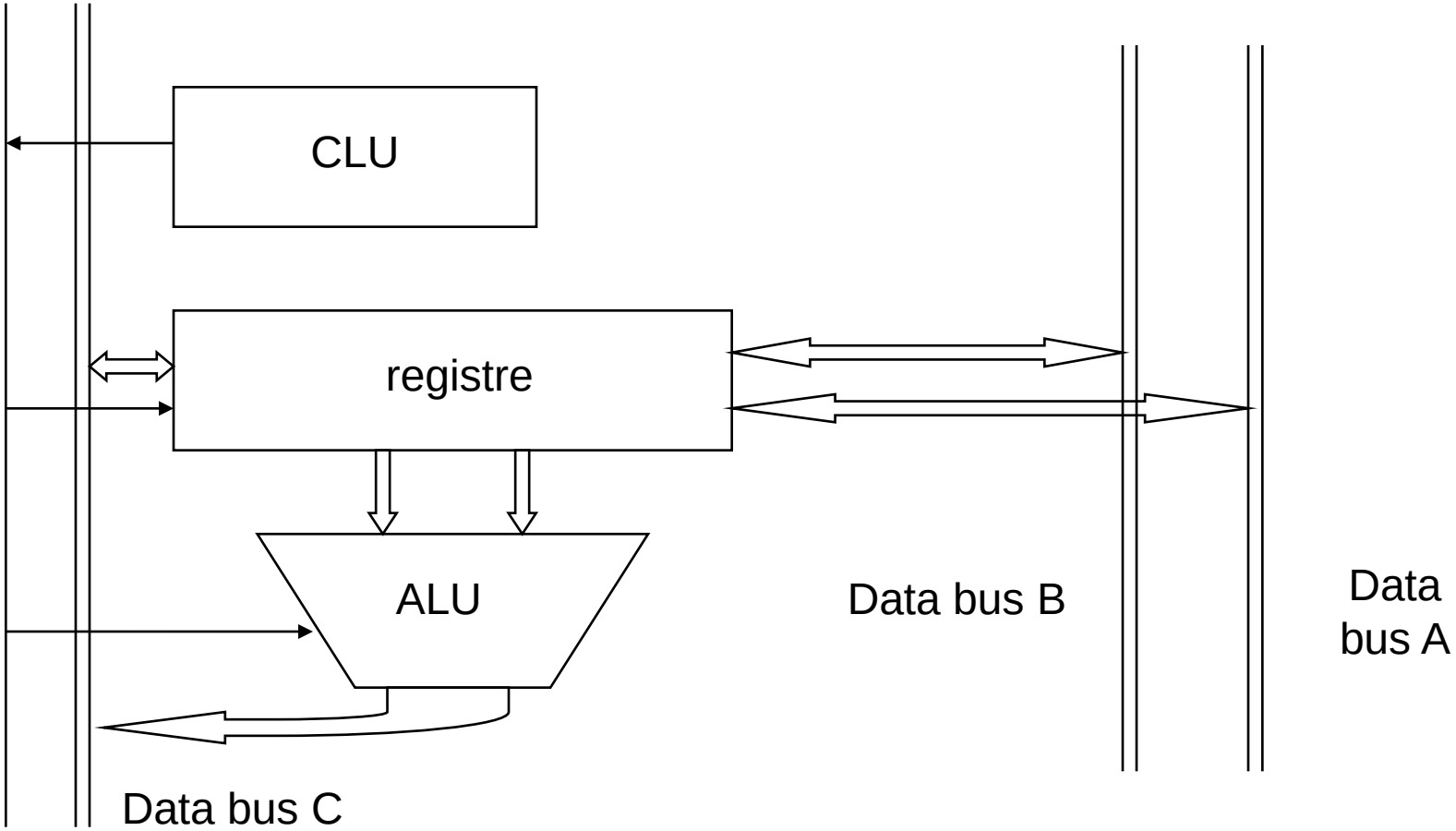
- CPU má 1 údajovú zbernicu
- Všetky údaje sa prenúšajú po tejto zbernici
- ALU potrebuje niekedy 2 operandy – jeden môže byť na zbernici, ale druhý musí byť v nejakom registri (buffer)
- CPU je kontrukčne jednoduchšia, ale spracovanie informácií trvá dlhšie, lebo údaje pre ALU treba najprv uložiť do registrov, potom vykonať operáciu a uložiť niekde výsledok

4.7. Konfigurácia CPU



4.7. Konfigurácia CPU – 3 dátové zbernice

Za cenu zložitejšej štruktúry sa dá zvýšiť výkon procesora



5. Mikroprogramovanie

- CLU riadi činnosť hardvéru počítača
  - Získanie inštrukcie (fetch)
  - Dekódovanie inštrukcie
  - Získanie operandov
  - Aktivizácia ALU
  - Uloženie výsledku
- Pre každú inštrukciu (makroinštrukciu) generuje CLU postupnosť riadiacich príkazov, pomocou ktorých sa daná makroinštrukcia vykoná
- Tieto príkazy sa nazývajú mikroinštrukcie a spravidla nevystupujú samostatne, ale tvoria mikroprogramy
- Čas potrebný na vykonanie (makro)inštrukcie sa nazýva *instruction cycle time*
- Makroinštrukcie majú rozličné dĺžky cyklov
- CLU rozdeľuje inštrukčné cyklu do *stavov*
- Stav zodpovedá trvaniu hodinového impulzu (taktu)
- Počas taktu/stavu možno vykonať jednu alebo niekoľko nezávislých mikroinštrukcií súčasne

5. Mikroprogramovanie - mikroinštrukcie

- Mikroinštrukcia = príkaz najnižšej úrovne
- Vykonávajú ju logické obvody
- Príklady mikroinštrukcií:
  - Otvoriť/uzavrieť prístup údajov z registra na zbernicu
  - Prenesenie údajov po zbernici
  - Inicializácia riadiacich signálov READ, WRITE, SET, CLEAR, SHIFT
  - Odoslanie signálu
  - Čakanie predpísanú dobu
  - Testovanie bitu v registri
  - Zápis do registra
- CLU môže mikroinštrukcie
  - Generovať (hard-wired logic)
  - Získavať z mikroprogramovej pamäte

5.1. Hard-wired CLU

- CLU=riadiaca jednotka realizovaná logickými obvodmi, ktorá generuje postupnosť signálov, riadiacich fetch, address, decode, execute and interrupt cycles
- CLU môže byť asynchrónna (ukončenie predchádzajúcej mikroopoerácie spúšťa vykonávanie ďalšej mikrooperácie)
- Synchronizovaná CLU: každá operácia je riadená časovým signálom
- Príklad ADD x

  t0: MAR:=(PC)

  t1: MBR:=M[MAR], PC:=(PC)+1

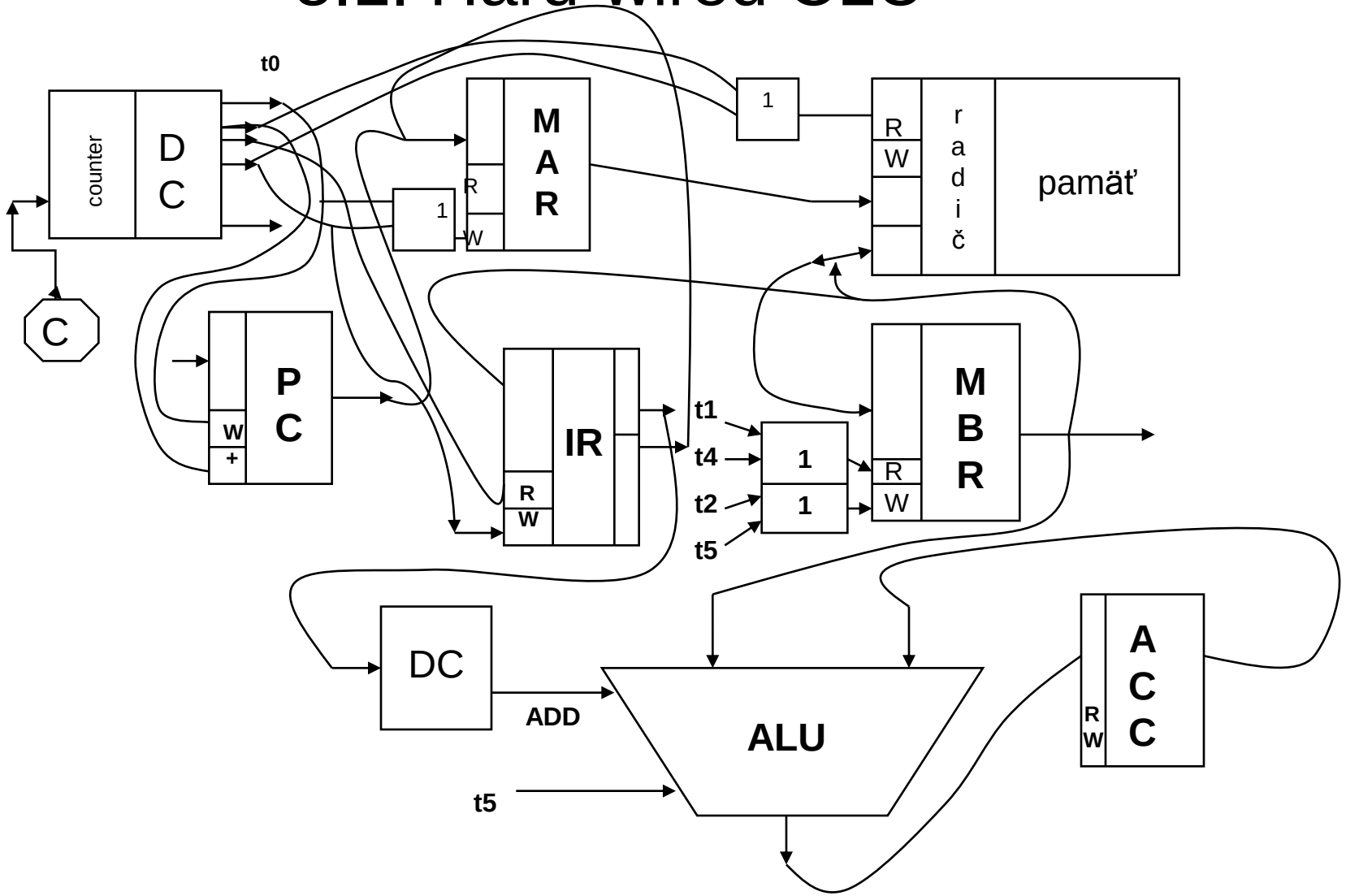
  t2: IR:=(MBR)

  t3: MAR:=(IR[ADDR])

  t4: MBR:=M[MAR]

  t5: ACC:=(ACC)+(MBR)

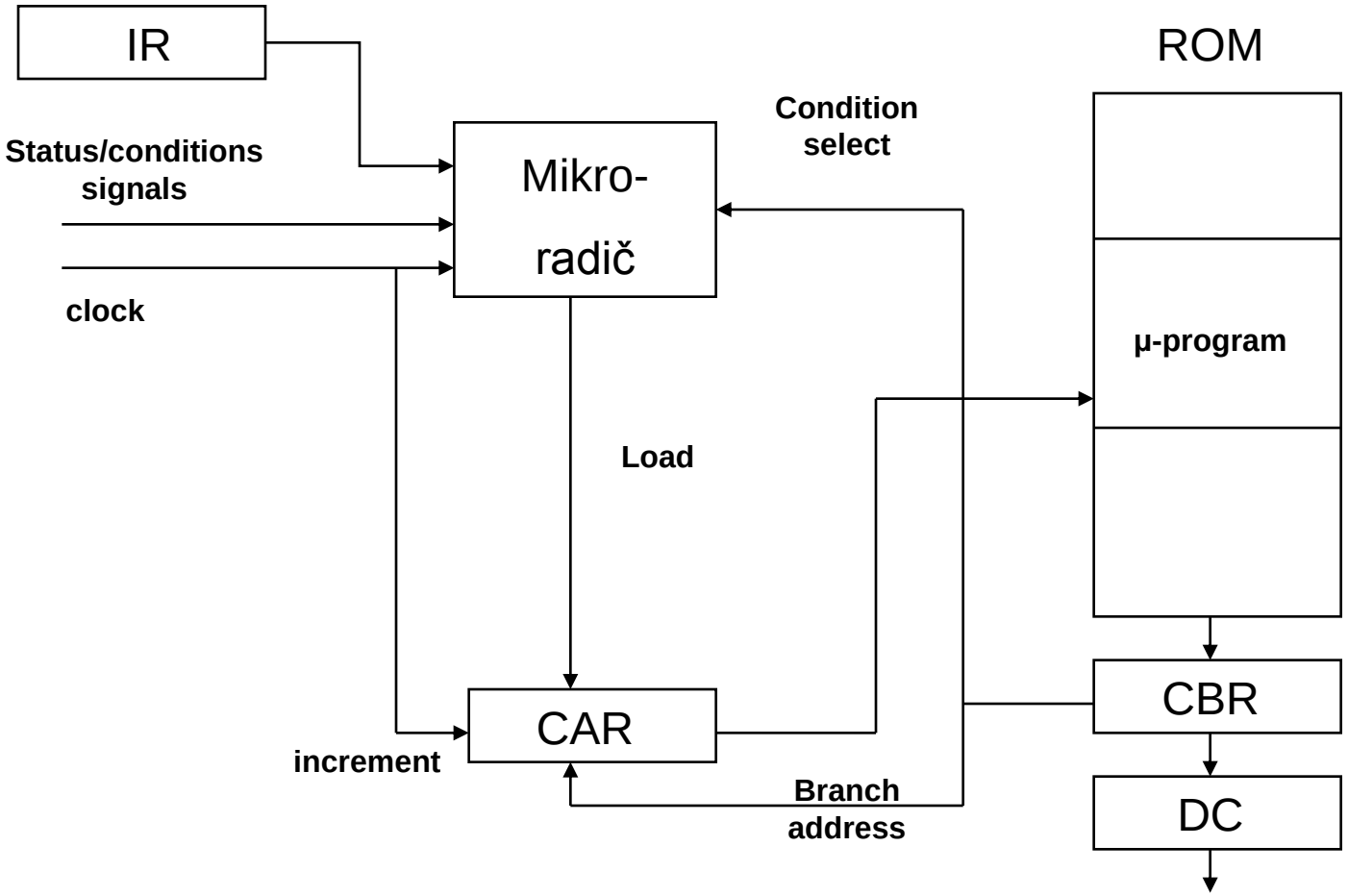
5.1. Hard-wired CLU



5.2. Mikroprogramové riadenie

- Základná idea M.V.Wilkes, začiatok 50. rokov
- Nevýhody hard-wired CLU:
  - Modifikácia, zavedenie novej mikroinštrukcie = nový návrh
- Ako to chceli riešiť: mikroprogram na vykonanie inštrukcie uložený v pamäti a interpréter schopný vykonať ho
- Na čom to skroskotávalo: neboli k dispozícii lacné a rýchle pamäte na uchovávanie mikroprogramov
- realizácia: 1964, IBM 360
- Mikroprogramové CLU sa dajú rozdeliť v závislosti na možnosti používateľa zasahovať do mikroprogramov:
  - Nemenné
  - Čiastočné zmeny
  - Programovateľné
- Mikroprogramová CLU je mikroprogramovateľná, ak používateľ môže naprogramovať vlastné makroinštrukcie

