A bíblia sagrada de Organização de Computadores

Uma abordagem para um aluno desesperado

Novembro 2014

1 Licença

"THE BEER-WARE LICENSE":
Matheus Ben-Hur wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return.

exercícios retirados de:

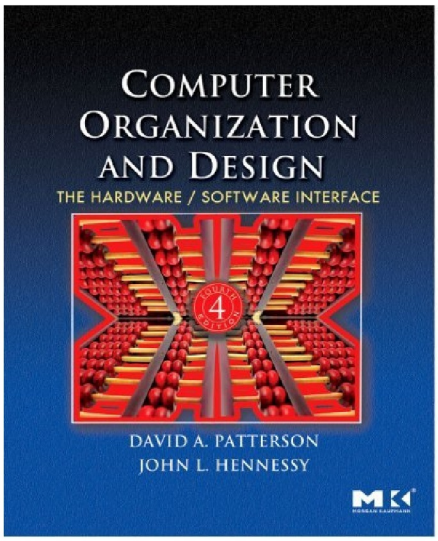


Figure 1: Computer Organization Software and Hardware Interface 4 ed.

EXERCÍCIOS – ORG

2.1.1 (a)

f = g + h + i + j;

f = $S4

g = $S0 h = $S1

i = $S2 j = $S3

```
add $t0 $S0, $S1
add $t1 $S2, $S3
add $S4 $t1, $t0
```

(b)

```
addi $t0 $S1, 5
add $S4 $t0, $S0
```

2.2.1

(a)

```
add $t0 $S4, $S2
add $S4 $S4, $t0
```

(b)

```
addi $t0 $S3, 2
add $S4 $t0, $S0
```

2.3.1

(a)

```
addi $t0 $S3, 2
add $t1 $S1, $S2
add $t2 $S4, $S0
add $S4 $t0, $t1
add $S4 $S4, $t2
```

(b)

```
addi $t0 $S4, 5
sub $S4 $S0, $t0
```

2.2.4 (a) f = f+b;

(b) f = 0 - f;
f = f + 1;

2.3.4
(a) f = -g + h;

(b) h = f+1;
f = g-h;

2.4.1 (a)

B = $S0	f = $S3
h = $S1	
g = $S2	

lw $t0 4($S0)
add $t0 $S1, $t0
add $S3 $t0, $S2

(b) não há uma resposta definida pois não sabemos o que vem de B[4]

2.4.4
(a)

a = $S0	c = $S2	e = $S4
b = $S1	d = $S3	

a = a+b;
a = a + c;
a = a+d;
a = d+e;

(b) $S6 = f

a = f[4];

2.5.4
(a) $(305419896)_{10}$

(b) $(3199070221)_{10}$

2.5.5 (a) big endian

0 7

1	2	3	4	5	6	7	8

little endian

0 7

8	7	6	5	4	3	2	1

(b) big endian

0 7

b	e	a	d	f	o	o	d

little endian

0 7

d	o	o	f	d	a	e	b

2.11.5

ⓐ add $at $v0, $v1

ⓑ sw $a1, 4($t3)

2.12.1
2.12.2

ⓐ

opcode	rs	rt	rd	Shmt	funct
5	3	3	3	5	6

25 bits

ⓑ

opc	rs	rt	im
5	5	5	10

25 bits

Considerações Ista

BIG ENDIAN

MSB LSB

AA	BB	CC	DD

regist.

0	AA
1	BB
2	CC
3	DD

LITTLE ENDIAN

MSB LSB

AA	BB	CC	DD

DD	0
CC	1
BB	2
AA	3

HEXADECIMAL PARA DECIMAL

$(1234)_{16} = (4660)_{10}$

$1 \cdot 16^3$

$4 \cdot 16^0$

$3 \cdot 16^1$

$2 \cdot 16^2$

$$\boxed{1 \cdot 16^3 + 2 \cdot 16^2 + 3 \cdot 16^1 + 4 \cdot 16^0}$$

2.15.2 ⓐ nor $t3, $zero, $t3
and $t1, $t2, $t3

ⓑ nor $t3, $zero, $t3
and $t1, $t2, $t3
nor $t1, $zero, $t1

2.15.5 ⓐ A = B & C[0]; $\boxed{A = \$s0,\ B = \$s1,\ C = \$s2}$
lw $t0, 0($s2)
and $s0, $s1, $t0

(b) $\boxed{A = \$s0 \ , \ B = \$s1 \ , \ C = \$s2}$

```
    beq $s0, $zero, else
    add $s0, $s1, $zero
    beq $zero, $zero, end
else: lw $s1, 0($s2)
end:
```

2.16.4

(a) $t2 = 2 (b) $t2 = 2

2.17.3

(a)
```
abs: sub $t2, $zero, $t3   # t2 = -t3
     ble $t3, $zero, done  # if t3<0. resut is t2
     add $t2, $t3, $zero   # if t3>0 result is t3
done:
```

(b) slt $t1, $t3, $t2

Considerações

- olhar exercício 2.15.2
- Instruções lógicas no mips é rep. somente por:
  - and
  - or
  - nor → nor something, $zero = $\overline{\text{something}}$

Com isso conseguimos implementar q/qr instrução!

- olhar e refazer ex: 2.17.3 (a)

2.19.1 (a)

```
compare: add: $sp, $sp, -4   #
         sw  $ra, 0($sp)     # Salvando o retorno na pilha
         add $s0, $a0, $zero # colocando os argumentos
         add $s1, $a1, $zero   nos saves
         jal sub
         addi $t0, $zero, 1   # poe 1 em $t0
         beq $v0, $zero, Exit # se v0 = 0 vai p/ a saída
         slt $t1, $zero, $v0  # se zero < $t1, $t1 = 1
         bne $t0, $t1, Exit
         add $t0, $zero, $zero # $t0 = zero.

Exit:    add $v0, $t0, $zero  # seta o valor de retorno
         lw $ra, 0($sp)       # restaura ra da pilha
         addi $sp, $sp, 4     # restaura a pilha
         jr $ra               # jump para ra

sub:     sub $v0 $a0, $a1
         jr ra
```

2.19.2

```
int compare (a,b){
  if( a-b >= 0) {
      return 1;
}else {
   return 0;
  }
```

```
sub $t0, $a0, $a1
slt $t1, $t0, $zero
addi $t0, $zero, 1
bne $t0, $t1, Exit
add $v0, $t1, $zero
jr $ra

Exit: add $v0, $zero, $zero
      jr $ra
```

//

2.19.4 (a)

```
f: addi $sp, $sp, -8
   sw $a2, 0($sp)      # salvando c
   sw $ra, 4($sp)      # salvando ra
   jal funct           # vai p/ funct
   add $a0, $zero, $v0 # poe o retorno como parâm
   lw $a2, 0($sp)      # restaura o c
   jal funct           # vai p/ funct de novo
   lw $ra, 4($sp)      # restaura o $ra
   addi $sp, $sp, 8    # arruma o stack pointer
   jr $ra              # volta p/ onde foi chamado
funct: something
```

Considerações Lista Procedures

Registradores

$a0 - $a3 → armazenamento de parâmetros, salvos pela rotina chamadora qdo necessário

$v0 - $v1 → valores de retorno

$ra → regist. do end. de retorno, salvo pela rotina chamadora!
jal $ra instr. q retorna.

