DCC007 – Organização de Computadores II

Aula 3 – Revisão de OC-I

Prof. Omar Paranaiba Vilela Neto

Instruções:

- **Linguagem de Máquina**
- **Mais *primitiva* que linguagens de alto nível**
  i.e., controle de fluxo não sofisticado
- **Muito restritiva**
  ex. MIPS Instruções Aritméticas

- **Vocês viram a arquitetura do conjunto de instruções do MIPS**
  - **similar a outras arquiteturas desenvolvidas após 1980's**
  - **Mais de 100 milhões de processadores MIPS fabricados em 2009**
  - **usado pela NEC, Nintendo, Silicon Graphics, Sony**

objetivos do projeto: maximizar o desempenho e minimizar custo e tempo de projeto

MIPS - Aritmética

- **Todas instruções tem 3 operandos**
- **A ordem dos operandos é fixa (destino primeiro)**

Exemplo:

Código C: `A = B + C`

Código MIPS: `add $s0, $s1, $s2`

(associação com variáveis pelo compilador)

Instruções

- **Instruções load e store**
- **Exemplo:**

Código C: `A[8] = h + A[8];`

Código MIPS:

```
lw $t0, 32($s3)
add $t0, $s2, $t0
sw $t0, 32($s3)
```

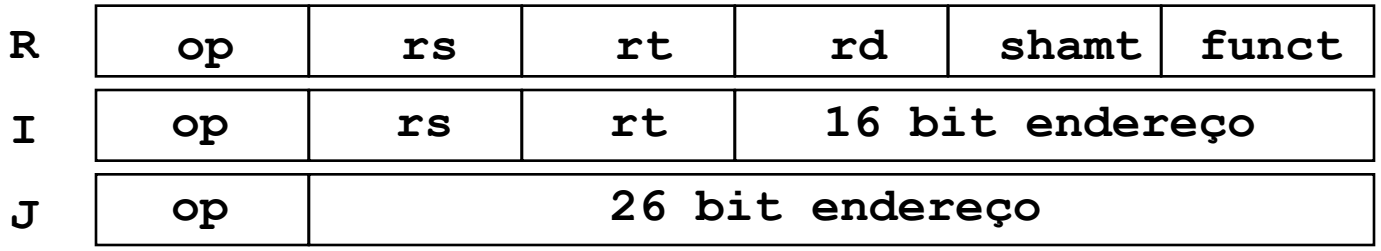
- **Store tem destino por último**
- **Relembre operandos aritméticos são registradores, não memória!**

Instruções:

- **Instrução** **Resultado**

```
add $s1,$s2,$s3   $s1 = $s2 + $s3
sub $s1,$s2,$s3   $s1 = $s2 – $s3
lw $s1,100($s2)   $s1 = Memória[$s2+100]
sw $s1,100($s2)   Memória[$s2+100] = $s1
bne $s4,$s5,L     próxima instr. é Label se $s4 ≠ $s5
beq $s4,$s5,L     próxima instr. é Label se $s4 = $s5
j Label           próxima instr. é Label
```

- **Formatos:**



Linguagem de Máquina

- Instruções – Código de máquina

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

Linguagem de Máquina

- Exemplo

```
   A[300] = h + A[300];
is compiled into
   lw   $t0,1200($t1) # Temporary reg $t0 gets A[300]
   add  $t0,$s2,$t0   # Temporary reg $t0 gets h + A[300]
   sw   $t0,1200($t1) # Stores h + A[300] back into A[300]
```

---

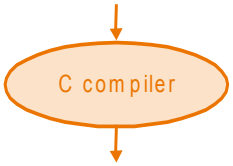
op	rs	rt	rd	address/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Linguagem de Máquina

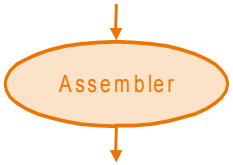
High-level language program (in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```



Assembly language program (for MIPS)

```
swap:
    muli $2, $5,4
    add  $2, $4,$2
    lw   $15, 0($2)
    lw   $16, 4($2)
    sw   $16, 0($2)
    sw   $15, 4($2)
    jr   $31
```



Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Agora vocês entendem!