5.3. Organizácia mikroprogramovej CLU (1)



5.3. Organizácia mikroprogramovej CLU (2)

IR register inštrukcií CPU
CAR control address register
CBR control buffer register
DC decoder

Ako funguje mikroprogramová CLU?
(„sa“ = mikroprogramová CLU)

- V IR je uložená aktuálna makroinštrukcia

1. Do CAR sa uloží adresa mikroinštrukcie (na začiatku prvej)
2. Tá sa prečíta z mikroprogramovej pamäte do CBR
3. Začína sa mikrocyklus, počas ktorého sa generujú riadiace signály na vykonanie mikroinštrukcie
4. CAR sa spravidla zvyšuje o 1, ale môžu sa vyskytnúť skoky
5. Cyklus 1-5 sa opakuje dovtedy, kým sa mikroprogram neskončí

5.4. Formáty mikroinštrukcií (1)

1. Horizontálny
2. Vertikálny

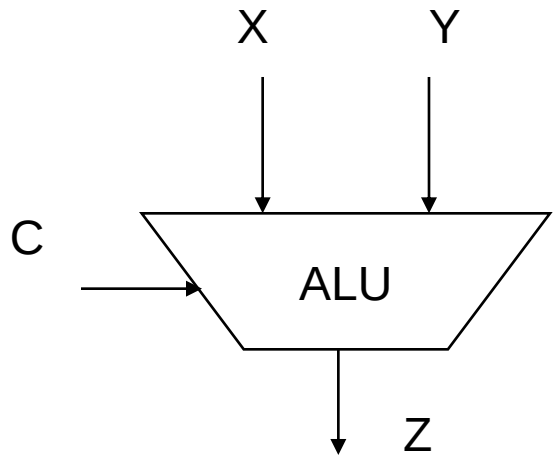
Horizonálny formát: binárny vektor, ktorý obsahuje toľko bitov, koľko môže mikroinštrukcia generovať riadiacich signálov

- Dá sa vykonať viacero elementárnych činností naraz (+)
- Dlhý a riedky vektor, lebo veľa mikroinštrukcií sa vzájomne vylučuje; nároky na pamäť (-)

Vertikálny formát

- Mikroinštrukcia špecifikuje len jednu mikrooperáciu (?)
- Vyžaduje si dekóder, zložitejší hardvér, nevyužíva paralelizmus (-)
- Krátke mikroinštrukcie, šetrí pamäť (+)

5.4. Formáty mikroinštrukcií (2)



- X,Y – 8 bitové vstupy
- Z – 8 bitový výstup
- C – 2 riadiace bity
- Vstupy a výstupy idú do/z registrov R0...R16, ktoré sú adresovateľné pomocou 4 bitov

5.4. Formáty mikroinštrukcií (3)

- ALU vykonáva 3 operácie
  - NOP 00
  - X+Y 01
  - X-Y 10
- Postupnosť operácií spojených s ALU:
  - X:=(register)
  - Y:=(register)
  - Z:=f(X,Y)
  - Register:= Z
- Pri horizontálnej mikroinštrukcii budeme potrebovať 14 bitov
- [op.kód][X-pole][Y-pole][Z-pole]

10 0100 0011 0000 : R0:=(R4)-(R3)

5.4. Formáty mikroinštrukcií (4)

Vertikálny formát

rozlišujeme:

- Výber X 00
- Výber Y 01
- Výber ALU 10
- Výber Z 11

A výber adresy (registra) 4 bity

V prípade výberu ALU musíme vybrať operáciu, na to potrebujeme 2 bity

Mikroprogram bude vyzerať takto:

00 0100
01 0011
10 10xx
11 0000

V tomto prípade potrebujeme 24 bitov, oproti 14 bitom pri použití jednej horizontálnej inštrukcie

5.4. Formáty mikroinštrukcií (5)

Postupnosť mikroinštrukcií:

- potrebujeme umožniť podmienené skoky
- Aby sa to zjednodušilo:
  - Mikroinštrukcia bude obsahovať 2 bitové COND field C1C2
  - 2 možné pokračovania:
    - na adrese danou v ADDR field
    - (CAR)+1
  - Hodnota COND field určí, ktorá možnosť nastane:
  - 00 (CAR)+1
  - 01 skoč na ADDR ak C1
  - 10 skoč na ADDR ak C2
  - 11 skoč na ADDR nepodmienene

5.5. Mikroprogramovanie - rozličné

1. Emulácia: ak je možné nahradiť mikroprogram iným, možno na počítači emulovať iný počítač s iným inštrukčným súborom (napr. neexistujúci počítač)
2. Bitové rezy = bloky pozostávajúce z obvodov a zberníc, umožňujúce spracovať 2-8 bitov. Možno z nich poskladať procesory spracovávajúce slová ľubovoľnej dĺžky. Prenosy medzi rezmi registrov (pri aritmetických operáciách) treba riešiť mikroprogramovo.
3. Mikroprogramovacie podporné prostriedky.

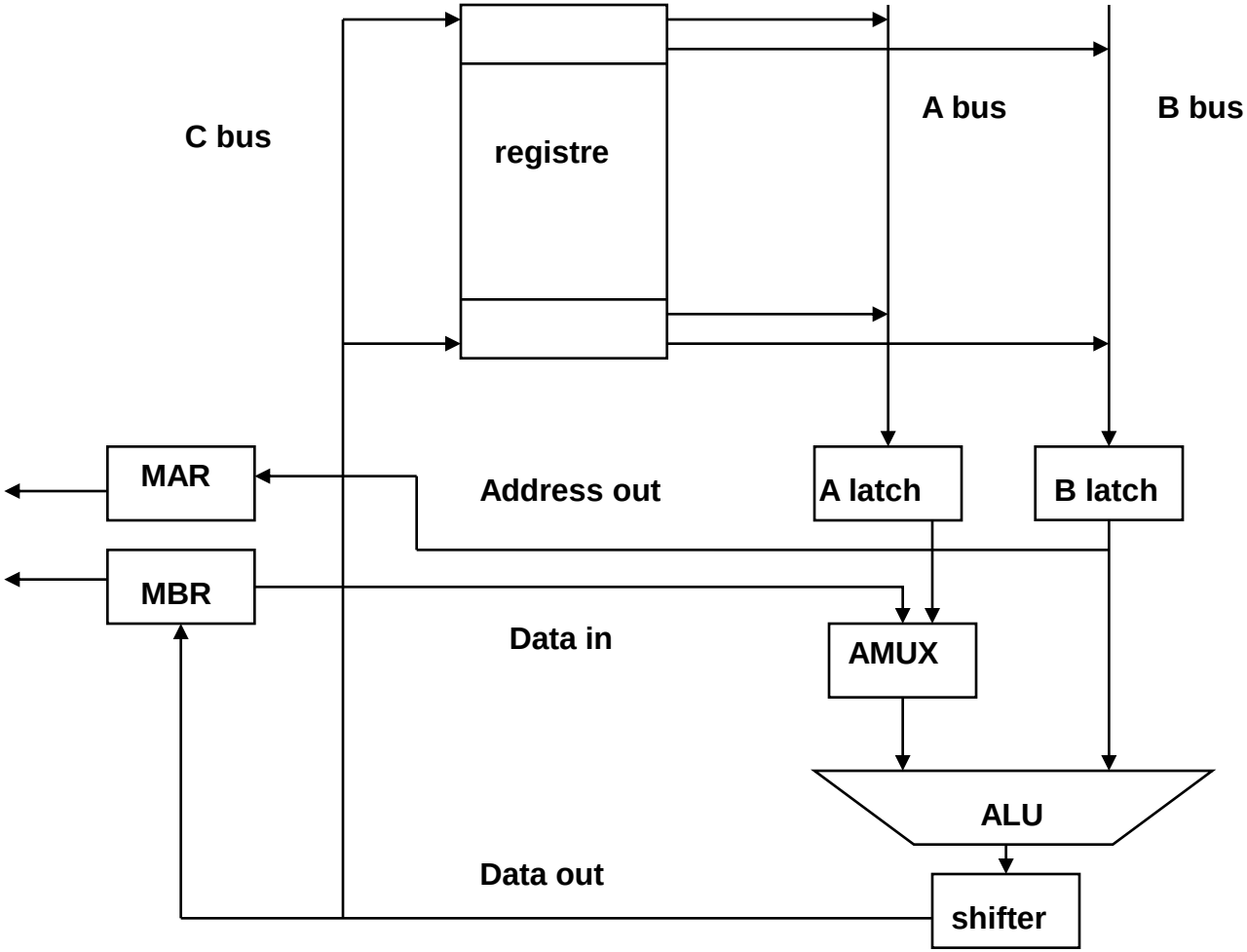
- Mikroprogramovanie je otravná záležitosť, treba si nejako pomôcť
- Mikroassembler
- Formátor (programovanie PROM)
- Vývojové systémy (testovanie, editovanie)
- Hardvérové simulátory

5.6. Výhody a nevýhody mikroprogramovania

- Štruktúrovaný prístup k návrhu CLU (+)
- CLU sa ľahšie mení a opravuje (+)
- CLU je spoľahlivejšia ako random logic CLU (+)
- CLU je pomalšia (-)

- Zlepšenie využívania mikroprogramovej pamäte – nanoprogramovanie
- Princíp:
  - Spraví sa zoznam v mikroprograme používaným mikroinštrukcií a tieto sa uložia v nanopamäti
  - Mikroprogram pozostáva z kódov mikroinštrukcií: pri spracovaní mikroprogramu sa mikroinštrukcia najprv „vytiahne“ z nanopamäte a potom spracuje

5.7. Príklad mikroarchitektúry (1)



5.7. Príklad mikroarchitektúry (2)

- Podľa A. S. Tanenbauma, Structured Computer organization
- Data path na prechádzajúcom obrázku obsahuje:
  - ALU a jej vstupy a výstupy
  - 16 registrov dĺžky 16 bitov
  - Každý register je pripojený na tri zbernice A, B, C
  - Do registra sa ukladajú údaje pomocou zbernice C
  - Obsah registra sa dá preniesť (napr. do ALU) prostredníctvom zberníc A, B
  - Zbernice A, B sú pripojené na ALU
  - ALU je 16-bitová ALU, ktorá je schopná vykonávať 4 operácie: A+B, A AND B, A, NOT A
  - Výber operácie ALU sa uskutočňuje pomocou dvoch riadiacich vstupov F0F1 (nie sú na obrázku)
  - ALU generuje dva stavové bity, ktoré závisia od výsledku operácie: N (výsledok operácie je negatívny); Z (výsledok operácie je 0)
  - Výstup ALU prechádza cez SHIFTER, tento môže posunúť výsledok o 1 miesto doprava alebo doľava, alebo ho ponechať bez zmeny.
  - SHIFTER riadia dva riadiace vstupy S0S1 (nie sú na obrázku)

5.7. Príklad mikroarchitektúry (3)

- Vstupy do ALU prechádzajú cez dva registre (latch A, B), ktoré slúžia na zabezpečenie stabilného vstupu ALU na potrebnú dobu
- Komunikáciu s pamäťou zabezpečuje dvojica registrov MAR, MBR
- Riadenie vstupov a výstupov MAR, MBR zabezpečujú riadiace vstupy M0,M1,M2,M3 (nie sú na obrázku):
  - M0 ukladanie adresy do MAR
  - M1 ukladanie hodnoty do MBR z výstupu SHIFTER-a
  - M2 čítanie z pamäte
  - M3 zápis do pamäte
- AMUX je multiplexor, ktorý vyberá vstup do ALU (buď z MBR, alebo z buffra A) pomocou riadiacej premennej A0
- registre: PC, AC, SP, IR, TIR, 0, +1, -1, AMASK, SMASK, A,B,C,D,E,F

5.7. Príklad mikroarchitektúry (4)

- Na riadenie data path potrebujeme 61 bitov
- 16 bitov na zápis z registrov na zbernicu A
- 16 bitov na zápis z registrov na zbernicu B
- 16 bitov na zápis do registrov zo zbernice C
- 2 bity na riadenie zápisu do buffrov A, B
- 2 bity na riadenie funkcií ALU
- 2 bity na riadenie SHIFTER-a
- 4 bity na riadenie MAR a MBR
- 2 bity na riadenie zápisu a čítania z/do pamäte
- 1 bit na riadenie AMUX

Optimalizácia

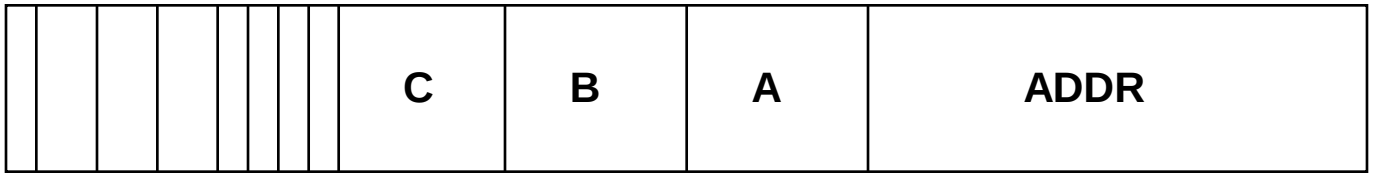
- Čísla registrov pristupujúcich na zbernice A,B,C = 3x4=12 bitov
- L0, L1 (buffre A,B) sa dajú nahradiť časovým signálom
- Nový riadiaci bit ENC (enable C) na zápis hodnoty z C do niektorého z registrov
- Na riadenie MBR stačia dva bity RD a WR

5.7. Príklad mikroarchitektúry (5)

- AMUX riadi ľavý vstup do ALU
  - 0 = A latch
  - 1 = MBR
- COND – podmienka pre skok
  - 0 = žiadny skok
  - 1 = skok ak N = 1
  - 2 = skok ak Z = 1
  - 3 = skok
- ALU – riadenie funkcií ALU
  - 0 = A+B
  - 1 = A AND B
  - 2 = A
  - 3 = ~A
- SH – riadenie posunu
  - 0 = žiaden posun
  - 1 = posun o 1 bit doprava
  - 2 = posun o 1 bit doľava
  - 3 = nepoužíva sa