$t0 - $t9 → temporários, salvos pela chamadora qdo necessário

$s0 - $s7 → "Saves" salvos pela chamada

$sp → stack pointer, deverá ser ajustado antes e depois de usar.

→ abre espaço na pilha
→ salva o q tem q salvar
→ faz os métodos do escopo
→ restaura os valores
→ ajusta a pilha
→ volta p/ sua chamadora

olhar todos os ex do capítulo!

USO SIMBÓLICO DO REGISTRADOR

se refere ao nome que utilizamos comumente $t0, $s0, $v0, $a0 ...
e **NÃO** ao seu número $1, $2, $9, $22

INSTR. IMPORT. sltu $t0, x, y seta no $t0 se $0 ≤ x < y

2.24.1 (a) $t0 = 00000012

em 0x1000 0010 terá = 00000012

(b) $t0 = 00000012

em 0x1000 0010 terá = ffffff12

2.25.1

(a) 1010 1101 0001 0000 | 0000 0000 0000 0010

AC10

lui $t1, AC10

ori $t1, 2 //

(b) mesmo código da a porém com as constantes do exercício

2.25.2 (a) no

(b) no } I need more instructions!

2.25.3

(a) no

(b) no

2.25.6 (a) $t0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

(b) $t0 | 1 | 2 | 3 | 4 | 0 | 0 | 0 | 0 |

Considerações

Trabalhando com constantes de 32 bits

- quebramos em duas instruções
  - lui e ori, nessa ordem, a lui zera os 16 bits menos significativos do registrador.

olhar ex: 2.25.1 e 2.25.6 //

Endereçamento

beq x, y Label → multiplicado por 4 e somado à PC+4

j Label → mesma história

Olhar Relatório (2) sobre Jump Address table

- basicamente é uma tabela onde os endereços contém outros endereços.

2.34.1

a) addi $s0, $zero, 10

Loop: add $s0, $s1, $s0

addi $s0, $s0, -1

beq $s0, $zero, loop

2.35.2 a) lw $s0, 0($s1)

2.36.1

a) sll $s1, $s1, 3

add $s3, $s2, $s1

2.37.2

d) sll $s2, $s2, 2

add $s4, $s4, $s2

lw $s3, 0($s4)

2.37.4

b) lui $d0, 0x1234

ori $d0, 0x5678

Considerações

- não há considerações :)
- ver slide sobre arquitetura ARM e x86

2.8.1 a)

```
 70000000
 7FFFFFFF
 --------
 EFFFFFFF
```

A B C D E F
10 11 12 13 14 15

não há overflow

b)

```
 70000000
 70000000
 --------
```

há overflow!

2.9.2 ⓐ não há

ⓑ há

2.9.4

ⓐ AD 100002
70000000 sim

ⓑ Tbm

2.28.2

The SC instruction referenced by register R6, not complete sucessfully, because R3 was complete.

2.28.3

```
try:  MOV  R3, R4
      MOV  R6, R7
      LL   R2, 0(R2)
      SC   R3, 0(R2)
      beqz R3, try
try2: LL   R6, 0(R1)
      SC   R6, 0(R1)
      beqz R6, try2.
      MOV  R4, R2
      MOV  R7, R5
```

2.28.4 ⓐ

Proc1	PROC2	Cycle	Proc1 $t1	Proc1 $t0	MEM ($s1)	PROC2 $t1	PROC2 $t0
		0	1	2	99	30	40
		1	99	2	99	99	40
ll $t1, 0($s1)	ll $t1, 0($s1)	2	99	1	2	99	40
sc $t0, 0($s1)		3	99	1	2	99	0
	sc $t0, 0($s1)						

1.3.1 $\text{time}_{CPU} = \underbrace{I}_{\text{instructions}} \times \overbrace{CPI}^{\text{Average}} \times \underbrace{T}_{\text{period}} = \frac{I \times CPI}{f} = \frac{\text{unidade}}{\text{segundos}}$

$\text{time}_{P1} = \frac{1,5}{2GHz} = 0,75 \cdot 10^{-9} = 7,5 \cdot 10^{-10}$

$\text{time}_{P2} = \frac{1,0}{1,5GHz} = 0,66 \cdot 10^{-9} = 6,66 \cdot 10^{-10}$

$\text{time}_{P3} = \frac{2.5}{3GHz} = 0.833 \cdot 10^{-9} = 8,33 \cdot 10^{-10}$

os tres executam o mesmo nº de Instruções, por isso podemos desconsiderar o I na conta.

P2 tem a maior performance

1.3.2 $\boxed{\text{tempo}_{CPU} = \text{ciclos}_{CPU} \times T = \frac{\text{ciclos}_{CPU}}{f}}$ $\boxed{\text{ciclos} = I \times CPI}$

P1 $10 = \frac{\text{ciclos}_{P1}}{2GHz}$ $10 \cdot 2GHz = 20 \cdot 10^9$ ciclos

$2 \cdot 10^9 = I \cdot 1,5 = I = \frac{2}{1,5} \cdot 10^9 = 1,\bar{3} \cdot 10^9$ Instrucoes

P2 $10 = \frac{\text{ciclos}_{P2}}{1,5GHz} = 15 \cdot 10^9$ ciclos

$15 \cdot 10^9 = I \cdot 1 = 15 \cdot 10^9$ Instrucoes

P3 $10 = \frac{\text{ciclos}_{P3}}{3GHz} = 30 \cdot 10^9$ ciclos

$30 \cdot 10^9 = I \cdot 2.5 = 12 \cdot 10^9$ Instrucoes

1.3.4

P1 $\boxed{IPC = \frac{I}{ciclos} = \frac{1}{CPI}}$

$\boxed{\text{tempo CPU} = I \times CPI \cdot f}$

$7 = 20\ 10^9 \times CPI \cdot 0,5$

$\boxed{CPI = 0,7}$ $IPC = \frac{1}{0,7} = 1,42$

P2

$10 = 30 \times 10^9 \times CPI \cdot 0,6$

$10 = 20 \cdot 10^9 \times CPI$

$CPI = 0.5$ $IPC = \frac{1}{0.5} = 2$

P3

$9 = 90 \cdot 10^9 \cdot CPI \cdot 0,3$

$9 = 27 \cdot 10^9\ CPI$

$0.333\ CPI$ $IPC = \frac{1}{0,333} = 3.3003$

1.4.1

$Time = I \cdot CPI \cdot T = \frac{I \cdot CPI}{clock\ rate}$

P1 $= \frac{10^6 \cdot (0,1 \cdot 1 + 0,2 \cdot 2 + 0,5 \cdot 3 + 0,2 \cdot 4)}{1,5 \cdot 10^9} = \frac{2.8 \cdot 10^6}{1.5 \cdot 10^9} = 1,86\ 10^{-3}$