Resumindo:

MIPS operands

Name	Example	Comments
32 registers	$0, $1, $2, ..., $30, $31	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $0 always equals 0. Register $1 is reserved for the assembler. Registers $26–$27 are reserved for the operating system.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $s1, $s2, $s3	$s1 = $s2 + $s3	Three operands; data in registers
	subtract	sub $s1, $s2, $s3	$s1 = $s2 − $s3	Three operands; data in registers
	add immediate	addi $s1, $s2, 100	$s1 = $s2 + 100	Used to add constants
Data transfer	load word	lw $s1, 100($s2)	$s1 = Memory[$s2 + 100]	Word from memory to register
	store word	sw $s1, 100($s2)	Memory[$s2 + 100] = $s1	Word from register to memory
	load byte	lb $s1, 100($s2)	$s1 = Memory[$s2 + 100]	Byte from memory to register
	store byte	sb $s1, 100($s2)	Memory[$s2 + 100] = $s1	Byte from register to memory
	load upper immediate	lui $s1, 100	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $s1, $s2, 25	if ($s1 == $s2) go to PC+4+100	Equal test; PC-relative branch
	branch on not equal	bne $s1, $s2, 25	if ($s1 != $s2) go to PC+4+100	Not equal test; PC-relative
	set on less than	slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compare less than; for beq, bne
	set less than immediate	slti $s1, $s2, 100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compare less than constant
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr $ra	go to $ra	For switch, procedure return
	jump and link	jal 2500	$ra = PC+4; go to 10000	For procedure call

Procedimento ou Função

Exemplo

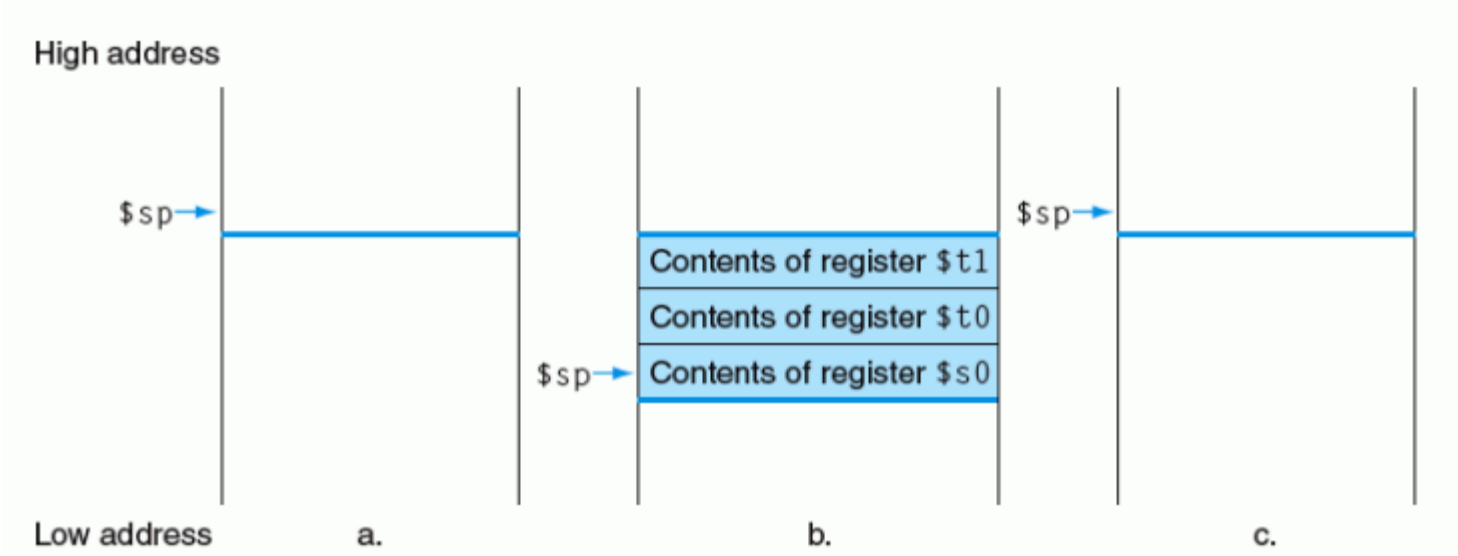
```
int folha (int g, int h, int i, int j)
{
      int f;

      f = (g+h) – (i+j);
      return f
}
```