5.7. Príklad mikroarchitektúry (6)

- MAR,MBR,RD,WR,ENC: 0 = nie, 1 = áno;
- MAR = ulož do MAR obsah latch B
- MBR = ulož do MBR výstup shiftera
- RD = čítaj z pamäte
- WR = zapisuj do pamäte
- ENC = ulož do registra
- C = výber registra na uloženie, ak ENC = 1;
- B = výber zdroja hodnôt pre zbernicu B
- A = výber zdroja hodnôt pre zbernicu A



5.7. Príklad mikroarchitektúry (7) Timing

- Základný cyklus ALU:
  - Nastavenie registrov (latches) A, B
  - Aktivizácia ALU a shiftera na potrebný čas
  - Uloženie výsledkov
- Je potrebné zaistiť správne poradie vykonávaných činností
- Základný cyklus sa rozdelí na 4 podcykly:
  - Uloženie nasledujúcej mikroinštrukcie do MIR (registra mikroinštrukcií)
  - Pripojenie registrov na zbernice A, B a naplnenie buffrov (latches) A, B
  - Keď sú vstupy do ALU stabilné, ponechanie ALU a shifteru dostatok času na vykonanie potrebných operácií
  - Keď je výstup shiftera stabilný, uloží sa hodnota zo zbernice C buď do registra, alebo do MBR

5.7. Príklad mikroarchitektúry (8)

- Výber nasledujúcej mikroinštrukcie
  - Nebýva to vždy nasledujúca mikroinštrukcia
  - Aby sa umožnil skok, mikroinštrukcia obsahuje dve polia: ADDR a COND
  - Ak COND = 0, nasledujúca mikroinštrukcia sa berie z adresy MPC+1
  - Ak COND > 0 a sú splnené ďalšie podmienky, pokračuje sa na adrese uvedenej v poli ADDR

5.8. Mikroprogramovanie – Súhrn (1)

- CPU pozostáva z
  - data path (registre, ALU, SHIFTER, zbernice)
  - riadiacej časti (mikroprogramová pamäť)
- Cyklus pozostáva zo získania operandov z registrov, spracovaniu pomocou ALU/shiftera a uloženia výsledku do registra
- Riadiaca sekcia – mikroprogramová pamäť s uloženým mikroprogramom
- Mikroinštrukcia kontroluje obvody data path počas jedného mikrocyklu
- Mikrocyklus môže byť rozdelený na podcykly
- Postupnosť mikroinštrukcií sa dá zaistiť jednak pomocou hodín a explicitného čítača (program counter) na mikroprogramovej úrovni
- V reálnych procesoroch mikroinštrukcia obsahuje adresu nasledujúcej mikroinštrukcie

5.8. Mikroprogramovanie – Súhrn (2)

- Mikroinštrukcie sa dajú organizovať horizontálne a vertikálne alebo niečo medzi tým
  - Horizontálne mikroinštrukcie: dlhé slová, paralelené vykonávanie mikrooperácií
  - Vertikálne inštrukcie: krátke slová, pomalšie, menšia mikroprogramová pamäť
- Optimalizácia: mikroinštrukcia = krátky pointer na dlhšiu nanoinštrukciu
  - Menšie nároky na mikroprogramovú pamäť, pomalšie vykonávanie programu
- Iné spôsoby optimalizácie: variabilná dĺžka cyklov, pipelining, použitie cache a pod.
- príklad: Motorola 68000
  - 3 nezávislé data path (2 – adresa, 1 – údaje)
  - 17 bitové nanoinštrukcie a 68 bitové horizontálne nanoinštrukcie

6. RISC a CISC (1/2)

- Prvé počítače – jednoduché, málo inštrukcií, a 1-2 spôsoby adresovania
- 1964 – IBM 360 mikroprogramovanie
  - Zložitý inštrukčný súbor (strojový jazyk)
  - Mikroprogramy uložené v ROM (nemodifikovateľné)
- Ďalší vývoj
  - typický počítač má cca 200 inštrukcií a viac než 10 spôsobov adresovania
  - Používanie vyšších programovacích jazykov so štruktúrami typu IF, WHILE, CASE a jazykov assemblera (JUMP, MOVE, ADD,...) vedie k vzniku a rozširovaniu sémantickej priepasti a problémom s písaním kompilátora
  - Neprichádza do úvahy znižovanie úrovne programovacích jazykov
  - Zvyšuje sa úroveň strojového jazyka (inštrukcie pre CASE, adresovacie spôsoby pre narábanie s poľami a zoznamami, volanie procedúr) – do mikrokódu
  - Pomalá hlavná pamäť a rýchla CPU

6. RISC a CISC (2/2)

- 70-te roky: technologické zmeny
  - Rýchle polovodičové RAM
- Zložitá práca s mikroprogramami
  - písanie, ladenie, udržiavanie, zmeny
- Chyba v mikroprograme
  - nutnosť vymeniť ROM s mikroprogramom v používateľskom počítači

6. RISC a CISC analýza programov (1/3)

- Čo je vlastne potrebné podporovať?
- Knuth, Wortman, Tanenbaum, Patterson skúmali Fortran, PL/I, C, Pascal, SAL (1971-1982)
- výsledky:
  - 85% programov: priradenia, IF, volania procedúr
  - 80% priradení: premenná:=hodnota (konštanta, premenná, prvok poľa)
  - 15% priradení obsahuje 1 operátor, napríklad a:=a+b
  - 5% priradení – zložitejších

6. RISC a CISC analýza programov (2/3)

Volanie procedúr

Lokálne premenné		parametre	
0	22%	0	41%
1	17%	1	19%
2	20%	2	15%
>2	41%	>2	25%

6. RISC a CISC analýza programov (3/3)

- Celková štatistika:

:=	47%
IF	23%
Call	15%
Loop	6%
Goto	3%
Ostatné	7%

- Počítače (mikroprogramy) podporujú zložité inštrukcie programy sa však zväčša skladajú z jednoduchých inštrukcií

6. RISC

- Možné riešenie problému: RISC
  - Počítač s malým počtom vertikálnych mikroinštrukcií
  - Používateľské programy sa kompilujú do postupnosti týchto mikroinštrukcií a potom vykonávajú hardvérovo (nepoužíva sa interpréter)
  - Výhoda: jednoduché operácie (sčítanie obsahu dvoch registrov) sa dajú vykonať pomocou jednej mikroinštrukcie
  - Pre porovnanie – CISC – najrýchlejšie strojové kódy vyžadujú až 8-15 mikroinštrukcií na makroinštrukciu
  - Ďalší predpoklad úspechu RISC = pokrok v mikroprogramovaní, optimalizácia technológie tvorby kompilátorov – generovanie mikrokódu
  - Predtým: programátor napísal ručne mikroprogram, ktorý interpretoval používateľov program
  - Teraz kompilátor vyprodukuje priamo mikrokód (preskakuje sa interpréter)

6. RISC

- Do roku 1964 boli všetky počítače typu RISC
- Potom prišiel Wilkes a IBM 360 s mikroprogramovaním
- Prvý moderný RISC – 1975 IBM 801 minicomputer

	IBM 370	VAX 11/780	Xerox Dorado	IBM 801	RISC I	MIPS
rok výroby	1973	1978	1978	1980	1981	1983
Počet inštr.	208	303	270	120	3	55
Mikrokód	54 Kb	61 Kb	17 Kb	0	0	0
Veľ. inštr.	2-6 b	2-57 b	1-3	4	4	4
Operácie	R-R R-M M-M	R-R R-M M-M	Stack	R-R	R-R	R-R

6. Princípy návrhu RISC

- 5 krokov (Tanenbaum)
  1. Analyzuj aplikáciu aby si našiel kľúčové operácie
  2. Navrhni data path optimálnu pre kľúčové operácie
  3. Navrhni inštrukcie na vykonanie kľúčových operácií pomocou navrhnutej data path
  4. Nové inštrukcie pridávaj len vtedy, ak nespomalia stroj
  5. Opakuj tento postup pre ďalšie časti CPU (cache, manažment pamäte, koprocesory a pod.)

  Dokonalosť sa dosiahne nie vtedy, keď už nieto čo pridať, ale vtedy, keď už nieto čo odobrať.

  Antoine de St. Exupéry

6. RISC a CISC – porovnanie (1/9)

RISC	CISC
1. Jednoduché inštrukcie vykonávané v jednom cykle	1. Zložité inštrukcie vykonávané vo viacerých cykloch
2. Prístup k pamäti len pomocou LOAD a STORE	2. Takmer každá inštrukcia môže pristupovať k pamäti
3. Intenzívne využíva pipelining	3. Pipelining používa obmedzene
4. Inštrukcie vykonáva hardvér	4. Inštrukcie interpretuje mikroprogram
5. Pevný formát inštrukcií	5. Variabilný formát inštrukcií
6. Málo inštrukcií a spôsobov adresovania	6. Veľa inštrukcií a spôsobov adresovania
7. Zložitosť v kompilátore	7. Zložitosť v mikroprograme
8. Viacero množín registrov	8. Jedna množina registrov

6. RISC a CISC – porovnanie (2/9)

Jedna inštrukcia na 1 cyklus data path

- Data path cyklus pozostáva zo
  - Získania operandov z registrov
  - Posun operandov na vnútorné zbernice
  - Spracovanie v ALU
  - Zápis výsledku do registra
  - Toto všetko sa dá stihnúť za 1 takt
- Inštrukcie RISC-u sa podobajú mikroinštrukciám
- Tie, ktoré sú príliš komplikované, sa do inštrukčného súboru nezaraďujú (násobenie, delenie, operácie s pohyblivou rádovou čiarkou; riešenie: knižnice, koprocesor)

6. RISC a CISC – porovnanie (3/9)

Architektúra LOAD/STORE

- Ak je cieľom jedna inštrukcia / cyklus Data path – inštrukcie pristupujúce k pamäti sú problematické
- Inštrukcie, ktoré majú operandy v registroch sa dajú vykonať počas jedného cyklu
- Inštrukcie, ktoré majú operandy v pamäti – nie
- Predĺženie dĺžky cyklu tak, aby sa to stíhalo – nie je prípustné
- Riešenie: inštrukcie majú operandy v registroch
- Existujú inštrukcie STORE, LOAD na čítanie z pamäte a ukladanie do pamäte
- Jednoduché spôsoby adresovania (žiadne nepriame adresovanie, indexové adresovanie a pod.)

6. RISC a CISC – porovnanie (4/9)

Pipelining

- LOAD/STORE aj tak trvajú dlhšie ako 1 data path cyklus
- Modifikujeme základný cieľ (1 inštrukcia / cyklus data path) na v priemere 1 inštrukcia / cyklus data path
- Podstata pipeleningu:
  - spracovanie inštrukcie sa dá rozdeliť na fázy,
  - v každej fáze sa na spracovaní inštrukcie podieľajú iné obvody,
  - môžeme mať súčasne rozpracovaných viac inštrukcií;
    - každú v inej fáze spracovania
  - v priemere sa spracuje v každom data path cykle jedna inštrukcia
- Je potrebné analyzovať, aké obvody sa využívajú v jednotlivých fázach spracovania inštrukcie, aby sa vyhlo kolíziám
- Posúva sa začiatok spracovania nasledujúcej problematickej inštrukcie
- V záujme optimalizácie programu: kompilátor má usporiadať preložený program tak, aby využil „medzery“ okolo LOAD a STORE na niečo užitočné
- Ďalší problém sú operácie skoku
  - CISC: predikcia, RISC na to nemá čas – oneskorený JUMP (inštrukcia nasledujúca po skoku sa vždy začne vykonávať)

6. RISC a CISC – porovnanie (5/9)

Žiaden mikrokód

- Inštrukcie generované kompilátorom v RISC-u vykonáva hardvér
- Nie sú interpretované mikroprogramom
- To je podstata rýchlosti RISC-u

Čo s komplikovanými inštrukciami?

- Nevyskytujú sa príliš často
- CISC – používa ich a realizuje ich pomocou mikroprogramu
- V RISC-u sa musia nahradiť programom (kompilátor)
- CISC možno trochu šetrí pamäť

6. RISC a CISC – porovnanie (6/9)

Pevný formát inštrukcií

- Kde by sa prejavili problémy, keby sme pripustili variabilný formát inštrukcií?
  - Načítavanie
  - Pipelining
  - Spracovanie

6. RISC a CISC – porovnanie (7/9)

Redukovaný súbor inštrukcií

- RISC principiálne nemusí mať malý počet inštrukcií
- Podmienky, ktoré sme na inštrukcie položili, mnohé inštrukcie vylučujú (napr. adresovanie)
- Malý inštrukčný súbor – menšie požiadavky na plochu čipu – ušetrená plocha sa dá využiť (registre)
- Málo a jednoduché spôsoby adresovania

6. RISC a CISC – porovnanie (8/9)

Zložitosť RISC v kompilátore

- Prečo?
  - Oneskorené LOAD, STORE, JUMP operácie
  - Obyčajné inštrukcie nemôžu adresovať pamäť
  - Cieľová množina inštrukcií (strojový kód), do ktorej sa prekladá program, je obmedzená
- CISC – zložitosť v mikroprograme

6. RISC a CISC – porovnanie (9/9)

Viac množín registrov

- Ušetrený priestor na čipe využívajú registre
- Ich použitím sa budeme špeciálne zaoberať

Otvorené otázky

- Aká časť hardvéru bude viditeľná pre tvorcu kompilátora
  - interlocking po LOAD – riešiť automaticky, alebo to nechať na programátora
  - CACHE – môže programátor predpokladať, že má potrebnú hodnotu v CACHE
- Big vs. little endian
- Použitie CC (condition code) – v každej inštrukcii alebo len špeciálne inštrukcie skoku

6. RISC použitie registrov (1/6)

- RISC má viac súborov registrov
- Cieľ – jedna inštrukcia (v priemere) na 1 cyklus data path
- LOAD, STORE vyžadujú v priemere 2 cykly
- Kompilátor musí za každou LOAD, STORE (a JUMP) operáciou umiestniť bezproblémovú operáciu (ktorá sa dá začať vykonávať spolu s LOAD a STORE)
- Treba minimalizovať počet LOAD, STORE
- Čo sa najčastejšie číta a ukladá?
- Volanie procedúr: parametre, obsahy registrov, návratové adresy
- Patterson a Sequin (1982) tvorcovia RISC I vymysleli metódu prekrývajúcich sa okien registrov (overlapping register windows)
- Podstata metódy:
  - CPU má veľa registrov
  - Viditeľných je len niekoľko z nich (najčastejšie 32)
  - Viditeľné registre sú rozdelené na 4 skupiny