P2 $= \frac{10^6 (0,1 \cdot 2 + 0,2 \cdot 2 + 0,5 \cdot 2 + 0,2 \cdot 2)}{2 \cdot 10^9} = \frac{2 \cdot 10^6}{2 \cdot 10^9} = 1\ 10^{-3}$

P2 é mais rápido

1.4.2

P1 = CPI global = 2.8

P2 = CPI global = 2

1.5.1

(a) $\frac{\text{clock rate}}{\text{CPI}} = \frac{\frac{\text{ciclos}}{\text{seg}}}{\frac{\text{ciclos}}{\text{Instr.}}} = \boxed{\frac{\text{Instr.}}{\text{seg}}}$

global CPI P1 ⇒ $\frac{1+2+3+4+3}{5} = \frac{13}{5} = 3$

$\boxed{P1}$ → $\frac{1 \cdot 10^9}{3} = 0.333 \cdot 10^9$ I/s

$\boxed{P2}$ → global CPI $= \frac{2+2+2+4+4}{5} = \frac{14}{5} = 3$

$\frac{1.5 \cdot 10^9}{3} = 0.5 \cdot 10^9$ I/s

1.6.1 (a) $\boxed{\text{Cycle} = 1\text{ns}}$ = clock rate $= \frac{1}{1\text{ns}} = 1$ GHz

$1 = \frac{I \cdot CPI}{f} = 1 = \frac{10^9 \cdot CPI}{10^9} = \boxed{CPI = 1}$ → compiler (A)

$1.4 = \frac{1.2 \cdot 10^9 \cdot CPI}{10^9} = CPI = \frac{1.4}{1.2} = \boxed{1.1\overline{6}}$ → compiler (B)

1.7.2

Pentium Pro $\longrightarrow$ Willamete

1.7.3

$$\frac{2.667 \cdot 10^3}{12.5 \cdot 10^6} = \boxed{2 \cdot 1 \cdot 10^2} \Rightarrow$$

$$\frac{95}{3.3} = \boxed{28.78}$$

1.7.4 $\boxed{C = P/V^2}$ // $\rightarrow$ desconsiderando o chaveamento

80286 $\rightarrow$ $C = 3.3/25$

80326 $\rightarrow$ $C = 4.1/25$

80486 $\rightarrow$ $C = 4.9/25$

⋮

1.8.1

$C = P/(V^2 \cdot clock) \Rightarrow$ farads

$\boxed{V1} \rightarrow C = 1/(25 \cdot 0{,}5 \cdot 10^9) = 8 \cdot 10^{-11}$ //

$\boxed{V2} \rightarrow C = 0{,}9/(10{,}89 \cdot 1 \cdot 10^9) = 8.264628 \cdot 10^{-11}$ //

Dimin. em $2.664628 \cdot 10^{-12}$ farads

1.8.2 $\boxed{V1} \rightarrow 8 \cdot 10^{-11}$

$\boxed{V2} \rightarrow 8 \cdot 10^{-11} = P/(1.089 \cdot 10^{10}) \Rightarrow 0{,}8712$ //

$1 - 0{,}8712 = 0{,}1288 \Rightarrow \boxed{12{,}88\%}$ //

1.9.3

(a) $1/49 = 0{,}020408$

1.10.2

(a) $\boxed{2\ GHz}$

(1) Total of instructions → 4096

$$\text{global CPI} = \frac{1 \cdot 2560 + 4 \cdot 1280 + 2 \cdot 256}{4096} = \frac{8192}{4096} = 2$$

$$\text{Time} = \frac{4096 \cdot 2}{2 \cdot 10^9} = \frac{8192}{2 \cdot 10^9} = 4{,}096 \cdot 10^{-9} \rightarrow \boxed{4{,}096 \cdot 10^{12}}$$

$\boxed{4{,}096\ \mu seg}$

(2) $I = 2048$

$$\text{global CPI} = \frac{1280 + 4 \cdot 640 + 2 \cdot 128}{2048} = 2$$

$$\text{Time} = \frac{2048 \cdot 2}{2 \cdot 10^9} = \frac{4096}{2 \cdot 10^9} = \boxed{2{,}048\ \mu seg}$$

(4) $I = 1024$

$$\text{global CPI} = \frac{640 + 320 \cdot 4 + 2 \cdot 64}{1024} = 2$$

$$\text{Time} = \frac{1024 \cdot 2}{2 \cdot 10^9} = \boxed{1{,}024\ \mu s}$$

(8) $I = 512$

$$\text{global CPI} = \frac{320 + 4 \cdot 160 + 2 \cdot 32}{512} = 2$$

$$\text{Time} = 0{,}512\ \mu s$$

1.10.4 (2) $\boxed{3\ GHz}$

(1) $Time = \frac{10^{10} \cdot 1.2}{3 \cdot 10^9} = 4\ s$

(2) $Time = \frac{5 \cdot 10^9 \cdot 1.3}{3 \cdot 10^9} = 2.16\ s$

(4) $Time = \frac{2,5 \cdot 10^9 \cdot 1.5}{3 \cdot 10^9} = 1.25\ s$

(8) $Time = \frac{1.5 \cdot 10^9 \cdot 1.8}{3 \cdot 10^9} = 0,75\ s$

1.10.5

(2) $Power = \frac{5 \cdot mA}{MHz} voltage^2$

$voltage = \frac{1}{5} freq + 0,4$

$3\ GHz \rightarrow V = \frac{1}{5} \cdot 3 + 0,4 = 1V$

$300\ MHz \rightarrow V = \frac{1}{5}\ 0,5 + 0,4 = 0,5\ V$

(1) $\rightarrow P = \frac{5}{3000} \cdot 1^2 = 0,001666 \parallel 0,001666 \times 0,25$

(2) $0,003333 \parallel 0,0008333$

(4) $0,00666 \parallel 0,001666$

(8) $0,01333 \parallel 0,00333$

1.12.1

cycle Time = 0.333 ns

Clock rate = $\frac{1}{0.333}$ = 3 GHz

(a) $500 = \frac{2118 \cdot 10^9 \cdot CPI}{3 \cdot 10^9}$ => $CPI = \frac{500}{706} = 0,7$

(b) $1200 = \frac{336 \cdot 10^9 \, CPI}{3 \cdot 10^9}$ => $\frac{1200}{112} = 10,71$

1.12.2

$$\boxed{\text{SPEC RATIO} = \frac{T_{reference}}{T_{execution}}}$$

(a) $\frac{9770}{500} = 19,54$

(b) $\frac{9120}{1200} = 7,6$

1.12.3

Geometric Mean = $\sqrt{x \cdot y}$ ou $\sqrt[3]{x \cdot y \cdot z}$

$Gm = \sqrt{19.54 \cdot 7.6} = 12.18$

1.14.1

$CPI = \frac{ciclos}{Instrução}$ clkrate = $\frac{ciclos}{segundo}$

$\frac{Instrução}{segundo} = \frac{clkrate}{CPI}$ => $\frac{\frac{ciclos}{seg}}{\frac{ciclos}{Inst}}$ => $\frac{Inst}{seg}$