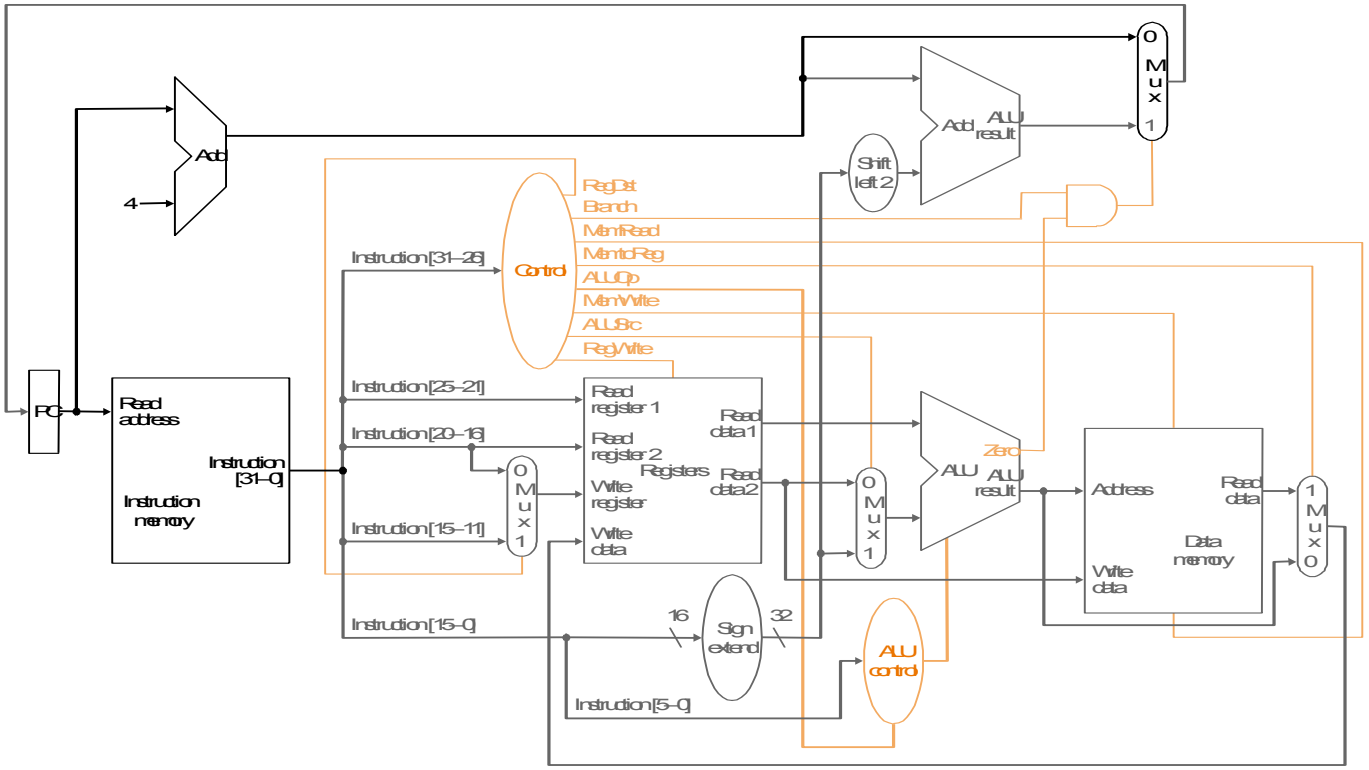
```
folha:   addi $sp, $sp, -12
         sw $t1, 8($sp)
         sw $t0, 4($sp)
         sw $s0, 0($sp)
         add $t0, $a0, $a1
         add $t1, $a2, $a3
         sub $s0, $t0, $t1
         add $v0, $s0, $zero
         lw $s0, 0($sp)
         lw $t0, 4($sp)
         lw $t1, 8($sp)
         addi $sp, $sp, 12
         jr $ra
```