6. RISC použitie registrov (2/6)

- Rozdelenie registrov okna (podľa RISC I):
  - Globálne premenné (10)
  - Vstupné parametre (6)
  - Lokálne premenné (10)
  - Výstupné parametre (6)
- Súčasné RISCs majú rozdelenie 8,8,8,8
- Použitie registrov okna:
  - Globálne premenné a pointre:
    - nie sú špecifické pre konkrétnu procedúru
    - O využití registra rozhoduje kompilátor
    - Niekde je R0 hardvérovo nastavené na 0 a nedá sa doň zapisovať

6. RISC použitie registrov (3/6)

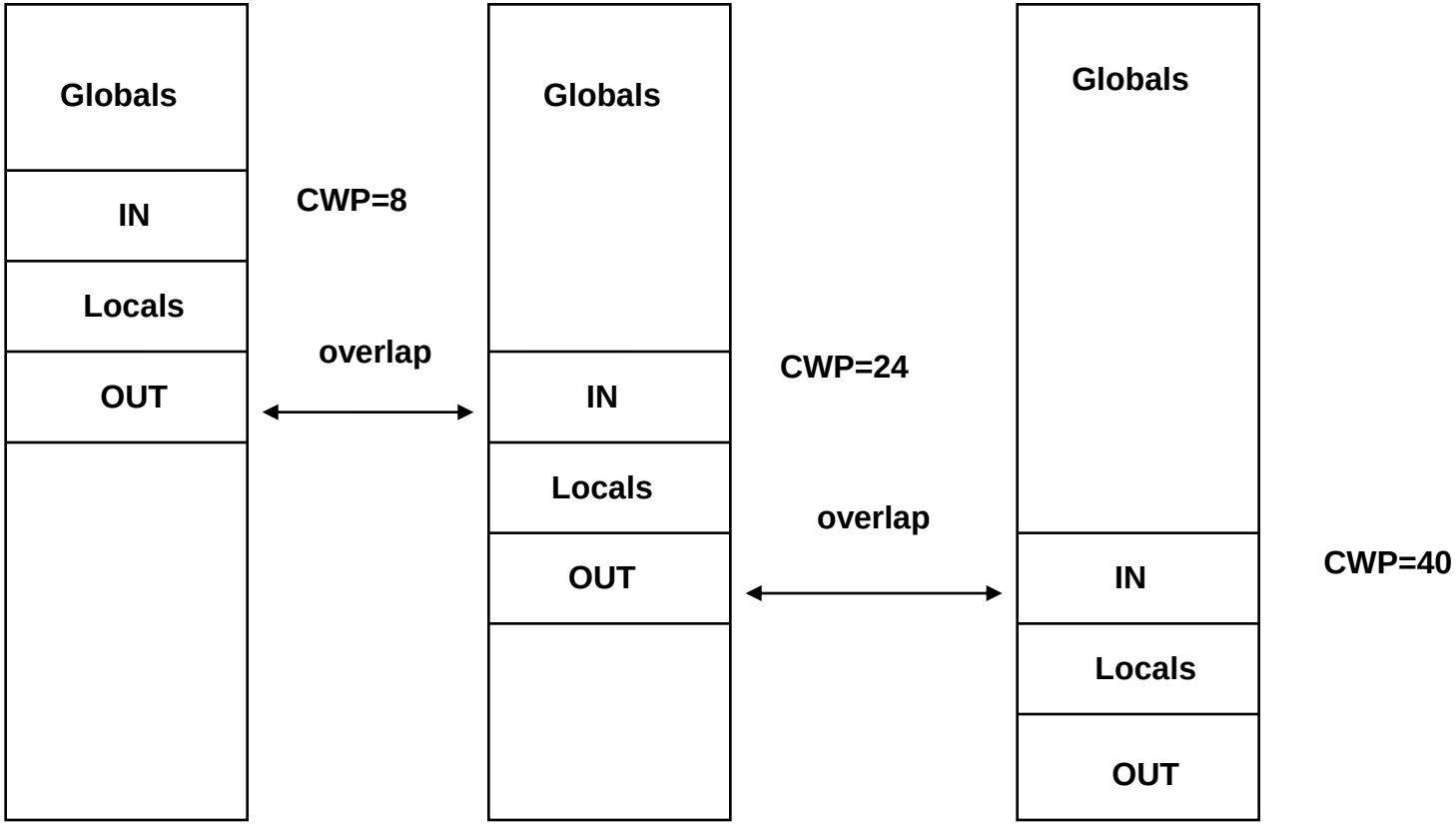
- Použitie registrov okna
  - Vstupné parametre
    - V CISC – parametre procedúr v zásobníku
    - V RISC – do registrov
    - Ak sa nezmestia – do zásobníka
    - Vo väčšine prípadov 8 registrov stačí
  - Lokálne premenné
    - Ak nebudú stačiť registre, použije sa zásobník
  - Výstupné premenné
    - Podobne: ak nebudú stačiť registre, použije sa zásobník

6. RISC použitie registrov (4/6)

- Ako to funguje?
- Zavedieme CWP (current window pointer), smerník na register
- Ak procedúra A volá procedúru B (obr.) zmenou CWP sa výstupy A stanú vstupmi B a B ich môže používať bez nutnosti prístupu k pamäti
- CWP funguje ako SP (stack-pointer)
- R0-R7 sú globálne premenné, ale ak sa objaví R8, predstavuje to hodnotu CWP
- Ďalšie podrobnosti:
  - R31 sa rezervuje na návratovú adresu
  - Dlhé slová sa ukladajú do zásobníka a do registrov sa ukladajú len smerníky na zásobník
  - Pri väčšom počte parametrov – pamäť
  - Príliš veľa vnorených volaní procedúr – vyvolá trap

  Trap handler – popresúva obsahy registrov do pamäte a rieši problém

6. RISC použitie registrov (5/6)



6. RISC použitie registrov (6/6)

Čo je lepšie registre, alebo CACHE na čipe?

Alokácia registrov

- Kompilátor priraďuje registre premenným
- Premenných môže byť viac ako registrov
- Premenná sa používa len v nejakej časti programu (je živá, live), mimo tejto časti nemusí byť uložená v registri (je mŕtva, dead)
- Alokácia registov a chromatické číslo grafu (programu priraďujeme graf)
- Ak je chromatické číslo grafu ≤ počet dostupných registrov, všetky premenné možno držať uložené v registroch a eliminovať tak potrebu LOAD, STORE

6. RISC alebo CISC

- Ktorý je lepší pre vykonávanie programov napísaných vo vyšších programovacích jazykoch (benchmarking)
- Koľko sa získa vďaka veľkému počtu registrov
- Čo sa píše ľahšie: kompilátory pre RISC alebo CISC
- ...

7. Spracovanie vstupu a výstupu (1/2) (I/O processing)

- I/O v počítači zabezpečuje I/O systém
- Úlohou I/O systému je prenášať informácie medzi CPU alebo hlavnou pamäťou a okolitým svetom
- I/O systém:
  - I/O zariadenia (periférie) (napr. SCSI scanner)
  - Radiče I/O zariadení (napr. SCSI radič)
  - Softvéru (napr. ovládače pre radič)
- Návrh I/O systému – základné problémy
  - CPU a I/O nemožno synchronizovať, dá sa len koordinovať
  - CPU je obyčajne omnoho rýchlejšie ako I/O zariadenie
    - I/O komunikujú s CPU asynchrónne
  - CPU – binárne kódovanie informácie
    - I/O – spolupracuje s človekom – treba kódovať a dekódovať

7. Spracovanie vstupu a výstupu (2/2) (I/O processing)

- Riadenie I/O operácií – globálne – CPU:
  - Výber I/O zariadenia a kontrola jeho pripravenosti
  - Inicializácia prenosu a koordinácia časovania I/O operácií
  - Prenos informácie
  - Ukončenie prenosu
- I/O zariadenia
  - Len vstup (napr. myš)
  - Len výstup (napr. monitor)
  - Obojsmerné (napr. pevný disk)
- Veľká variabilita konkrétnych riešení, budeme sa zaoberať len principiálnymi otázkami:
  - Spôsob, akým sa uskutočňujú I/O operácie
  - Základné mechanizmy prenosu údajov
  - Spájanie rozličných zariadení (interfacing)
  - I/O procesory

7.1. Prístup k I/O portom (1/2) (I/O accessing)

- Dva základné spôsoby
  - Memory mapped I/O
  - I/O mapped I/O
- **Memory mapped I/O:**
  - I/O porty sú pripojené k adresovej zbernici
  - Každé I/O zariadenie má svoje číslo, ktoré sa chápe ako pamäťová adresa (vstupné I/O je ako pamäť z ktorej sa číta a výstupné I/O ako pamäť do ktorej sa zapisuje)
  - Každá inštrukcia, ktorá sa odvoláva na pamäť, sa tak môže odvolávať na I/O port
  - Nevyžadujú sa špeciálne I/O inštrukcie
  - Pomalšie než druhé riešenie
  - Zmenšuje sa adresový priestor

7.1. Prístup k I/O portom (2/2)
(I/O accessing)

I/O mapped I/O:

- I/O porty sú nezávislé od pamäte
- CPU rozlišuje, či zapisuje/číta do/z pamäte alebo do/z I/O portu
- Na prenos informácie medzi CPU/pamäťou a I/O portami sa používajú špeciálne operácie INPUT a OUTPUT
- Tá istá adresa môže byť adresou pamäťového miesta aj I/O portu

7.2. Riadenie I/O operácií (1)

I/O operácie sa dajú klasifikovať podľa toho, odkiaľ je riadený prenos údajov:

1. Programom riadený I/O
2. I/O využívajúci prerušenia
3. DMA prenos

7.2. Riadenie I/O operácií (2) programmed I/O

- Programom riadený I/O
  - Najjednoduchšia priama metóda
  - Malé množstvo špeciálneho I/O hardvéru
  - CPU riadi celý prenos údajov
  - Existuje špeciálny program (postupnosť I/O inštrukcií) podľa ktorého sa
    - Inicializujú
    - Usmerňujú
    - Ukončujú I/O operácie
- Potrebný I/O hardvér:
  - Status register
  - Buffer register
  - Data counter
  - Buffer pointer

7.2. Riadenie I/O operácií (3) programmed I/O

- **Status register:** súčasný stav I/O zariadenia a údajov, ktoré sa majú preniesť
  - zariadenie: zapnuté / vypnuté, pripravené / obsadené, má sa čítať alebo zapisovať, chyba parity,...
  - typ a formát údajov, byte, reťazec, znaky,...
- **Buffer register:** uchováva údaje, ktoré sa majú preniesť
- **Data counter:** koľko údajov [B] sa má preniesť
  - pri prenose sa testuje na 0
- **Buffer pointer:** adresa pamäťového miesta, kam sa má zapisovať, alebo odkiaľ sa má čítať

7.2. Riadenie I/O operácií (4) programmed I/O

- Ako to funguje
  - Prenos sa uskutočňuje v cykle, počet opakovaní určuje data counter
  - Testuje sa status register, aby sa zistilo, či sa môže uskutočniť prenos
    - Pri čítaní:
      - načíta sa z I/O do buffer registra
      - obsah buffer registra sa zapíše do pamäte na adresu, ktorá je v registri buffer pointer
    - Pri zápise:
      - sa z pamäťového miesta, ktorého adresa je v registri buffer pointer zapíše do buffera
      - obsah buffer registra sa zapíše do I/O
  - Aktualizuje sa Data counter (--) a Buffer pointer (++)
- Jednoduché ale pomalé, zaťažuje to CPU, veľa sa testuje, kým sa niečo spraví

7.2. Riadenie I/O operácií (5)
Princíp interrupt I/O

- I/O zariadenia (ale aj iné) žiadajú CPU o odpoveď
- Namiesto toho, aby CPU sústavne kontrolovalo, či niektoré z nich nechce čítať/zapisovať využíva interrupt
- I/O zariadenie, ktoré „chce“ napr. poslať údaje do CPU alebo pamäte pošle CPU signál INTR (interrupt request)
- CPU má žiadateľov rozdelených do dvoch aktuálnych kategórií: podstatní a nepodstatní
  - Podstatným vyhovuje okamžite, nepodstatní musia počkať
- **Maskovanie:** v programe je možné nastaviť pomocou interrupt enable a interrupt disable inštrukcie
- **Maskovateľné prerušenie** musí čakať
- **Nemaskovateľné prerušenie** sa vykoná okamžite

7.2. Riadenie I/O operácií (6) ošetrenie súčasných žiadostí o prerušenie

- v okamihu vyhodnocovania žiadosti o prerušenie môže existovať viacero žiadostí o prerušenie
- CPU musí rozhodnúť, ktorej vyhovie
- Viacero spôsobov riešenia:
  - Polling
  - Vector interrupt

- **Polling:** ak sa objaví žiadosť o prerušenie (INTR), CPU kontroluje stavové bity jednotlivých zariadení, ktoré mohli žiadať o prerušenie
  - od zariadenia s najvyššou prioritou po zariadene s najnižšou prioritou
- Postupné zisťovanie hodnôt stavových bitov jednotlivých zariadení je zdĺhavé

7.2. Riadenie I/O operácií (7) ošetrenie súčasných žiadostí o prerušenie

- **Vector interrupt:**
  - Zariadenie pošle INTR
  - CPU pošle signál INTA (interrupt acknowledge), keď je pripravený spracovať ďalšie prerušenie
  - Zariadenie, ktoré vyslalo INTR, pošle na údajovú zbernicu vektor (adresu, na ktorej je interrupt handler na spracovanie daného prerušenia)
  - Ak bolo žiadostí o prerušenie viac, vyberie sa spomedzi nich tá s najväčšou prioritou
  - Na spracovanie súčasných žiadostí o prerušenie sa používa prioritný kóder alebo metóda daisy chain

- **Prioritný kóder:** na vstupy prioritného kódera sú v poradí od najvýznamnejšieho po najmenej významný pripojené bity (napr.) vektora, ktorý je výsledkom prieniku masky a vektora žiadostí o prerušenie jednotlivých zariadení. Ak je vektor nenulový prioritný kóder vypočíta pozíciu prvej jednotky v ňom.