(P1) = $\frac{4 \cdot 10^9}{1,2 s} = 3.2 \cdot 10^9$ I/s

(P2) = $\frac{3 \cdot 10^9}{0.7 s} = 4 \cdot 10^9$ I/s

P2 é mais rápido

1.14.2 $\text{Time} = \frac{10^6 \cdot 1.25}{4 \cdot 10^9} = 0.3125 \cdot 10^{-3}$

P2 $\Rightarrow 0.3125\, 10^{-3} = \frac{I \cdot 0,75}{3 \cdot 10^9} = 0,9375 \cdot 10^6 = I \cdot 0,75$

$I = 1.25 \cdot 10^6$ Instructions

1.14.3

$\boxed{P1} = 4 \cdot 10^9 \cdot 10^{-6} / 1.25 = 3200$

$\boxed{P2} = 3 \cdot 10^9 \cdot 10^{-6} / 0.75 = 4000$

$\boxed{\text{Perf } P1 < \text{Perf } P2}$

Considerações

Performance e Benchmarks são duas bostas!

$$\text{Time} = I \times CPI \times T = \frac{I \times CPI}{f}$$

$$\text{Ciclos} = I \times CPI$$

$$\frac{\text{Instruções}}{\text{segundo}} = \frac{\text{clock rate}}{CPI}$$

$$CPI = \sum_{i=1}^{n} \left(\frac{I_i}{I} \cdot CPI_i \right)$$

$$P_{din} = C \times V_{DD}^2 \times f_{0 \to 1}$$

$$IPC = \frac{1}{CPI}$$

$$\text{SPECRATIO} = \frac{T_{reference}}{T_{execution}}$$

Exercícios Importantes
- 1.3.1, 1.3.2,
- 1.10.2, 1.10.4

- O maior clock rate nem sempre tem a maior performance

(b)

Proc1	Proc2	Cycle	Proc1 $s4	Proc1 $t1	Proc1 $t0	Mem ($t1)	Proc2 $s4	Proc2 $t1	Proc2 $t0
		0	2	3	4	99	10	20	30
	try: add $t0, $0, $s4	1	2	3	4	99	10	20	10
try: add $t0, $0, $s4	ll $t1, 0($s1)	2	2	3	2	99	10	99	10
ll $t1, 0($s1)		3	2	99	2	99	10	99	10
sc $t0 0($s1)		4	2	99	1	2	10	99	10
beqz $t0, try	sc $t0, 0($s1)	5	2	99	1	2	10	99	0
add $s4, $0, $t1	beqz $t0, try	6	99	99	1	2	10	99	0

Considerações

- cuidar com overflow aritmetico
- conversão hex to dec

↳ ativa uma flag

Load linker e Store conditional

dois loads podem ser feito simultaneamente, no mesmo reg, mas se isso for feito, quando dermos um sc, ao dar o próx irá haver falha, pois o conteúdo pego foi modificado.

um sc bem sucedido retorna com 1 no Reg fonte

um sc mal sucedido retorna com 0 no Reg fonte

2.30.1

(a)
```
#la $s0,v
lui $at , end de V
ori $s6, $at, 0
```

(b)
```
# blt $a0, $v0 , loop
slt $a0, $a0, $v0
addi $t0, $zero, 1
beq $a0, $t0, loop
```

2.21.1

(a) $sp 0x7fff fffC

$sp 0x7fff fff8 → $ra → dados do main

$sp 0x7fff fff4 → $ra de leaf_function

(b) $sp 0x7fff fffC

$sp 0x7fff fff8 → $ra → dados do main

$sp 0x7fff fff4 → $ra → dados de leaf_function

$gp 0x10008000 → 100 , my_global

2.31.1

(a) 0x0040 0000 lw $a0, 8000($gp)

0x0040 0004 jal 0x140 → lembre-se q é pc+4!

⋮

0x0040 0140

0x0040 0144 sw $a1, 8004($gp)

0x0040 0148 jal 0x40000000

⋮

0x0040 0348 END of text

Dta 0x1000 0000 x

⋮

0x1000 0040

0x1000 0044 y

Text

(b) 0x0040 0000 lui $at, 0x1000 0000

0x0040 0004 ori $a0, $at, 0x1000 0000

0x0040 0008 jal 0x0040 0140

⋮

0x0040 0140

0x0040 0144 sw $a0, 8044($gp)

0x0040 0148 j 0x004002C8

⋮

0x004002C8 jr $ra

---

data 0x1000 0000 X

⋮

0x1000 0040

0x1000 0044 Y

Considerações

Linker

- prestar atenção no tamanho do texto e dos dados de cada procedur
- lembrar que operações envolvendo o PC, sempre usam pc + 4
- lembrar que $gp começa em 10008000.

ARITH pd/d e $gp

10008000 → end do gp

− 10000044 → end do dado que queremos

7FBC → fazer o comp. de 2

8043 + 1

8044 → end. para colocar no lw ou sw!

divisão e multiplicação

Hi = 32 MSBS
Lo = 32 LSBS] Multiplicação

Hi = resto da divisão
Lo = quociente inteiro] divisão

para pegar do hi e lo usamos..

mfhi $s1

mflo $s0

4.12.1

(a) pipelined = cycle = 500ps

nonpipelined = cycle = 1650 ps

4.12.2

(a) pipelined = 500 x 5 = 2500 ps

nonpipelined = 1650 ps

4.12.3

(a) O melhor estágio para dividir será o que tem mais latência, neste caso MEM, pois isso irá diminuir nosso ciclo que era de 500 ps e agora é 400ps

4.13.2

(a) lw $1, 40 ($6) IF ID EX MEM WB

add $6, $2, $2 IF ID EX MEM WB

sw $6, 50 ($1) IF ID EX MEM WB

lw $1, 40 ($6) IF ID EX MEM WB

add $6, $2, $2 IF ID EX MEM WB

nop IF ID EX MEM WB

sw $6, 50 ($1 IF ID EX MEM WB

(b)

```
lw $5, -16($5)        IF ID EX MEM WB
sw $5, -16($5)           IF ID EX MEM WB
add $5, $5, $5              IF ID EX MEM WB
```

```
lw $5, -16($5)        IF ID EX MEM WB
nop                      IF ID EX MEM WB
nop                         IF ID EX MEM WB
sw $5, -16($5)              IF ID EX MEM WB
add $5, $5, $5
```

4.13.3

(a)

```
lw $1, 40($6)         IF ID EX MEM WB
add $6, $2, $2           IF ID EX MEM WB
sw $6, 50($1)               IF ID EX MEM WB
```