Procedimento ou Função

Situação da pilha



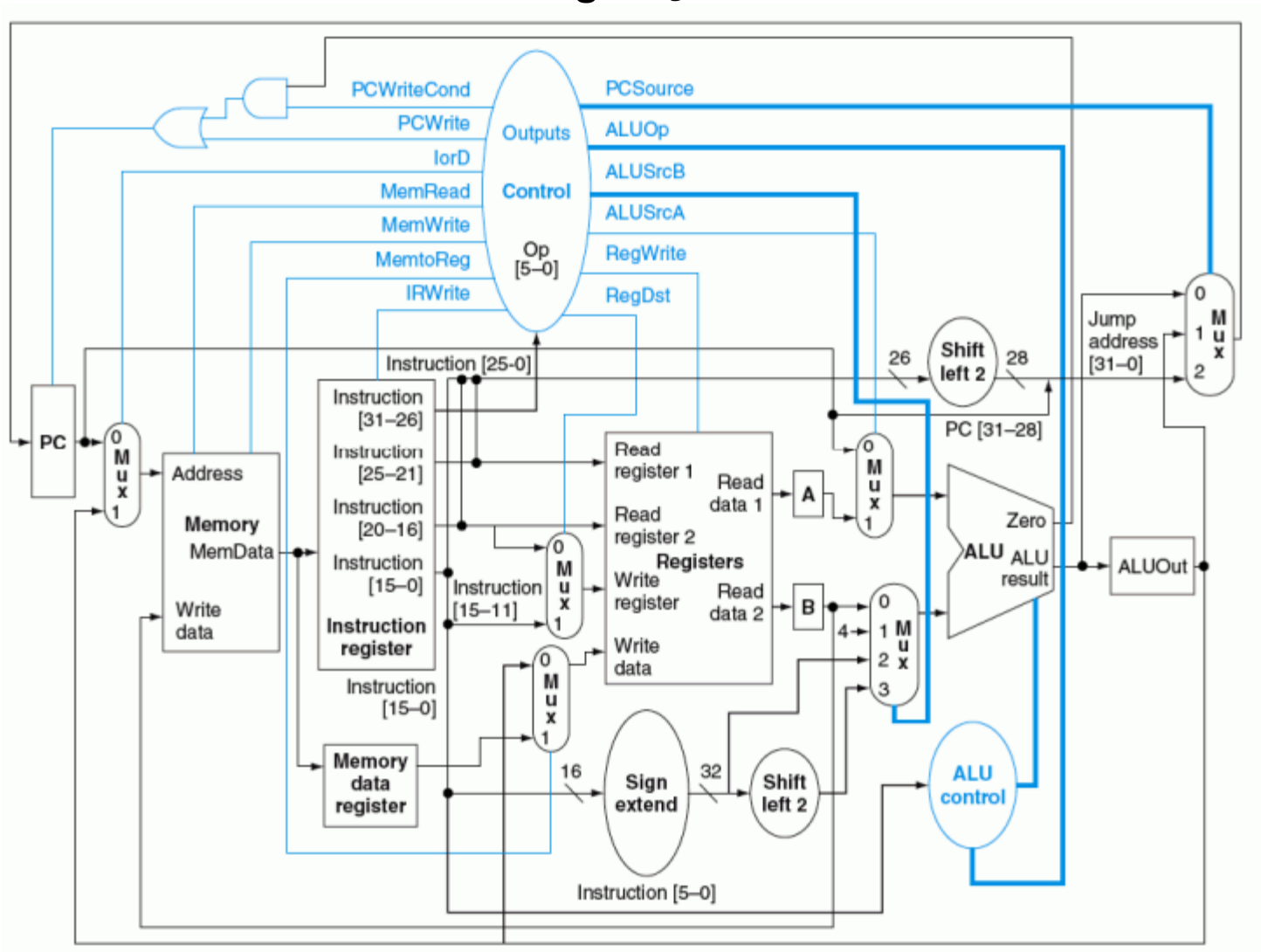
Ciclo Único



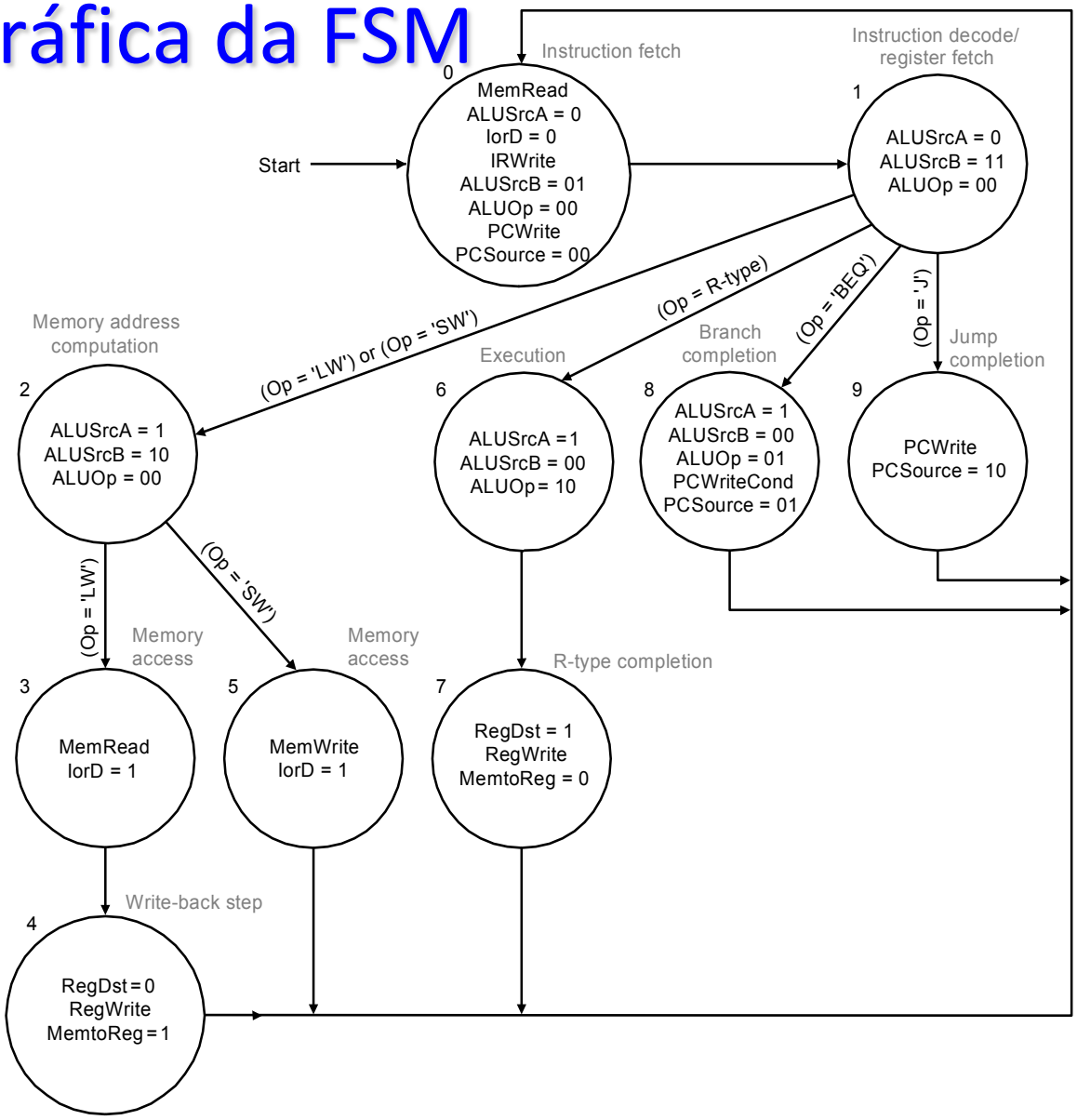
Instrução	RegDst	OrigALU	MemparaReg	EscreveReg	LeMem	EscreveMem	Branch	ALUOp1	ALUOp0
formato R	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