7.2. Riadenie I/O operácií (8) ošetrenie súčasných žiadostí o prerušenie

- **Daisy chain:** (princíp) zariadenia, ktoré môžu vyslať INTR sú navzájom lineárne prepojené v poradí zodpovedajúcom ich priorite.
  - CPU vyšle INTA, prvému zariadeniu, ak toto zariadenie žiadalo o prerušenie, pošle na údajovú zbernicu vektor (adresu interrupt handlera) a zablokuje šírenie INTA.
  - Ak zariadenie nežiadalo o prerušenie, pošle INTA ďalšiemu zariadeniu s nižšou prioritou
  - Keďže niektoré zo zariadení o prerušenie požiadalo, INTA sa k nemu napokon dostane
  - Ak o prerušenie žiadalo viacero zariadení, tak sa INTA dostane ako k prvému k tomu z nich, ktoré malo najvyššiu prioritu
- Daisy chain sa používa aj v iných súvislostiach, napr. pri vyhodnocovaní súčasných žiadostí o priradenie zbernice

7.2. Riadenie I/O operácií (9) direct memory access, DMA

- Interrupt I/O podstatne zefektívňuje vstup a výstup údajov, ale nepostačuje na obsluhu periférií, schopných prenášať veľké množstvá údajov (napríklad pevný disk)
- Medzi pamäťou a periférnym zariadením sa obvykle prenášajú veľké množstvá údajov
- Takýto prenos sa rieši metódou direct memory access (DMA) a je riadený pomocou DMAC, radiča DMA
- **Podstata DMA:**
  - CPU inicializuje DMA kanál, potom prenos riadi DMAC bez účasti CPU
  - Vďaka „obchádzaniu“ CPU sa dosahujú prenosové rýchlosti rádovo rovné cyklu hlavnej pamäte
  - I/O prenáša veľký blok údajov v jednej súvislej operácii
    - DMA block transfer
  - Počas DMA prenosu môže dôjsť ku kolízii (DMAC aj CPU môžu potrebovať zbernicu, alebo pristupovať k pamäti)

7.2. Riadenie I/O operácií (10) direct memory access, DMA

- Riešenia kolízií (CPU - DMAC) pri prístupe k pamäti:
  - Dual port memories
  - Cycle stealing (kradnutie cyklov)
- **Cycle stealing:**
  - DMA zariadenie má priradený cyklus pamäte na prenos údajov
  - Počas tohto cyklu CPU nemôže pristupovať do pamäte
  - Prenesie sa niekoľko slov a riadenie sa vráti CPU
  - Ak CPU nepotrebuje pracovať s pamäťou, odovzdá riadenie údajovej zbernice DMAC
  - Ak je DMAC synchronizovaný s CPU, počas execute cyklu, keď CPU nepotrebuje pamäť, môže uskutočniť DMA prenos (ukradnúť cyklus)
  - **Transparentný DMA:** ak je dostatok voľných cyklov a DMA prenosy sa stihnú uskutočňovať v tých cykloch, keď CPU nepracuje s pamäťou

7.2. Riadenie I/O operácií (11) DMAC

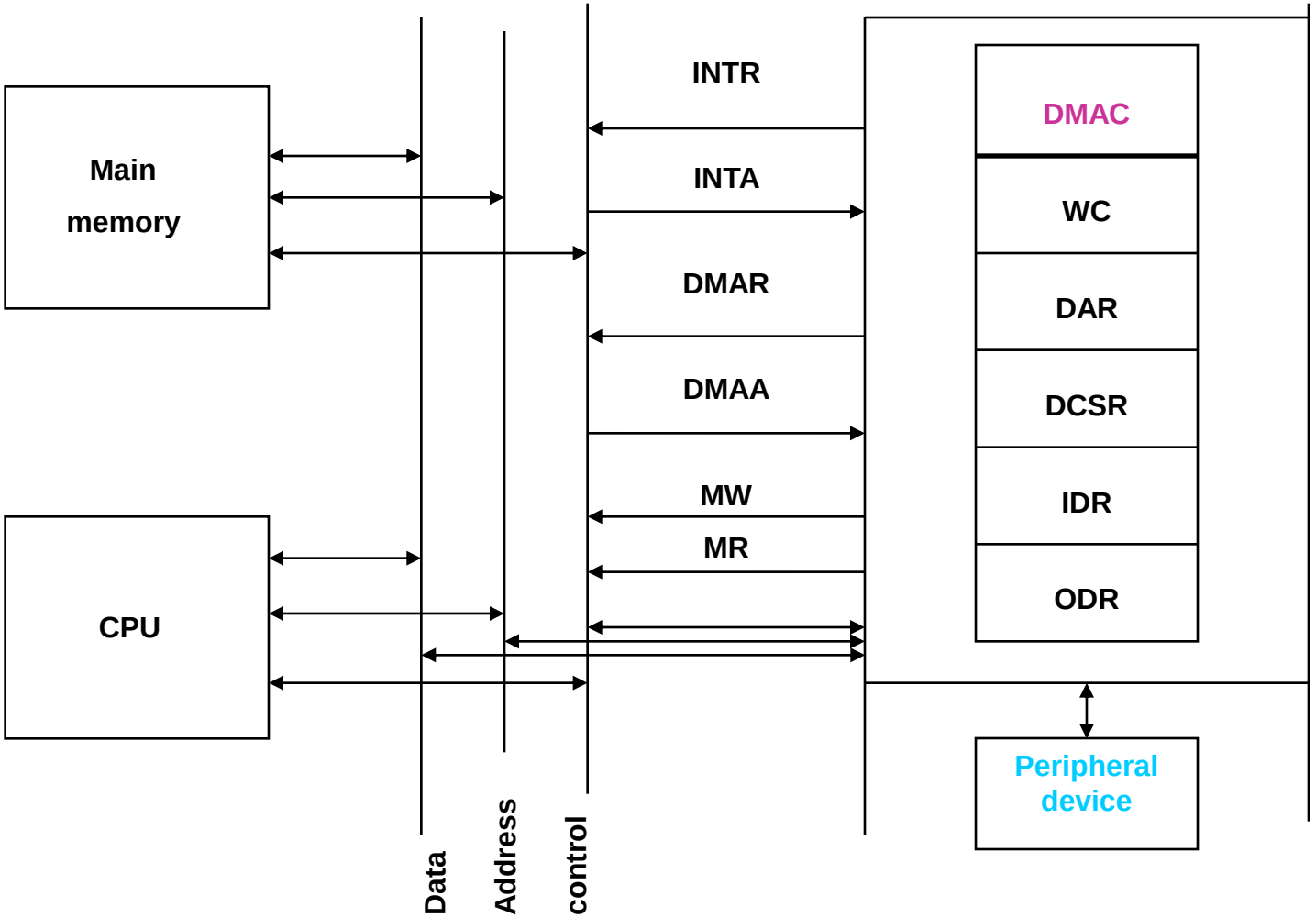
Direct memory access controller (DMAC):

- Riadi prenos údajov v móde DMA
- Môže obsluhovať jedno alebo niekoľko I/O zariadení
- Pozostáva z 5 registrov a riadiacich obvodov:
  - **WC = word counter:** register, ktorý obsahuje počet slov, ktoré sa majú preniesť (po každom prenose sa WC dekrementuje o 1)
  - **DAR = DMA address register:** register obsahujúci adresu ďalšieho slova, ktoré sa má preniesť (adresu v pamäti, odkiaľ sa má čítať, alebo kam sa má zapisovať); automaticky sa inkrementuje o 1
  - **ODR = output data register:** obsahuje slovo, ktoré sa má poslať na I/O zariadenie
  - **IDR = input data register:** do tohto registra sa ukladá slovo, ktoré prichádza z I/O zariadenia
  - **DCSR = control/status register:** popisuje stav DMAC a zariadení pripojených k DMAC

7.2. Riadenie I/O operácií (12)
DMAC

– DCSR:
  - Device enable flag
  - Done/ready flag
  - Interrupt enable flag
  - Error bits
  - Device status bits

7.2. Riadenie I/O operácií (13)
DMAC



7. 2. Riadenie I/O operácií (14)
DMA handshaking protocol

- DMA prenos:
  - CPU spracuje INTR a ak je možný DMA prenos inicializuje ho (nastaví WC, DAR a DCSR) a pošle signál INTA
  - DMAC vyšle DMAR (DMA request) signál CPU, aktivizuje
    - MR (memory read) alebo
    - MW (memory write) a
    - Pripraví registre IDR alebo ODR
  - Ak je možný DMA prenos CPU vyšle signál DMAA (DMA acknowledge) uvoľní údajovú zbernicu (CPU môže uvoľniť zbernicu po každom ukončenom prenose a DMAA nemusí byť len odpoveďou na DMAR)
  - Potom sa v cykle, riadenom WC uskutoční DMA prenos

7.3. Prenos údajov (1)

- Základné otázky
  - Formát prenosu údajov
    - Sériový
    - Paralelný
  - Spôsob prenosu údajov
    - Synchronizovaný
    - Asynchrónny

7.3. Prenos údajov (2)

Sériový prenos

- 1 prenosová linka
- Údaje treba transformovať z paralelného na sériový tvar pomocou interface a naopak
- Prenos na väčšie vzdialenosti
- Pripojenie pomalších zariadení
- Lacnejší ale pomalší

Paralelný prenos

- Viac paralelných prenosových liniek
- Prenáša sa viac bitov naraz
- Prenos na kratšie vzdialenosti
- Pripojenie rýchlejších zariadení
- Rýchlejší ale drahší

7.3. Prenos údajov (3)

- Okrem údajových liniek sa pri prenose používajú aj riadiace linky
- Pomocou riadiacich liniek sa prenášajú riadiace signály

Synchronizovaný prenos

- CPU na adresovú zbernicu pošle adresu zariadenia
- Na riadiacej zbernici nastaví WRITE = 1
- Na údajovú zbernicu dá údaje
- I/O zariadenie musí údaje zo zbernice prečítať, kým je WRITE = 1
- Podobne pri čítaní údajov z I/O zariadenia (CPU nastavuje na riadiacej zbernici READ = 1)

Problémy

- I/O zariadenia musia informáciu spracovať, kým je READ alebo WRITE = 1
- I/O zariadenia majú rozličné rýchlosti:
  - Synchronizačné signály budú mať rozličnú dĺžku
  - Dĺžka synchronizačného signálu stačí pre najpomalšie zariadenie

7.3. Prenos údajov (4)

Asynchrónny prenos údajov

- Po riadiacich linkách sa prenášajú signály koordinujúce činnosť vysielajúceho (V) a prijímajúceho (P) zariadenia
- Riadiace signály: protokol (hanshaking protocol)
- Všeobecný príklad:
  - V: request
  - P: ak je pripravený : acknowledge
  - V: inicializácia prenosu, samotný prenos
  - P: po prijatí údajov: Data valid (data received)
  - V: ukončenie prenosu
- Príklad zápisu údajov na I/O zariadenie:
  - CPU: údajová zbernica = údaje, adresová = adresa I/O zariadenia, riadiaca: WRITE = 1
  - I/O zariadenie: prečíta údaje z údajovej zbernice, po prečítaní nastaví riadiaci signál DATA RECEIVED na 1
  - CPU nastaví WRITE = 0, odstráni adresu I/O zariadenia z adresovej zbernice a údaje zo zbernice
  - I/O nastaví DATA RECEIVED = 0

7.3. Prenos údajov (5)

Porovnanie synchronizovaného a asynchrónneho prenosu údajov:

- **Synchronizovaný prenos**
  - Rýchlejší
  - Menej riadiacich liniek
  - Problémy s rozličnými rýchlosťami periférií

- **Asynchrónny prenos**
  - Pomalší
  - Komplikovanejšie riadenie
  - Flexibilnejší (možnosť pripojenia zariadení s rozličnými rýchlosťami)

7.4. Princípy operácií prenosu údajov (programmed I/O)

1. CPU pravidelne testuje stavový bit I/O zariadenia, aby zistila, či je zariadenie pripravené na prenos údajov
2. Ak áno, CPU nastaví (napr.) WRITE=1, slovo z pamäte sa uloží do registra CPU a potom sa pomocou operácie OUT prenesie na I/O zariadenie. Pri čítaní z I/O CPU nastaví READ=1 a pomocou operácie IN prenesie slovo z I/O zariadenia do CPU registra a odtiaľ následne do pamäte
3. CPU čaká, kým I/O zariadenie oznámi, že operácia prenosu údajov je ukončená

7.4. Princípy operácií prenosu údajov (interrupt I/O).

1. I/O zariadenie požiada o prerušenie (vyšle INTR)
2. CPU zistí, či je žiadosť o prerušenie maskovaná.
   - Ak áno, v danom okamihu sa ňou nezaoberá.)
   - Ak nie, CPU zistí, či prebiehajúci program nemá vyššiu prioritu, ako žiadosť o prerušenie.
     - Ak áno, žiadosťou sa v danom okamihu nezaoberá.
     - Ak nie, zistí, či žiadosť I/O má najvyššiu prioritu spomedzi všetkých aktuálnych žiadostí o prerušenie (ak nie, žiadosťou I/O sa v danom okamihu nezaoberá, ak áno **goto** 3)
3. CPU pošle I/O zariadeniu INTA a prečíta interrupt vektor
4. CPU uloží PSW do zásobníka
5. CPU prejde na adresu príslušného interrupt handlera
6. Po obslúžení prerušenia CPU prečíta hodnotu PSW zo zásobníka a vracia sa pôvodne vykonávanému programu

7.4. Princípy operácií prenosu údajov (DMA)

CPU inicializuje registre

1. WR = počet slov, ktoré sa majú preniesť
2. DAR = počiatočná adresa v pamäti, odkiaľ sa má čítať, alebo kam sa má zapisovať
3. DSCR
4. Keď sa naplní IDR (alebo vyprázdni ODR) nastaví sa DMAR. Potom (zápis)
   - MAR:=(DAR)
   - MBR:=(IDR)
5. CPU pošle DMAA signál a DMAC
   - WC:=(WC) -1
   - DAR:=(DAR) + 1
   - WC = 0 ?
   - Ak WC = 0, DMAC generuje signál INTR, signalizujúci ukončenie procesu
   - Ak WC ≠ 0 **goto** 4

7.5. Device interfacing

- Parametre periférnych zariadení a CPU sa výrazne líšia
  - rýchlosť
  - formát údajov
- Periférne zariadenia nemožno na počítač pripojiť priamo
- Na prekonanie týchto rozdielov sa používa
  - Interface (rozhranie)
  - Riadiace obvody
  - Riadiaci softvér
- Interface = hardvér potrebný na pripojenie periférneho zariadenia a jeho riadiacich obvodov na zbernicu
- Device controller (radič zariadenia)
  - spracováva status a príkazy periférneho zariadenia
- Základná funkcia interface:
  - synchronizovať prenos údajov medzi perifémymi zariadeniami a CPU
  - ≠ synchronizácia I/O zariadenia a CPU