(b)

```
lw $5, -16($5)        IF ID EX MEM WB
sw $5, -16($5)           IF ID EX MEM WB
add $5, $5, $5              IF ID EX MEM WB
```

```
lw $5, -16($5)        IF ID EX MEM WB
nop                      IF ID EX MEM WB
sw $5, -16($5)              IF ID EX MEM WB
add $5, $5, $5
```

4.13.4

(a) whithout → 8 × 300 = 2400 ps

full → 7 × 400 = 2800 ps

(b) without → 9 × 200 ps = 1800 ps

full → 8 × 250 ps = 2000 ps

não tivemos ganho algum com o full-foward, pelo contrário, houve perda de desempenho

4.13.5 não é possível usar ALU-ALU forward nos dois casos

4.13.6

(a) ALU-ALU → 8 × 360 ⇒ 2880 ps

(b) ALU-ALU → 9 × 220 → 1980 ps

continuamos sem ganho :(

4.14.1 MEMÓRIA UNICA

(a)

```
      lw $1, 40($6)        IF ID EX MEM WB
      beq $2, $0 Label        IF ID EX MEM WB
      sw $6, 50($2)              IF ID EX MEM WB
Label: add $2, $3, $4               IF ID EX MEM WB
      sw $3, 50($4)
```

```
      lw $1, 40($6)        IF ID EX MEM WB
      beq $2, $0 Label        IF ID EX MEM WB
      sw $6, 50($2)              IF ID EX MEM WB
Label: add $2, $3, $4               IF ID EX MEM WB
      parada
      sw $3, 50($4)
```

9 ciclos

(b)

```
lw $5, -16 ($5)        IF ID EX MEM WB
sw $4, -16 ($4)           IF ID EX MEM WB
lw $3, -20 ($4)              IF ID EX MEM WB
beq $2, $0, Label               IF ID EX MEM WB
add $5, $1, $4                     IF ID EX MEM WB
```

```
lw $5, -16 ($5)        IF ID EX MEM WB
sw $4, -16 ($4)           IF ID EX MEM WB
lw $3, -20 ($4)              IF ID EX MEM WB
Parada                          IF ID EX MEM
Parada                             IF ID EX
parada
beq $2, $0, Label
add $5, $1, $4
```

12 ciclos

lembrete: nop's não podem ser adicionados pois eles são instruções e como tais, devem ser buscados da memória. Então para resolver este tipo de hazard, o pipeline deve ficar parado, sem buscar instrução nova.

4.14.2 EX MEM → um bloco só!

(a)

```
IF ID EM WB
   IF ID EM WB
      IF ID EM WB
         IF ID EM WB
```

7 ciclos $9/7 = 1.2857$ x mais rapido

(b)

```
IF ID EM WB
   IF ID EM WB
      IF ID EM WB
         IF ID EM WB
            IF ID EM WB
```

$12/8 = 1.5$ x mais rapido

8 ciclos

4.14.3

RESOLUÇÃO ID

(a) IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

(9) ciclos

RESOLUÇÃO EX

IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

(10) ciclos

$10/9 = 1,1$

RESOLUÇÃO ID

(b) IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

(10) ciclos

RESOLUÇÃO EX

IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

(11) ciclos

$11/10 = 1.1$

4.14.4 (a) $8 \times 130 = 1040$ ps
$7 \times 150 = 1050$ ps
$\div = 0.99$

(b) $9 \times 220 = 1980$ ps
$8 \times 240 = 1920$ ps
$\div = 1.03$

4.14.5

(a) $9 \times 180 = 1620$
$10 \times 130 = 1300$ $\Rightarrow \frac{1300}{1620} = 0.8$

(b) $10 \times 220 = 2200$ ps
$11 \times 220 = 2420$ ps $\Rightarrow \frac{2420}{2200} = 1.1$

4.14.6

(a) ao fazermos esta mudança, obtemos mais um
(b) ciclo de parada quando encontramos um branch, como o caminho crítico em ambos os casos é regido pela MEM, diminuir o tempo de EX não afetará o tempo do ciclo mas o facto de termos mais um stall aumentará o tempo de execução.

4.16.4

(a)

2ª Iteração

IF	ID	EX	MEM	WB										
	IF	ID	EX	MEM	WB									
		IF	ID	EX	MEM	WB								
			IF	ID	EX	MEM	WB							
				IF	ID	EX	MEM	WB						
					IF	ID	EX	MEM	WB					
						IF	ID	EX	MEM	WB				
							IF	ID	EX	MEM	WB			
								IF	ID	EX	MEM	WB		
									IF	ID	EX	MEM	WB	

a 2ª iteração ainda está na pipeline

3ª Iteração

(b)

WB				
MEM	WB			
EX	MEM	WB		
ID	EX	MEM	WB	
IF	ID	EX	MEM	WB
	IF	ID	EX	MEM
		IF	ID	EX
			IF	ID
				IF

4.16.5

(a) $1/5 = 20\%$

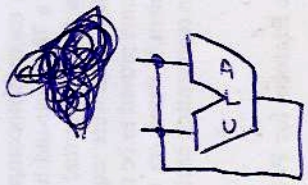
(b) $1/5 = 20\%$

Considerações

- nop's são contados como instruções, pois são resoluções por software.
- quando há pipeline, o tamanho do ciclo é o tamanho do estágio mais lento.
- Refazer: 4.13.2, 4.13.3

full-forwarding: temos todos os atalhos possíveis em todos os estágios

ALU-ALU-forwarding:



- É assumido que o resultado do beq é conhecido ao final do ID

4.29.4

IF ID EX MEM WB
IF ID EX MEM

(a) LoopOriginal: lw $1, 40($6)
add $5, $5, $1
sw $1, 20($5)
addi $6, $6, 4
addi $5, $5, -4
beq $5, $0, LoopOriginal

Assumindo Full-Forwarding

NovoLoop: lw $1, 40($6)
addi $6, $6, 4
add $5, $5, $1
sw $1, 20($5)
addi $5, $5, -4
beq $5, $0, NovoLoop

desenrolando 1x

lw $1, 40($6)
addi $6, $6, 4
add $5, $5, $1
sw $1, 20($5)
addi $5, $5, -4
lw $1, 40($6)
addi $6, $6, 4
add $5, $5, $1
sw $1, 20($5)
addi $5, $5, -4
beq $5, $0, Loop

lw $10, 40($6)
lw $1, 44($6)
addi $6, $6, 8
addi $5, $5, -8
add $11, $5, $10
add $5, $11, $1
sw $10, 28($11)
sw $1, 24($5)
beq $5, $0, Loop

2 - issue

lw $10, 40($6)	lw $1, 44($6)
addi $6, $6, 8	addi $5, $5, -8
add $11, $5, 10	
addi $5, $11, $1	sw $10, 28($11)
sw $1, 24($5)	beq $5, $0, loop