Abordagem Multiciclo

- **Precismos de uma nova configuração de controle**

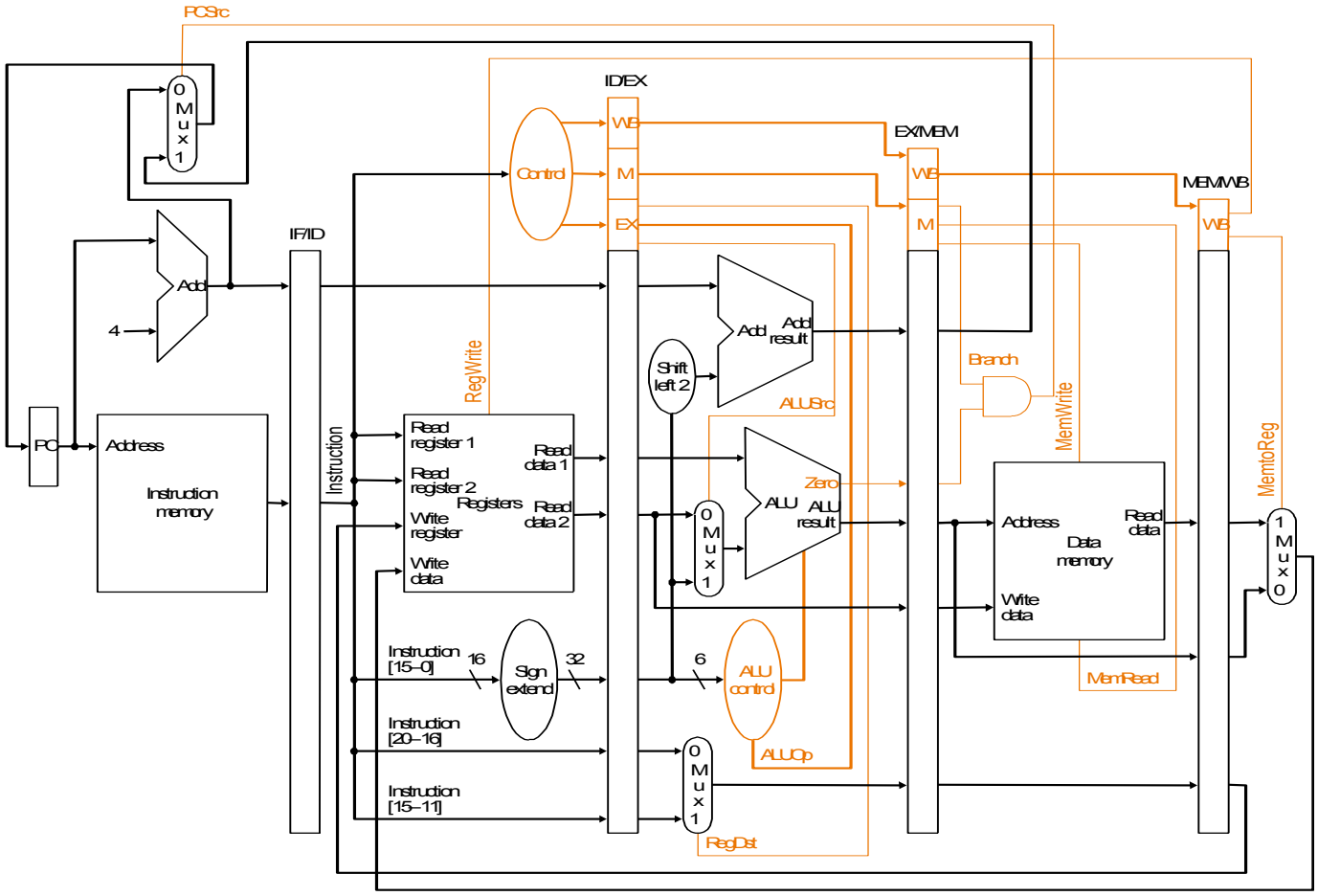


Especificação Gráfica da FSM



- **Quantos bits nós necessitamos para especificar os estados?**

Pipeline


PCSrc
0
M
u
x
1
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
Add
4
PC
Address
Instruction
memory
Instruction
RegWrite
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
Shift
left 2
Add
Add
result
ALUSrc
Zero
ALU
ALU
result
Branch
MemWrite
MemtoReg
Address
Data
memory
Read
data
Write
data
MemRead
Instruction
[15–0]
16
Sign
extend
32
6
ALU
control
ALUOp
Instruction
[20–16]
Instruction
[15–11]
RegDst
1
M
u
x
0