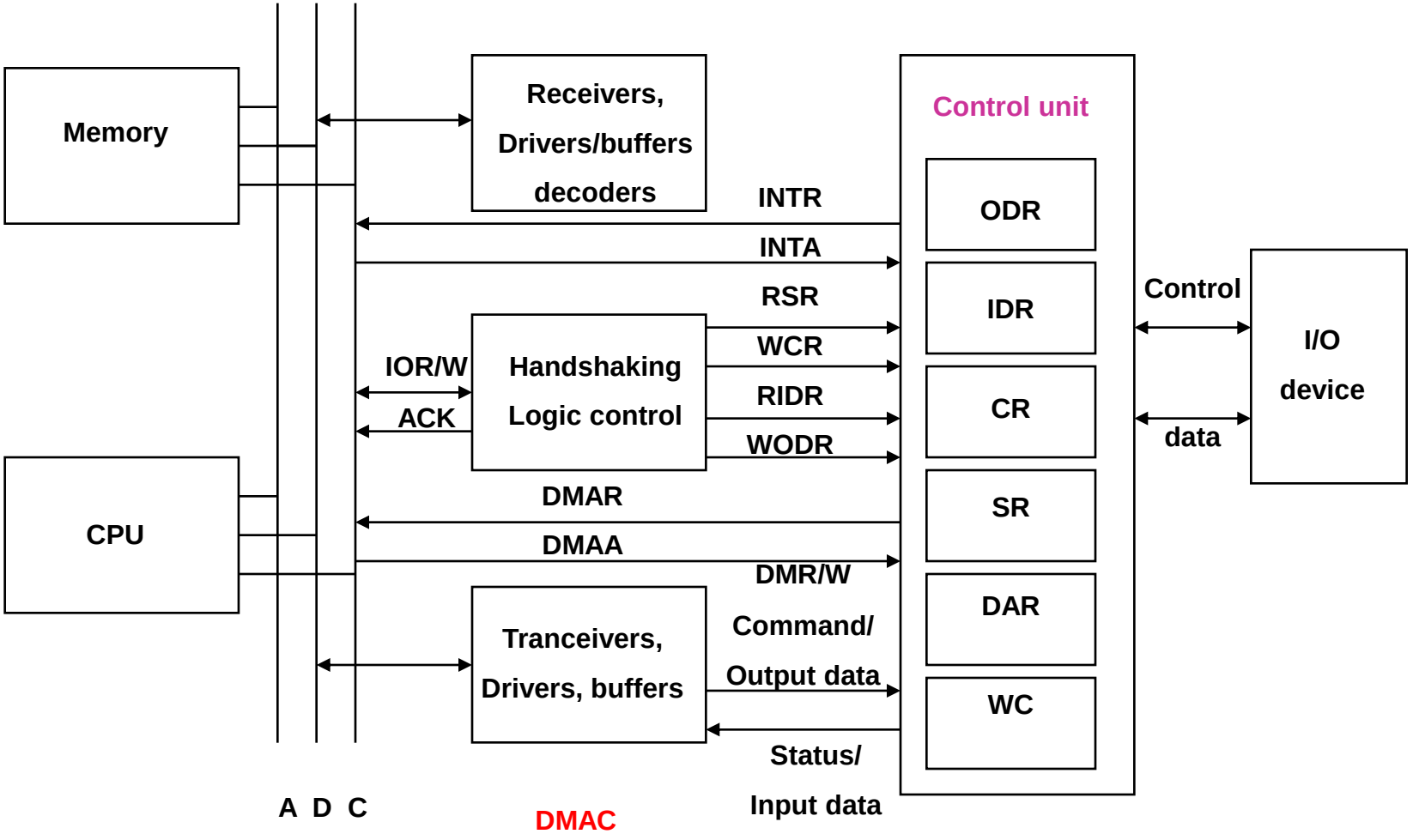
7.5. Device interfacing

- Interface – dve časti:
  - Device dependent
    - obsluhujúca periférne zariadenie
  - Device independent
    - obsluhujúca CPU (pripájajúca interface unit na zbernicu)
- Vo väčšine prípadov sa používajú štandardné interfaces, v špeciálnych prípadoch treba navrhovať vlastné
  - UART = Universal Asynchronous Receiver Transmitter
  - USART = Universal Synchronous Asynchronous Receiver Transmitter

7.5. Device interfacing

- Funkcie interface:
  1. Sprístupňovať status periférnych zariadení CPU
  2. Schopnosť generovať INTR a/alebo DMA
  3. Signalizuje CPU ukončenie operácie a/alebo chyby počas prenosu
  4. Prenos riadiacich príkazov CPU periférnemu zariadeniu
  5. Ukladanie údajov do buffra
     - pri prenose medzi CPU/pamäťou a periférnym zariadením
  6. Test parity a hlásenie chyby, niekedy aj opravovanie chýb
  7. Kódovanie a dekódovanie údajov
  8. Konverzia medzi sériovým a paralelným formátom údajov

7.5. Device interfacing
štruktúra interface (1)



7.5. Device interfacing štruktúra interface (2)

- Tri zbernice:
  - A = Address
  - D = Data
  - C = Control
- Na každú zbernicu je pripojený
  - Bus receiver – register, ktorý uchováva vstupné údaje tak dlho, ako je potrebné
  - Bus transceiver – obvod, ktorý sa používa pre obojsmernú zbernicu a môže slúžiť ako receiver alebo transceiver/transmitter
  - Bus driver/buffer – slúži na umiestňovanie údajov na zbernicu (tristabilné zariadenie, ktoré sa môže odpojiť od zbernice)
- Interface adresovej zbernice obsahuje aj dekóder (aby sa dalo určiť, ktorý register adresuje daná I/O inštrukcia)

7.5. Device interfacing štruktúra interface (3)

- Riadiaca zbernica má viacero liniek:
  - Handshaking signals (INTR,INTA,ACK)
  - I/O read/write
  - DMA signály (DMAR,DMAA, DMA memory read/write)
- Časť DMA:
  - CR (control register)
    - ukladajú sa doň inštrukcie a informácia pre periférne zariadenia
  - SR (status register) – stav zariadenia a informácia o chybách
  - IDR, ODR – vstupný a výstupný buffer

- **Vstup údajov**
  - Periférne zariadenie pošle slovo, to sa uloží do IDR
  - V SR sa nastaví flag **data_ready**
  - Ak je **interrupt_enable** flag v CR nastavený na 1, pošle sa CPU žiadosť o prerušenie INTR, aby CPU umožnil vstup údajov (interrupt I/O)

7.5. Device interfacing štruktúra interface (4)

- **Výstup údajov**
  - Výstupné slovo sa uloží do ODR
  - Testuje sa pripravenosť periférneho zariadenia (v SR má nastavený bit **device_ready**)
  - Ak je **device_ready**, slovo z ODR sa pošle na periférne zariadenie
  - Ak **device_ready**=0 (zariadenie nie je pripravené, alebo zariadenie nie je zapnuté) čaká sa, alebo sa vyhlási chyba
- Operácie vstupu a výstupu riadi **handshaking logic unit** pomocou 4 signálov:
  - WCR – write control register
  - RSR – read status register
  - RIDR – read input data register
  - WODR – write output data register

7.6. System resident I/O processors

- CPU je najzložitejšia, najrýchlejšia a najdrahšia časť počítača
- Je zbytočné zaťažovať ju I/O operáciami
- I/O operácie riadi špeciálny radič, I/O kanál
- I/O kanál môže obsluhovať jedno alebo viacero I/O zariadení
- Ďalšie zdokonalenie:
  - I/O operácie riadi I/O procesor IOP
  - IOP je univerzálny procesor, určený na spracovanie I/O relatívne nezávisle od CPU
- Problém: viacero procesorov (CPU,IOP)
  - multiprocesorová konfigurácia, bude potrebné riešiť prideľovanie spoločných zdrojov a koordináciu činnosti

7.6. I/O kanál (1)

- I/O kanál je špeciálny procesor na sprostredkovanie spojenia viacerých I/O zariadení s pamäťou
  - DMAC obsluhuje len jedno I/O zariadenie
- Vie odhaľovať a opravovať chyby, formátovať údaje, kódovať a dekódovať
- Systém obsahujúci I/O kanály je zvyčajne hierarchicky usporiadaný:
  - každý I/O kanál obsluhuje buď jedno rýchle, alebo
  - niekoľko pomalých periférnych zariadení
- I/O kanály sú pripojené na spoločnú (systémovú zbernicu)
- O tom, kto bude ovládať zbernicu rozhoduje bus controller
- Činnosť I/O kanálu:
  1. Vyberie I/O zariadenie a preverí jeho stav
  2. Vykonáva program na realizáciu I/O operácií
  3. Definuje oblasti v hlavnej pamäti, kam sa zapisuje, alebo odkiaľ sa číta
  4. Poskytuje možnosti formátovania, kódovania a dekódovania údajov
  5. Control end-of-transfer činnosti

7.6. I/O kanál (2)

- Jeden I/O kanál môže obslúžiť viacero I/O zariadení
- **Multiplexer channel:** funguje tak, že prepína medzi viacerými I/O zariadeniami a údaje z nich prichádzajúce dáva dokopy (character interleaving):
  - Zariadenie A: abcd ...
  - Zariadenie B: ABCD ...
  - Zariadenie C: ❁❂✻❄ ...
  - aA❁bB❂cC✻dD❄...
- Keď sa multiplexer channel dostane do ťažkostí (nestíha) prepne sa do havarijného módu, nechá jedno I/O zariadenie dokončiť prenos a neprepína medzi rozličnými I/O zariadeniami

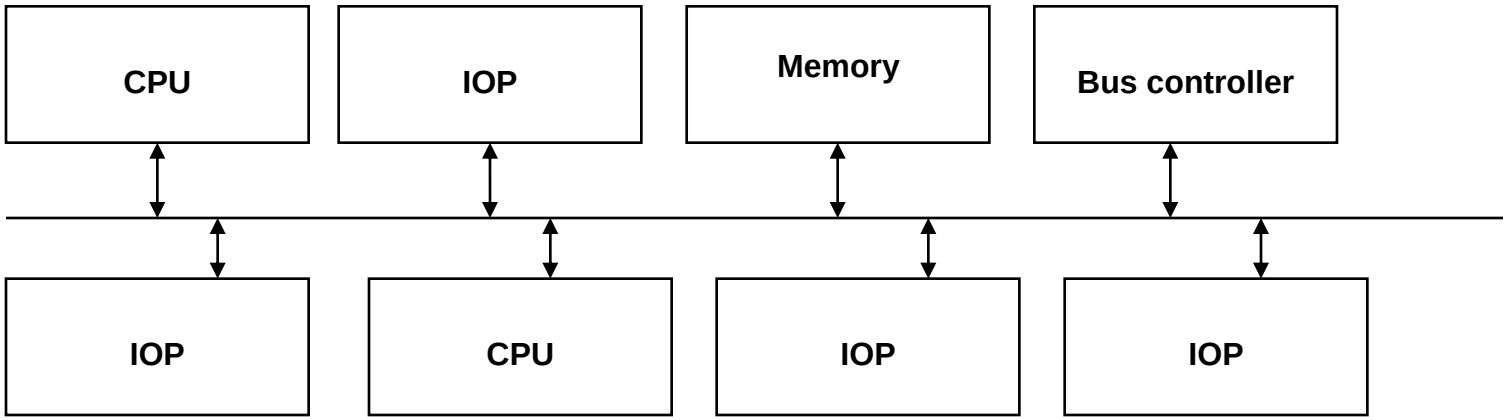
7.6. I/O kanál (3)

- Iné riešenie: **selector channel**
  - Hardvérová organizácia podobná ako DMAC
    - byte counter, MAR, MBR, DAR, S/CR
  - Vyberie I/O zariadenie, ktorému priradí I/O kanál. Toto priradenie zostáva v platnosti, kým dané zariadenie neukončí prenos údajov

- **Block multiplexer channel:**
  - Kombinácia multiplexer a selector channel
  - Funguje ako rýchly multiplexer channel, ale namiesto znakov sa prenášajú bloky údajov
  - Používa sa na zariadenia ako disky, pásky, kde je potrebný nejaký čas na vyhľadanie informácie, ale potom je už prenos údajov rýchly

7.7. Konfigurácia multiprocesorového systému (1)

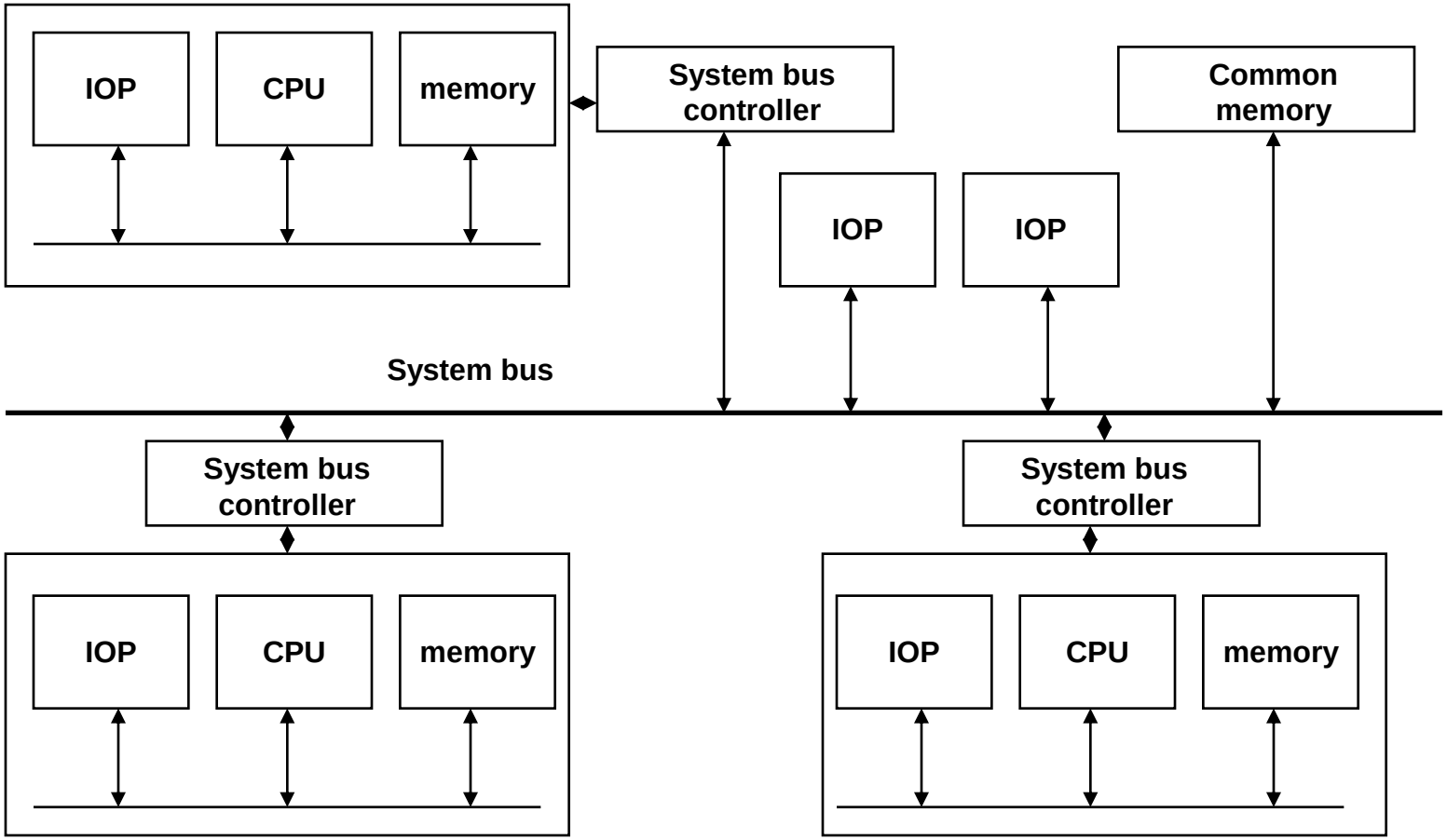
- Logickým zavŕšením vývoja I/O radičov je použitie univerzálneho procesora ako I/O procesora
- Problém – ako zapojiť viacero procesorov do fungujúceho celku
- Najjednoduchšie riešenie: spoločná zbernica