IF ID EX MEM WB
IF ID EX MEM WB

desenrolando 1x

(b) Loop: add $1, $2, $3
sw $0, 0($1)
addi $2, $2, 4
beq $2, $0, loop

add $1, $2, $3
addi $2, $2, 8
sw $0, -4($1)
sw $0, 0($1)
beq $2, $0, loop

2 - issue

add $1, $2, $3	addi $2, $2, 8
sw $0, -4($1)	sw $0, 0($1)
beq $2, $0, loop	

4.29.5

(a) Loop original = 6 x 2 (multiple issue) = 12
Loop desenrolado 5 → 12/5 = 2.4

(b) Loop O = 4 x 2 = 8
Loop D = 3 → 8/3 = 2.6

4.29.6

(a) 2 x 6 = 12 , 9 => 12/9 = 1.3

(b) 2 x 4 = 8 , 5 = 8/5 = 1.6

Considerações

- Emissão multipla (sempre prestar atenção nas restrições)
- renomear os registradores sempre que necessário quando for desenrolar o loop, mas sempre prestando atenção para não alterar a funcionalidade do mesmo.
- fazer mais exemplos de loop unrolling
- refazer : 4.29.4

Cache - P3

5.2.1 32 bit = 4 bytes

$\frac{16}{4} = 4$ inteiros de 32-bits

5.2.2

ⓐ I é considerado localidade temporal pois é referenciado o mesmo I a cada 8 "J" diferentes

ⓑ J é considerado localidade temporal pelo motivo citado acima

5.2.3

ⓐ J explora a localidade espacial em B[J][0]

ⓑ I explora a localidade espacial em A[I][J].

5.2.4

ⓐ 8000 linhas x 8 colunas

64000 — 32 bits

↳ x4 bytes = $\frac{256\,000 \text{ bytes}}{16}$ ⇒ 16.000 linhas de 16-bytes

ⓑ mesmo que a letra ⓐ

5.2.5

ⓐ I representa localidade temporal

ⓑ J " " "

5.2.6

ⓐ J em A(I,J)

ⓑ J em A(I,J) e I em A(J,I)

5.3.1

16 blocos de 1 palavra

(a)

$$\boxed{\begin{array}{l} Tag = 32\text{ bits} - (\log_2 n \text{ blocos} + \log_2 m \text{ palavras} + 2) \\ n \to \text{número de blocos da cache} \\ m \to \text{número de palavras por bloco} \end{array}}$$

$$Tag = 32 - (\log_2 16 + \log_2 1 + 2)$$

$$Tag = 32 - (4 + 0 + 2)$$

$$\boxed{Tag = 26 \text{ bits}}$$

	Tag	index	MISS/HIT	
1 →	00000000000000000000	0000	M	
134 →	00 10	0001	M	
212 →	00 11	0101	M	
1 →	00 00	0000	H	Aê!
135 →	00 10	0001	H	→ Aê!
213 →	00 11	0101	H	→ Aê!
162 →	00 10	1000	M	→ sobrescreve o 135
2 →	00 00	0000	H	→ Aê!
44 →	00 00	1011	M	→ sobrescreve o 2
41 →	00 00	1010	H	→ Aê!
221 →	00 11	0111	M	→ sobrescreve o 213

Hit rate = $\frac{5}{11}$

Miss rate = $\frac{6}{11}$

(b) Tag = 26 bits → calculada no (a)

	Tag	index	Miss/Hit	
6	0 - - - - 0	0001	M	
214	0 - - - - 11	0101	M	
175	0 - - - - 10	1011	M	
214	0 - - - - 11	0101	H	Aê!
6	0 - - - - 0	0001	H	Aê!
84	0 - - - - 01	0101	M	
65	0 - - - - 01	0000	M	→ sobrescreveu o 84
174	0 - - - - 10	1011	H	Aê!
64	0 - - - - 01	0000	H	Aê!
105	0 - - - - 01	1010	M	
85	0 - - - - 01	0101	H	Aê!
215	0 - - - - 11	0101	M	→ sobrescreveu o 85
	Hit rate: 5/12			
	Miss rate = 7/12			

5.3.2

(a)

31 … 6	5 … 1	0
25	3	1 2

	Tag	index	off	woff	
1	0000000	000	0	01	Miss
134	0000010	000	1	10	Miss
212	0000011	010	1	00	Miss
1	0000000	000	0	01	HIT
135	0000010	000	1	11	Miss, substituiu o 134
213	0000011	010	1	01	Miss, substituiu o 212
262	0000010	100	0	10	Miss
161	0000010	100	0	01	Hit
2	0000000	000	0	10	Miss, substituiu o 1
46	0000000	101	1	00	Miss
41	0000000	101	0	01	Hit, o 44 veio antes e o trouxe
221	0000011	011	1	01	Miss
	Miss rate = 8/12				
	Hit rate = 4/12				

(b)

	Tag	index	off	word	
6	000000	000	1	10	Miss
214	000011	010	1	10	miss
175	000010	101	1	11	miss
214	000011	010	1	10	Hit
6	000000	000	1	10	Hit
84	00001	010	1	00	miss, substituio o 214
65	00001	000	0	01	miss, substituio o 6
174	00010	101	1	10	miss, substituio o 175
64	00001	000	0	00	Hit
105	00001	101	0	01	miss, substituio o 174
85	00001	010	1	01	Hit
215	000011	010	1	11	miss, substituio o 85.

$Miss_{rate} = \frac{8}{12}$

$Hit_{rate} = \frac{4}{12}$

Revisão

Localidade Espacial: Quando temos um array e vamos percorrendo sucessivamente. A[i], i++. As palavras são organizadas continuamente na memória e quando a localidade espacial é explorada, isto ajuda muito no desempenho da cache. Um programa sem branches explora muito esta localidade na cache de instruções.

Localidade Temporal: ~~Que~~ Como a cache tende a guardar os melhores momentos dos dados, quando utilizamos um dado repetidamente ele tende a já estar na cache devido a utilização, isto melhora o desempenho.

divisão do endereço

Tag	index	off	00

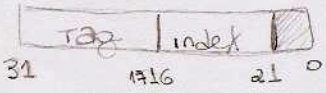
- 00 → sempre 2 bits qdo temos 32 bits para endereçar
- off → necessidade de endereçar mais de uma palavra por bloco
- index → necessidade de endereçar blocos
- Tag → o que sobra.

Refazer → 5.3.1a, 5.3.2a

5.3.4

(a) mapeamento direto

1 palavra



$2^{10} \cdot 2^{6} \cdot 2^{3}$ bits

1 palavra = 2^{5} bits

$\frac{2^{19}}{2^{5}} = 2^{14}$ palavras

Tag → 16 bits
index → 14 bits
woff → 0 bits
off → 2 bits

(b)

Tag = 16 bits
index = 13 bits
woff = 1 bit
off = 2 bits

Com uma cache de 16-word block teríamos mais substituições dos blocos devido ao fato de ser mapeamento direto e de termos reduzido as linhas e isso irá degradar o desempenho.

5.4.1

(a) 4 bits de offset sendo que 2 são para offset dentro da palavra sobram 2 bits de woffset e portanto são 4 palavras por linha.