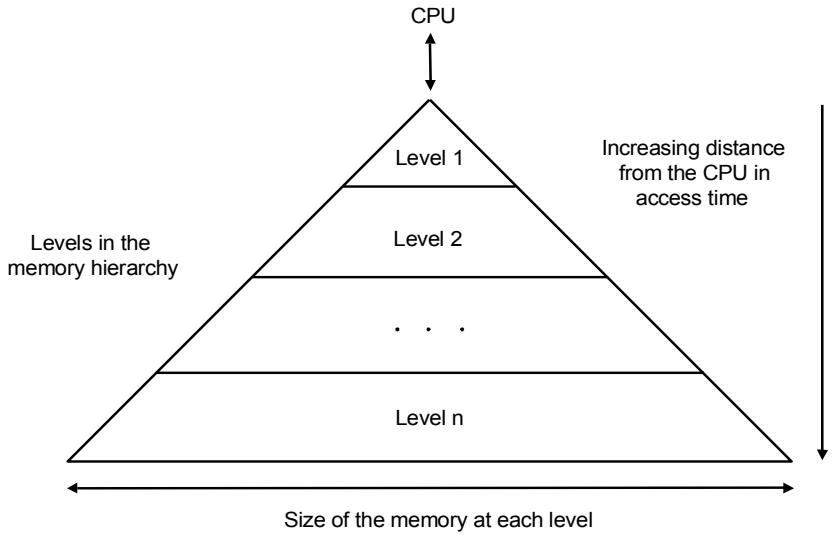
Explorando a Hierarquia de Memórias

- **Usuários desejam memórias rápidas e grande!**

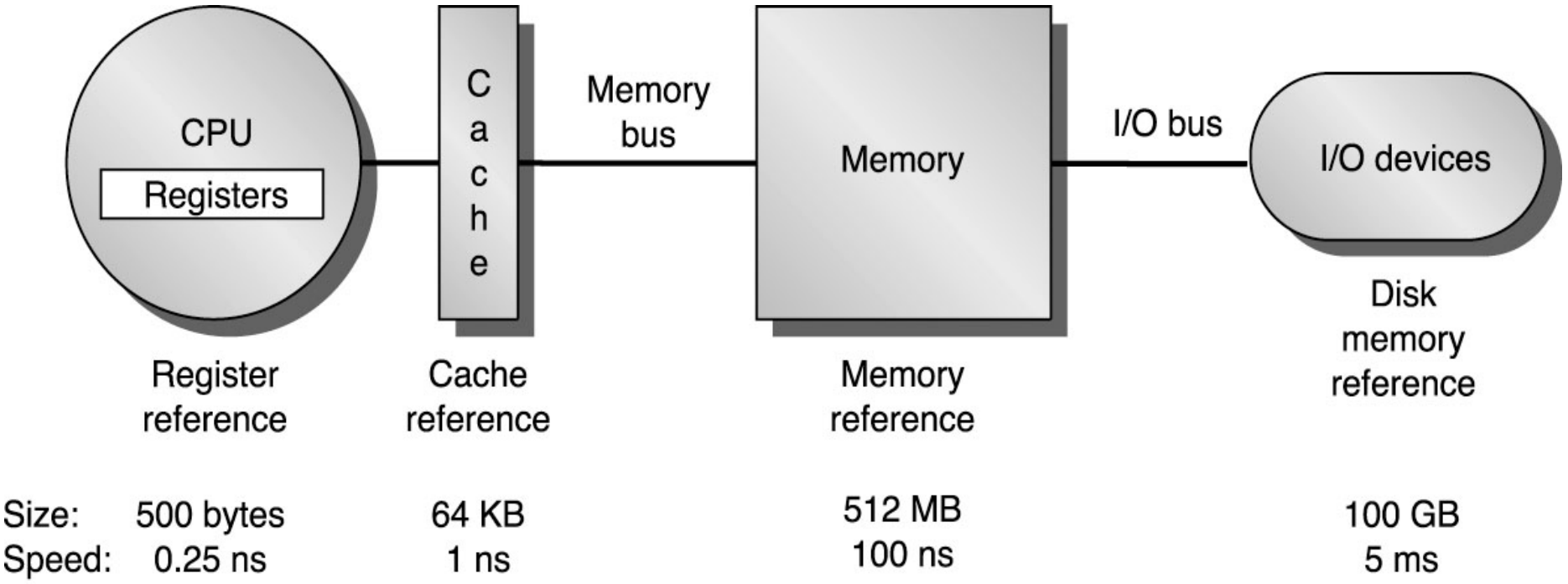
  SRAM tempo de acesso são 2 - 25ns e custam de $4000 a $10000 por GB.
  DRAM tempo de acesso são 60-120ns e custam de $100 to $200 por GB.
  Disco tempo de acesso 10 to 20 milhões ns e custam $.50 to $2 por GB.

  2004