7.7. Konfigurácia multiprocesorového systému (2)

- **Multiprocesorový systém so spoločnou zbernicou:**
  - Prideľovanie zbernice – bus controller
  - Na pridelenie zbernice sa čaká
  - Vyhodnocovanie požiadaviek na pridelenie zbernice
    - Sériové
    - Paralelné
    - Kombinované
- **Multiprocesorový systém s duálnou zbernicou:**
  - (Dual bus multiprocessor organization)
  - Každý má svoju vlastnú internú zbernicu, pamäť aj IOP
  - Prostredníctvom System bus controller je pripojený na systémovú zbernicu
  - Takýto systém je výkonnejší, lebo v čase, keď počítač čaká na pridelenie spoločných zdrojov (systémová zbernica) môže čosi robiť s lokálnymi údajmi

7.7. Konfigurácia multiprocesorového systému (3)

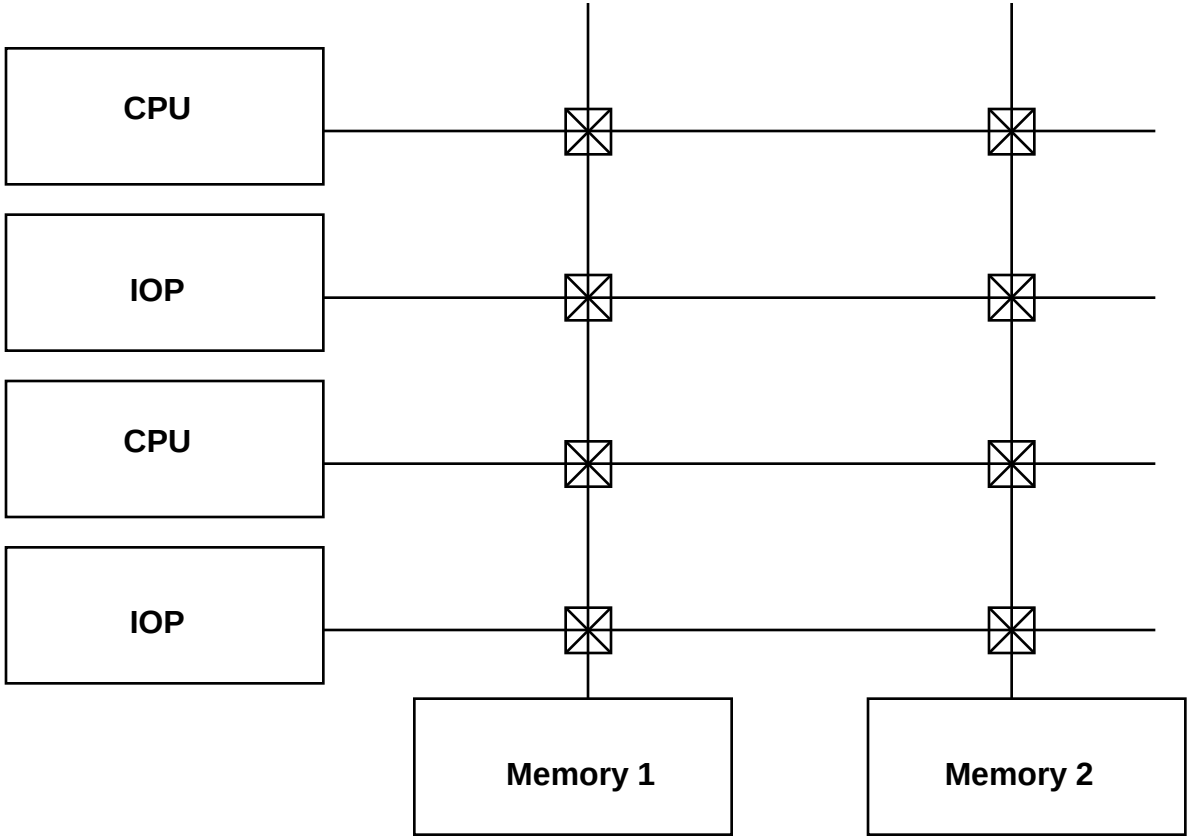


7.7. Konfigurácia multiprocesorového systému (4)

- **Switching (crossbar) matrix interconnection scheme:**
  - Prepínač sa aktivizuje, ak procesor umiestni na zbernicu číslo pamäťového modulu
  - Ak daný modul nie je už priradený, prepínač spojí procesor s pamäťou
  - Ďalšie možné riešenie: pamäte s viacerými portami – drahé

- Delenie multiprocesorových systémov:
  - Voľne viazané (veľké vzdialenosti)
  - Silne viazané (malé vzdialenosti)
    - na vzájomnú komunikáciu používajú o.i. Mailbox (poštovú schránku)

7.7. Switching (crossbar) matrix interconnection scheme



7.8. I/O Processor (IOP) (1)

- Je to síce autonómny procesor, ale jeho funkcie závisia od toho, ako je multiprocesorový systém organizovaný
- Systém môže byť organizovaný ktorýmkoľvek z vyššie uvedených (a zrejme aj iných) spôsobov
- IOP môže byť aj (trochu) závislý od CPU – vtedy potrebuje CPU na odštartovanie I/O operácií a rozhodovanie o I/O činnostiach
- IOP môže byť viazaný na pevnú skupinu I/O zariadení, alebo pomocou crossbar matice pripojený na viacero I/O zariadení

7.8. I/O Processor (2)

- IOP je univerzálny procesor použitý na špeciálne účely (riadenie I/O operácií)

- Inštrukčný súbor IOP pozostáva z
  - Inštrukcií na prenos údajov
  - Univerálnych inštrukcií
    - aritmetické, logické a iné inštrukcie všeobecného charakteru
  - Riadiacich inštrukcií pre I/O zariadenia

- Formát inštrukcie IOP (na prenos údajov)
  - Operačný kód (typ požadovanej operácie prenosu)
  - Adresa pamäte
  - Word count (počet slov)
  - Control field (v inštrukciách určených pre špeciálne I/O zariadenia)
  - Status field

7.8. Komunikácia CPU - IOP

- Komunikačný protokol typu handshaking, používajú mailbox
  - **IOP**→ **CPU** : žiadosť o prerušenie (INTR)
  - **CPU** → **IOP**: INTA + inštrukcia STATUS_I/O
  - **IOP**: pošle na miesto určené inštrukciou STATUS_I/O svoj STATUS WORD;
  - **CPU**: Ak je I/O zariadenie READY, CPU → IOP: inštrukcia START_I/O
  - **IOP**: ovládne zbernicu (cycle stealing) a využíva DMA zariadenie na prenos údajov
  - CPU môže zastaviť prenos údajov pomocou inštrukcie STOP_I/O, IOP môže prerušiť činnosť CPU pomocou INTR,...
  - Je to zložité a v každom multiprocesorovom systéme to môže byť riešené trocha inak

8. Pamäť

- Jedna zo základných častí počítača
- Funkcia pamäte: uchovávanie informácie (údajov aj programov)
- Rozdelenie pamätí:
  - Hlavná (operačná) pamäť
  - Pomocná pamäť
- Operačná pamäť
  - Uloženie programov a údajov, ktoré majú byť spracované CPU
  - Musí byť dostupná aj používateľovi, aj CPU
  - Vykonávanie programu je spojené so získavaním údajov (inštrukcií a ich operandov) z pamäte – rýchlosť počítača závisí aj od operačnej pamäte
  - Veľká operačná pamäť – viacrozmerná, pomalšia
  - Malá operačná pamäť – jednorozmerná, rýchlejšia
  - Realizácia operačnej pamäte – integrované obvody

8. Pamäť

- Pomocná pamäť – pomalšia, lacnejšia, energeticky nezávislá, s väčšou kapacitou
- Slúži na uchovávanie údajov, ktoré CPU bezprostredne nepoužíva, prenos údajov medzi počítačmi, archivovanie,...
  - SAM (Sequential Access Memory) – magnetická páska
  - DAM (Direct Access memory) – disk, disketa CR ROM

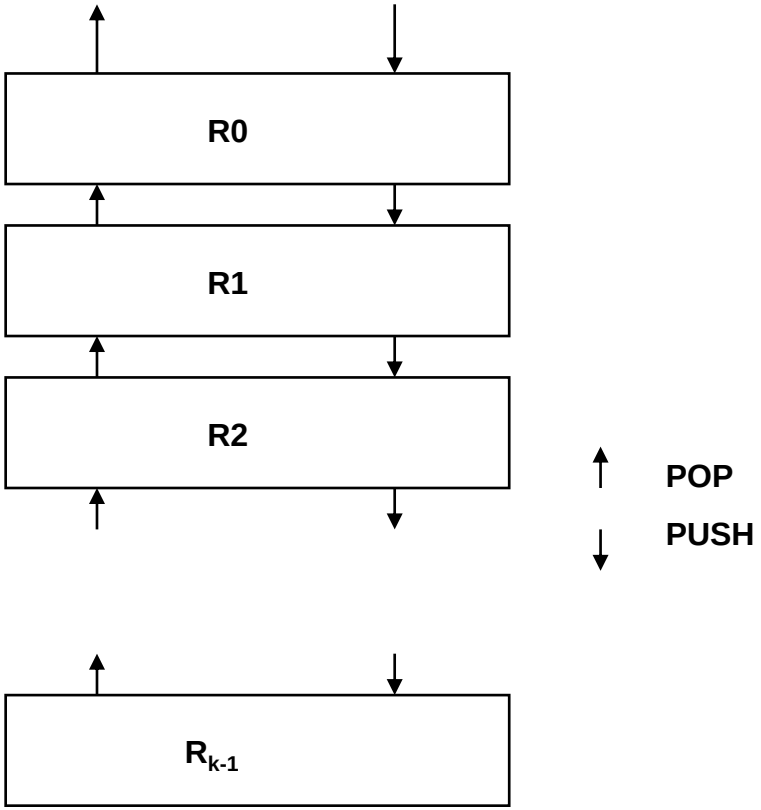
8.1. Parametre pamäťových systémov

1. **Capacity**
   - Kapacita udáva, koľko bitov je pamäť schopná uchovať
2. **Access time** (vybavovacia doba)
   - čas potrebný na prístup k údajom
3. **Data transfer rate**
   - počet bitov, ktoré sa dajú prečítať za sekundu
4. **Cycle time** (cyklus pamäte)
   - koľkokrát za sekundu možno pristúpiť k pamäti
5. **Cost** – cena/bit

8.2. Zásobníková pamäť

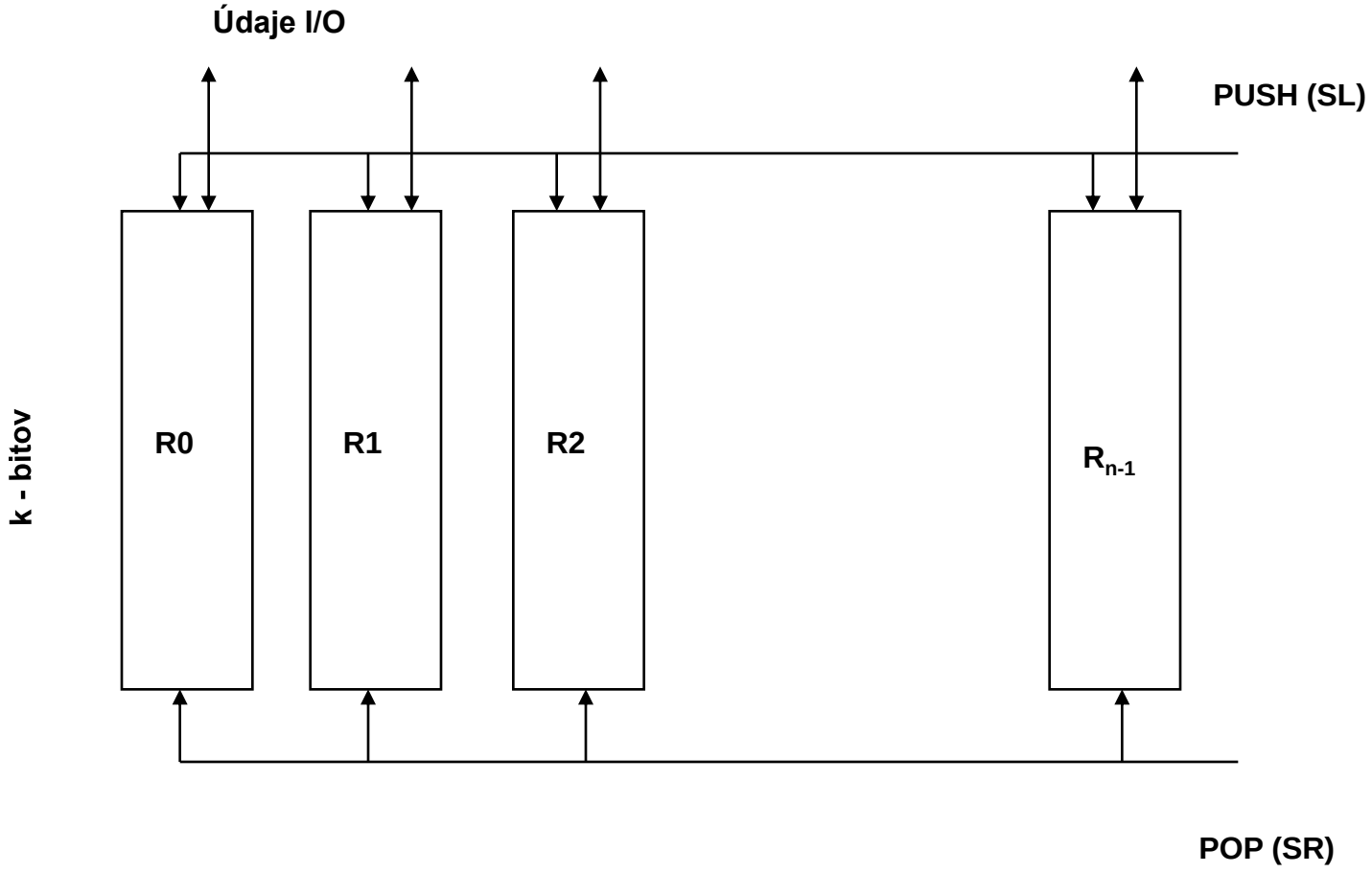
- Slúži najmä na uchovávanie údajov pri rekurzívne volaných procedúrach, podprogramoch, vnorených prerušeniach, vyhodnocovaní aritmentických výrazov
- Dá sa realizovať hardvérovo, softvérovo aj kombinovane
- podstata:
  - Údaje sa ukladajú na vrch zásobnika (operácia PUSH)
  - Údaje sa čítajú z vrchu zásobníka (POP)
  - Uloženie posunie vrch zásobníka o +1
  - Pri čítaní sa odstráni najvrchnejší údaj (-1)
  - Zásobník predstavuje údajovú štruktúru LIFO (last in, first out)
  - Dve pomocné logické funkcie FULL(stack) a EMPTY(stack) kontrolujú, či nechceme pridávať do plného, alebo čítať z prázdneho zásobníka