(b) 5 bits
- 2 bits offset

3 bits woffset ∴ 8 palavras por linha

5.4.2

(a) $2^{6} \cdot 2^{2} = 2^{8}$ palavras

(b) $2^{7} \cdot 2^{3} = 2^{10}$ palavras

5.4.3

ⓐ 256 palavras . 4 = 1024 bytes . 8 = $\boxed{8192 \text{ bits de dados}}$

((bit Validade + Tag) + (32 * nº palavras)) * linhas = Tam Total Cache

((1 + 22) + (32 * 4)) * 2^6 = 9664 bits

$\frac{9664}{8192} = \boxed{1,1796}$

ⓑ 1024 palavras . 4 = 4096 bytes . 8 = $\boxed{32768 \text{ bits de dados}}$

((21) + (32 * 8)) * 2^7 = 35456 bits

$\frac{35456}{32768} = \boxed{1,0820}$

5.4.4

	② (20) Tag	⑥ index	④ off	② woff	
0	0000000	000000	0000	00	miss
4	0000000	000000	0001	00	Hit
16	0000000	000000	0100	00	Hit
132	000000	000010	0001	00	miss
232	0000000	000011	1010	00	miss
160	000000	000010	1000	00	Hit
1024	000000	10000	0000	00	miss Replace
30	0000000	000000	0111	10	Hit
140	000000	000010	0011	00	Hit
3100	000000	110000	0111	00	miss
180	0000000	000010	1101	00	Hit
2180	0000000	100010	0001	00	miss

Hit rate = $\frac{6}{12}$

Miss rate = $\frac{6}{12}$

(b)

	(18) tag	(7) index	(5) off	(2) woff	
0	00000000	0000000	00000	00	Miss
4	00000000	0000000	00001	00	Hit
16	00000000	0000000	00100	00	Hit
132	00000000	0000001	00001	00	Miss
232	0000000000	0000001	11010	00	Hit
160	00000000	0000001	01000	00	Hit
1024	0000000000	0001000	00000	00	Miss
30	00000000	0000000	00111	10	Hit
140	00000000	0000001	00011	00	Hit
3100	00000000	0011000	00111	00	Miss
180	00000000	0000001	01101	00	Hit
2180	00000000	0010001	00001	00	Miss

Hit rate = $7/12$

Miss rate = $\frac{5}{12}$

5.4.5

(a) $\frac{6}{12} = 50\%$

(b) $7/12 = 58{,}33\%$

5.46 está desenhada na questão 5.4.4

Revisão

Tamanho Total da Cache = (1 bit Validade + Tag) + (32 * nº palavras por bloco) * nº linhas

Mapeamento direto

Com poucas palavras por bloco: captura menos localidade espacial

muitas palavras por bloco: captura localidade espacial pois traz mais palavras quando procura por uma

Refazer: 5.34; 5.4.1, 5.4.2, 5.4.3, 5.4.4 (a) e (b).

5.3.3

d) C1

tag	index	off	woff
27	3	0	2

→ 2 ciclos

C2

tag	index	off	woff
27	2	1	2

→ 3 ciclos

C3

tag	index	off	woff
27	1	2	2

→ 5 ciclos

$\boxed{\text{Stall} = 25 \text{ ciclos}}$

	tag	IC1	IC2	IC3	offC1	offC2	offC3	C1	C2	C3
1	00000	000	00	0	X	0	00	M	M	M
134	000100	001	00	0	X	1	01	M	M*	M*
212	0000110	101	10	1	X	1	01	M	M	M
1	000000	000	00	0	X	0	00	H	M*	M*
135	0000100	001	00	0	X	1	01	H	M*	M*
213	0000110	101	10	1	X	1	01	H	H	H
162	0000101	000	00	0	X	0	00	M*	M*	M*
161	0000101	000	00	0	X	0	00	H	H	H
2	000000	000	00	0	X	0	00	M*	M*	M*
44	000001	011	01	0	X	1	11	H	M	M*
41	000001	010	01	0	X	0	10	M	H	H
221	0000110	111	11	1	X	1	11	M	M	M*

* → Replace

			TOTAL
Hitrate C1 = $4/12$ = 33,33%	0,16667 · 25 = 17 ciclos	12 * 2 = 24 ciclos	41
Hitrate C2 = $\frac{3}{12}$ = 25%	0,75 · 25 = 19 ciclos	12 * 3 = 36 ciclos	55
Hitrate C3 = $\frac{3}{12}$ = 25%	0,75 · 25 = 19 ciclos	12 * 5 = 60 ciclos	79

$\boxed{\text{C1 é melhor}}$

5.7.1

a) P1 → $\frac{1}{0.62 \cdot 10^9}$ = 1,6 GHz

P2 → $\frac{1}{0.66 \cdot 10^9}$ = 1,5 GHz

b) P1 → $\frac{1}{0.96 \cdot 10^9}$ = 1,04 GHz

P2 → $\frac{1}{1.08 \cdot 10^9}$ = 0,92 GHz ≈ 920 MHz

Caches nível um tem a característica do tempo levado para o hit ser igual ao tempo de 1 ciclo da CPU do processador, ou seja, no hit não há penalidades

5.7.2 AMAT → (Average Memory Acess Time)

a) AMAT = hitrate + (missrate * miss penality)

P1 Amat = 0,886 + (0,114 * 70)
= 8,6

P2 Amat = 0,92 + (70 * 0,08)
= 6,52

b) P1 Amat = 0,957 + (70 * 0,043)
= 3,967

P2 Amat = 0,966 + (70 * 0,034)
= 3.346

5.7.3

(a)

P1

$CPI_{com\ stall} = (I * 36\%) \times 13.4\% \times 3.13 = 4.63752$

$\boxed{CPI = 1 + 4.63752 = 5.63}$

P2 $CPI_{com\ stall} = (I * 36\%) \times 8\% * 3.07 = 3.08$

$\boxed{CPI = 1 + 3.08 = 4.08}$

$\boxed{\text{P2 é mais rápido}}$

(b)

P1 $CPI_{STALL} = (I * 36\%) * 4.3\% * 73 = 1.13$

$\boxed{CPI = 1 + 1.13 = 2.13}$

P2 $CPI_{STALL} = (I * 36\%) * 3.4 * 65 = 0,7956$

$\boxed{CPI = 1 + 0,7956 = 1,7956}$

$\boxed{\text{P2 é mais rápido}}$

$\boxed{\text{Ciclos STALLS} = (\text{Instruções} * \%\ \text{memoryAccess}) * \%\ \text{missRate} * \text{ciclos (Penalidade)}}$

$\boxed{CPI_{TOTAL} = CPI_{ideal} + CPI_{stalls}}$

Revisão

- Caches nível 1 tem como característica ter o tempo levado para buscar um dado que fez HIT igual ao tempo de 1 ciclo do relógio da CPU!