- **Sugere a construção de uma hierarquia de memória**



Explorando a Hierarquia de Memórias



Localidade

- **Princípio que faz com que ter uma hierarquia de memória seja uma boa idéia**

- **Se um item é referenciado,**

  Localidade temporal : ele tende a ser referenciado de novo, logo

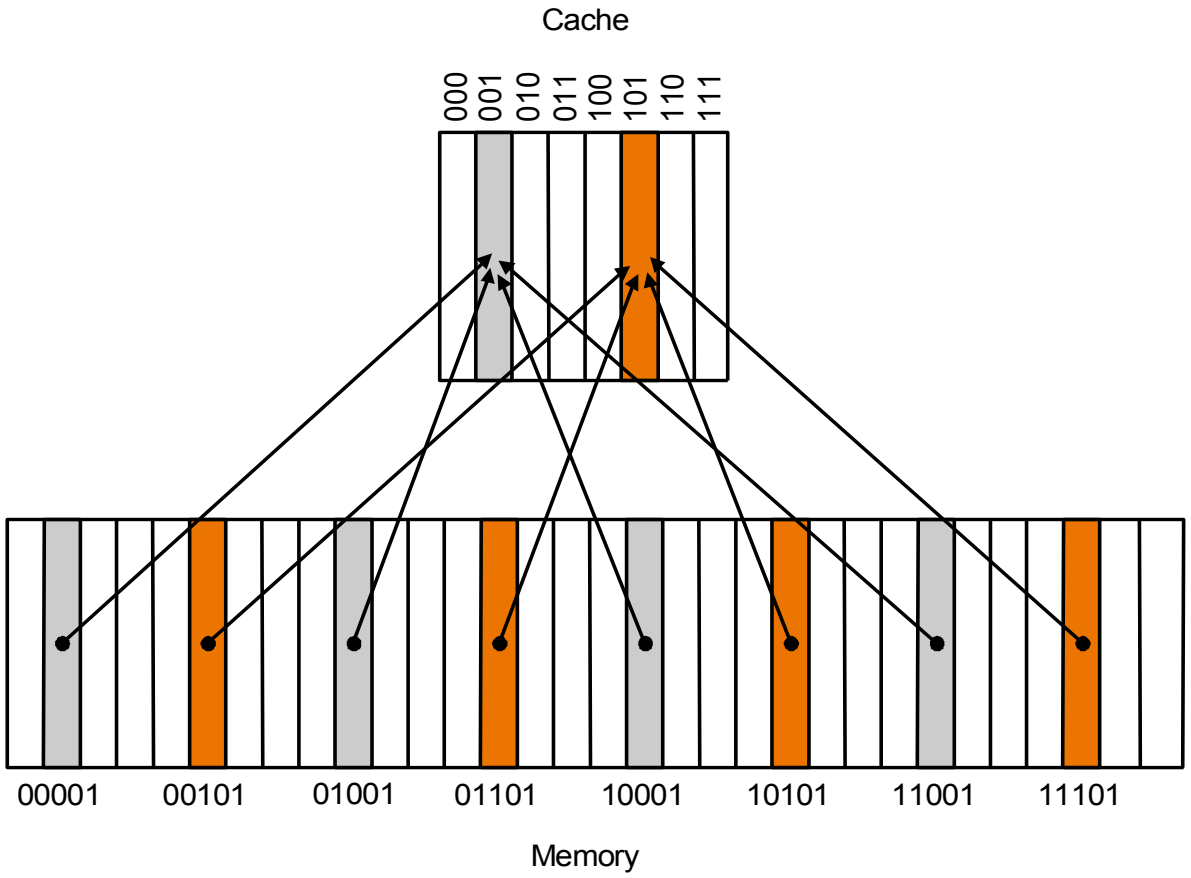
  Localidade espacial: itens próximos tendem a ser referenciados de novo, logo.