8.2. Hardvérová implementácia zásobníka (1)



- Pomocou registrov:
  - k n-bitových registrov s paralelným zápisom a čítaním
  - Register predstavuje 1 položku zásobníka
- Iné riešenie:
  - N posuvných k-bitových registrov
  - PUSH a POP sa realizujú pomocou SL a SR
  - Hodnotu i-tej položky v zásobníku predstavujú i-te bity všetkých registrov

8.2. Hardvérová implementácia zásobníka (2)

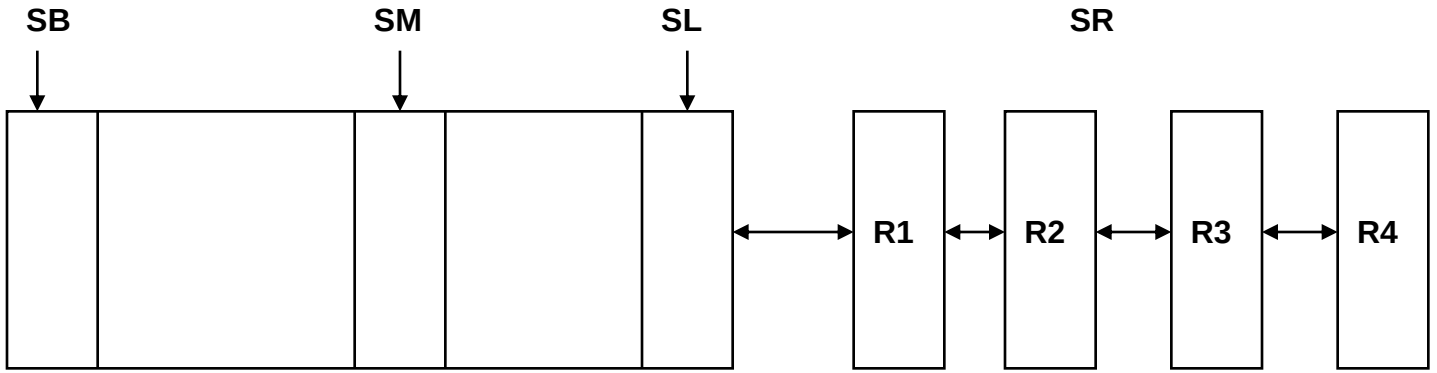


8.2. Implementácia zásobníka pomocou RAM

- V pamäti je vyhradený priestor na zásobník
- Pri operáciách POP a PUSH sa údaje fyzicky nepresúvajú, mení sa len adresa vrchu zásobníka, ktorá je uložená v registri SP (stack pointer)
- Na manažovanie zásobníka sú potrebné ešte ďalšie dva údaje: stack base (SB) a SL (stack limit)
  - SB = adresa dna zásobníka
  - SL = veľkosť zásobníka
- Hardvérová realizácia zásobníka je rýchla a drahá
- Softvérovo realizovaný zásobník je pomalší, ale má väčšiu kapacitu
- Kompromis: kombinovaný zásobník – vrchná časť registre, spodná časť – pamäť (obrázok)
- Výhoda kombinovaného zásobníka:
  - za sebou idúce operácie so zásobníkom využívajú údaje uložené v registrovej časti zásobníka (rýchlosť)
  - Pamäťová časť – veľká kapacita

8.2. Kombinovaná implementácia zásobníka (HP 3000 Stack)

- SB – dno
- SL – najväčšia možná adresa vrchu
- SM – vrch zásobníka v pamäti
- SR – počet použitých registrov z registrovej časti
- SP – stack pointer

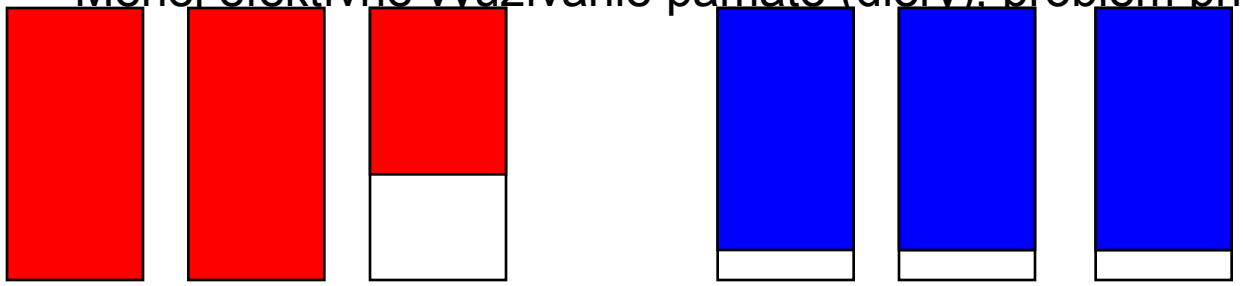


8.3. Modulárna pamäť

- Urýchlenie vykonávania programu
  - paralelné spracovanie inštrukcií
- Úzke miesto je pamäť
- Paralelné operácie s pamäťou
  - Pamäte s viacerými I/O
  - Modulárne organizovaná pamäť
- Modulárna pamäť:
  - Rozdelená do viacerých nezávislých modulov
  - Adresa sa skladá z dvoch častí:
    - Vyššie bity = adresa modulu
    - Nižšie bity = adresa v module

8.3. Modulárna pamäť

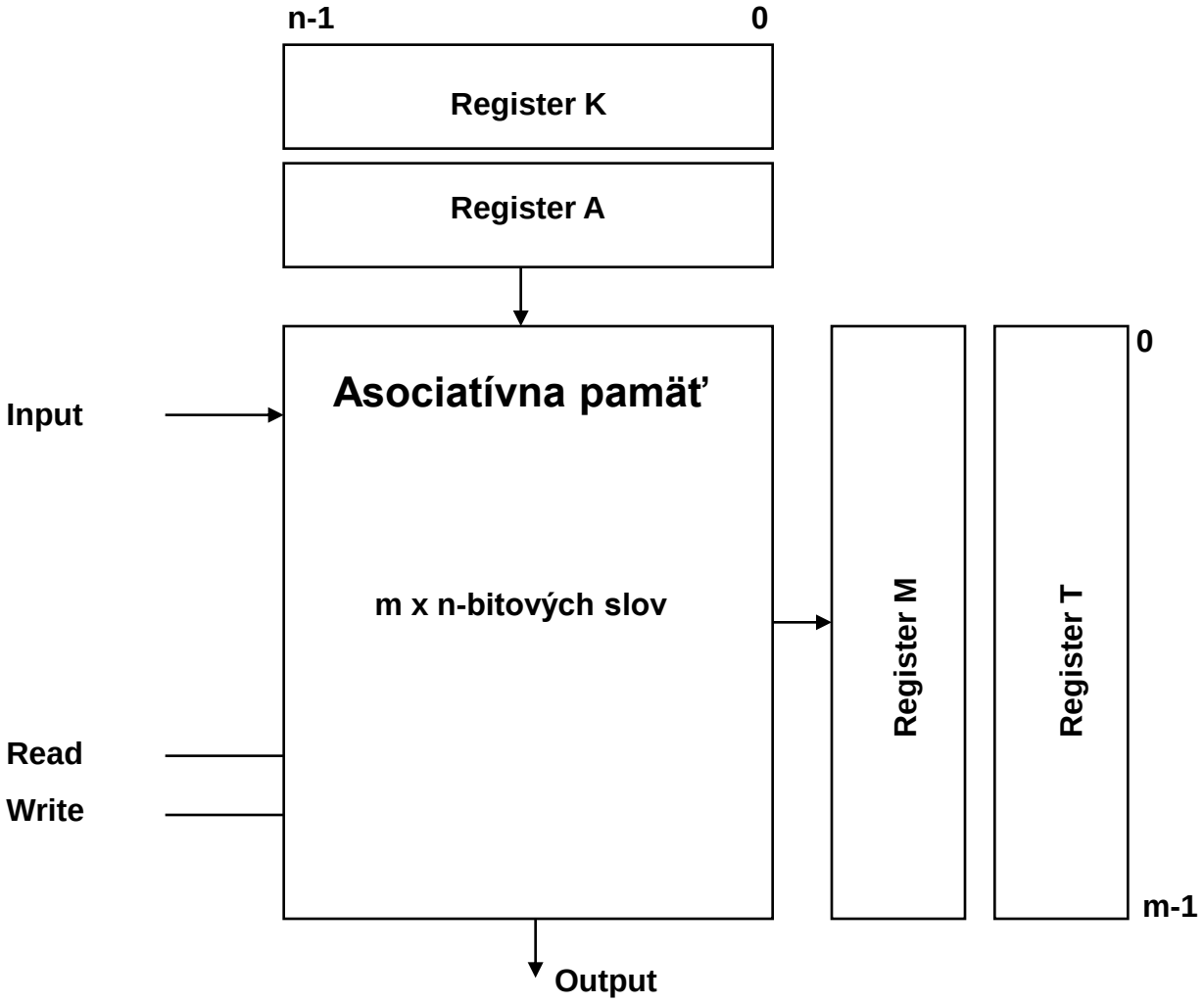
- Vyššie bity – modul, nižšie bity – adresa v module
  - „Susedné“ informácie sú uložené v tom istom module (zápis do stĺpca)
  - Pri spracovaní programu sa pracuje s jedným modulom
  - Ostatné môže využívať napr. zariadenia s DMA
  - Moduly sa využívajú postupne, ľahko dajú sa pridávať nové moduly
- Nižšie bity – modul, Vyššie bity – adresa v module
  - „Susedné“ informácie sú uložené v iných moduloch (zápis do riadka)
  - „Susedné“ informácie možno čítať súčasne (memory interleaving)
  - Menej efektívne využívanie pamäte (diery), problém pridať ďalší modul



8.4. Asociatívna pamäť (1)

- RAM – k údajom sa pristupuje na základe adresy pamäťového miesta, na ktorom sú uložené
- Asociatívna pamäť
  - Údaje sa porovnávajú so vzorkou
  - Indikuje sa, na ktorých adresách došlo k zhode so vzorkou
  - Testuje sa paralelne – pamäť je rýchla
  - Nevýhoda – zložitosť a cena
- Popis asociatívnej pamäte (obr.)
  - Tradičný vstup a výstup
  - Key register K (n-bitový)
  - Argument register A (n-bitový)
  - Match register M (m-bitový)
  - Tag register T (m-bitový)
  - Slová v pamäti sú n-bitové
  - Každé slovo má 1 bit v registri M

8.4. Asociatívna pamäť (2)



8.4. Asociatívna pamäť (3)

- Princíp činnosti:
  - Do registra A sa umiestni vzorka
  - Obsah A sa porovná so všetkými m slovami pamäte
  - Tam kde nastala zhoda sa do príslušného bitu registra M zapíše 1
    - Zhoda sa ale dá definovať aj inak
  - Pracuje sa (číta, zapisuje) s tými slovami asociatívnej pamäte, ktoré majú v registri M zapísanú jednotku
  - Porovnávacie obvody (je ich m) sa konštruujú pomocou XOR alebo DNF

  - Obyčajne sa neporovnáva celé slovo, ale len niektoré jeho polia
  - Maska (bity, ktoré chceme porovnávať) sa špecifikujú v registri K a hodnota týchto bitov v registri A

8.4 Asociatívna pamäť (4)

- Čítanie
  - Ak register M po porovnaní obsahuje viac jednotiek tak treba čítať obsah jednotlivých slov postupne
- Zápis
  - Problém s inicializáciou pamäte
  - Na začiatku je tag register T rovný 0
  - Postupne sa pri zápise do i-teho slova nastaví i-ty byt registra T na 1.

8.4 Asociatívna pamäť (5)

- Nech
  - x_i označuje i-ty bit registra M
  - t_i označuje i-ty bit registra T
  - w_{ij} označuje j-ty bit i-teho pamäťového miesta
  - k_j označuje j-ty bit registra K
  - a_j označuje j-ty bit registra A

- Potom

$$x_i = t_i \wedge \left(\bigwedge_j \left[(w_{ij} \Leftrightarrow a_j) \vee \neg k_j \right] \right)$$

8.5 Cache memory (1)

- Locality of reference principle
  - Programy a údaje sú zväčša organizované lokálne
  - Inštrukcie sa ukladajú za sebou
  - Cyklus je opakovanie istej postupnosti
  - Dáta sú uložené za sebou
- Rýchlosť vykonávania operácií závisí aj od rýchlosti ich načítania z pamäte
- Cache je malá rýchla pamäť
  - S operačnou pamäťou nekomunikuje priamo ale cez CPU
  - Obsahuje časť programov a údajov s ktorými práve pracuje CPU

8.5 Cache memory (2)

- Rýchlosť cache sa dosahuje:
  - Malá RAM
    - Čas prístupu je funkcia počtu slov v pamäti
  - Asociatívna pamäť
    - Čas prístupu je funkcia dĺžky slov v pamäti
  - Bipolárne pamäte namiesto MOS
  - Kombinácia predošlých možností
- Rýchlosť cache je rádovo vyššia ako rýchlosť operačnej pamäte