Porque um código tem localidade?

- **Nosso foco inicial: dois níveis (superior, inferior)**
  - **bloco: unidade mínima de dado**
  - **acerto: dado requisitado está no nível superior**
  - **falta: dado requisitado NÃO está no nível superior**

Cache Mapeado Diretamente

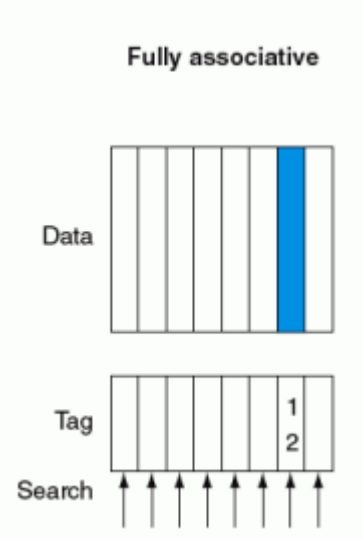
- **Mapeamento: o endereço é o módulo do número de blocos no cache**



Decrescendo a taxa de faltas usando associação

Chaches totalmente associativas

- Blocos podem ser colocados em **qualquer local** da cache;
- Buscar um bloco é **mais custoso** (vários testes necessários).

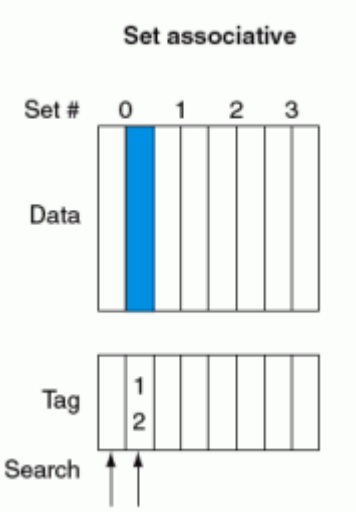


Decrescendo a taxa de faltas usando associação

Chaches associativas por conjunto

- Blocos podem ser colocados em qualquer local **dentro de um determinado conjunto**;

- Combina o Mapeamento direto com associatividade.

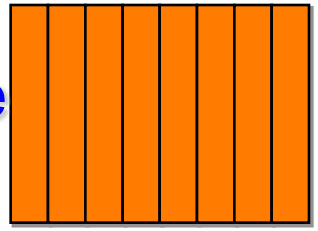


Resumo de Cache

Totalmente associativa

Block no. 0 1 2 3 4 5 6 7

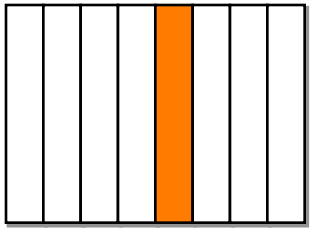
Cache



Mapeamento direto

(12 % 8) = 4

Block no. 0 1 2 3 4 5 6 7

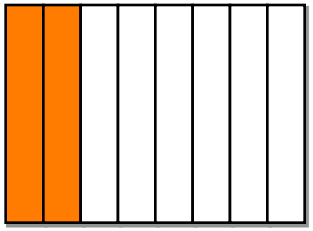


Parc. associativa

(12 % 4) = Set 0

Set no. 0 1 2 3

Block no. 0 1 2 3 4 5 6 7



Block no.

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

Memory