- AMAT = hit rate + (miss rate * miss penality)

- CPI stals = %memoryAccess * % missRate * ciclos (penalidade)

CPI total = CPI ideal + CPI stals

Refazer: 5.33 (olhar), 5.7.1, 5.7.3

5.7.4

AMAT semL2 = 8,6

(a)

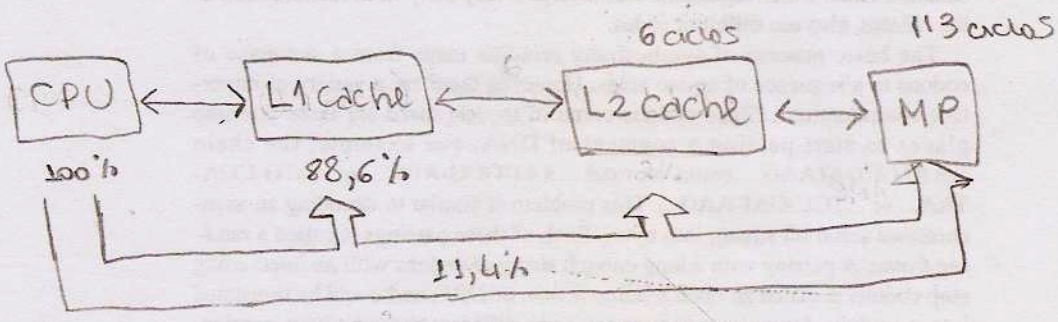
AMAT comL2 = 0,62 + 0,114 * 3,22 + 0,98 * 0,114 * 70 = 2,80 ns PIOR

(b) AMAT semL2 = 3,967

AMAT comL2 = 0,96 + 0,043 * 11,48 + 0,73 * 0,043 * 70 = 3,65 ns MELHOR

5.7.5

(a)



CPI = 1 + 0,114 * 6 + (0,114 * 0,98 * 113) = 14,30836 * 0,36 = 5.151009 6

(b) CPI = 1 + 0,043 * 12 + (0,043 * 0,73 * 73) = 3.80747 * 0,36 = 1,37

5.7.6

ⓐ P2 ainda é ⊕ rápido

$CPI_{P2} = 4.08 \cdot 0,66 = 2.6928$

$P1 = (1 + x * 6 + x * 0,98 * 113) * 0,62 = 2.6928$

$\boxed{x = 0.0286382 \rightarrow 2,86\%}$

ⓑ P1 é ⊕ Rapido

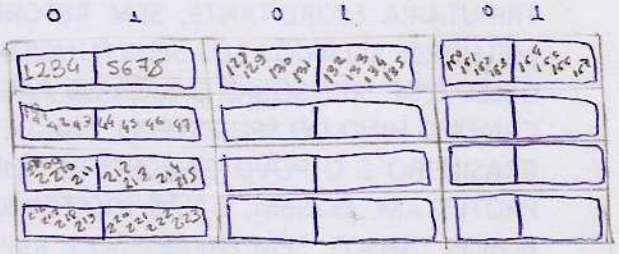
$1,37 * 0,96 = 1,3152$

$P2 = (1 + (0,36 * x * 65)) * 1,08 = 1,3152$

$\boxed{x = 0,00930674 \rightarrow 0,9\%}$

5.8.1

3-way-set-associative
2 word-blocks – 24 words (TOTAL)
using LRU



ⓐ

Tag	index	off	woff
27	2	1	2

	Tag	index	off	woff	Miss/Hit
1	000000	00	0	01	Miss
134	000100	00	1	10	Miss
212	000110	10	1	00	Miss
1	000000	00	0	01	HIT
135	000100	00	1	11	HIT
213	000110	10	1	01	HIT
162	000101	00	0	10	Miss
161	000101	00	0	01	HIT
2	000000	00	0	10	HIT
44	000001	01	1	00	Miss
41	000001	01	0	01	HIT
221	000110	11	1	01	Miss

Hit rate = $\frac{6}{12}$

Miss rate = $\frac{6}{12}$

(b)

	Tag	index	off	woff	MISS/HIT
6	000000	00	1	10	MISS
214	0000110	10	1	10	MISS
175	0000101	01	1	11	MISS
214	0000110	10	1	10	HIT
6	000000	00	1	10	HIT
84	000010	10	1	00	MISS
65	000010	00	0	01	MISS
174	0000101	01	1	10	HIT
64	000010	00	0	00	HIT
105	000011	01	0	01	MISS
85	000010	10	1	01	HIT
215	0000110	10	1	11	HIT

Missrate = 6/12

HITrate = 6/12

	0	1	0	1	0	1
00	0 1 2 3	4 5 6 7	64 65 66 67	68 69 70 71		
01	168 169 170 171	172 173 174 175	104 105 106 107	108 109 110 111		
10	208 209 210 211	212 213 214 215	80 81 82 83	84 85 86 87		
11						

5.8.2

(a)

fully associative
2-word-blocks
8 word (Total)

Tag	index	off	woff
29	0	1	2

	Tag	off	woff	MISS/HIT	
1	0000000001	0	01	MISS	tempo1
134	0001000 0	1	10	MISS	tempo2
212	0001101 0	1	00	MISS	T3
1	0000000 0	0	01	HIT	T4
135	0000100 00	1	11	HIT	T5
213	0000110 10	1	01	HIT	T6
162	0000101 00	0	10	MISS	T7
161	0000101 00	0	01	HIT	T8
2	0000000000	0	10	HIT	T9
44	0000000101	1	00	MISS	*Replace 135 T10
41	0000000101	0	01	HIT	T11
221	0000110 11	1	01	MISS	T12 *Replace 0 213

0	1	0	1	0	1	0	1
0..3	4..7	128..131	132..135	208..211	212..215	160..163	164..167

40..43 44..47 substituiu bloco no tempo 10!

215..213 220..223 substituiu bloco no tempo 12!

Missrate = 6/12

HITrate = 6/12

(b)

	Tag	off	woff	MISS	HIT	
6	000000	1	10	X		T1
214	0011010	1	10	X		T2
175	0101011	1	11	X		T3
214	0011010	1	10		X	T4
6	000000	1	10		X	T5
84	0010100	1	00		X	T6
65	0010000	0	01	X		T7
174	0010101	1	10	X		T8
64	010000	0	00		X	T9
105	001101	0	01	X		T10
85	0010101	1	01		X	T11
215	0010110	1	11			T12

tempo

Miss rate = 6/12

hit rate = 6/12

0	1	0	1	0	1	0	1
0..3	4..7	208..211	212..215	168..171	172..175	80..83	84..87

104..107 108..111 — substituí o bloco no tempo 10

168..171 172..175 — substituí o bloco no tempo 8

64..67 68..71 — substituí o bloco no tempo 7

208..211 212..215 — substituí o bloco no tempo 12

5.16.1

P1	P2	P1	P2
x[0]++			x[0]=2
	x[0]=2	x[0]++	
	x[1]++		x[1]++
x[1]=4		x[1]=4	
x[0]++	x[0]=2	x[0]++	x[0]=2
x[1]=4		x[1]=4	x[1]++
	x[1]++		

x0 x1 [2,4]

x0 x1 [3,4]

com coerência

x0 x1 [2,5]

x0 x1 [3,5]

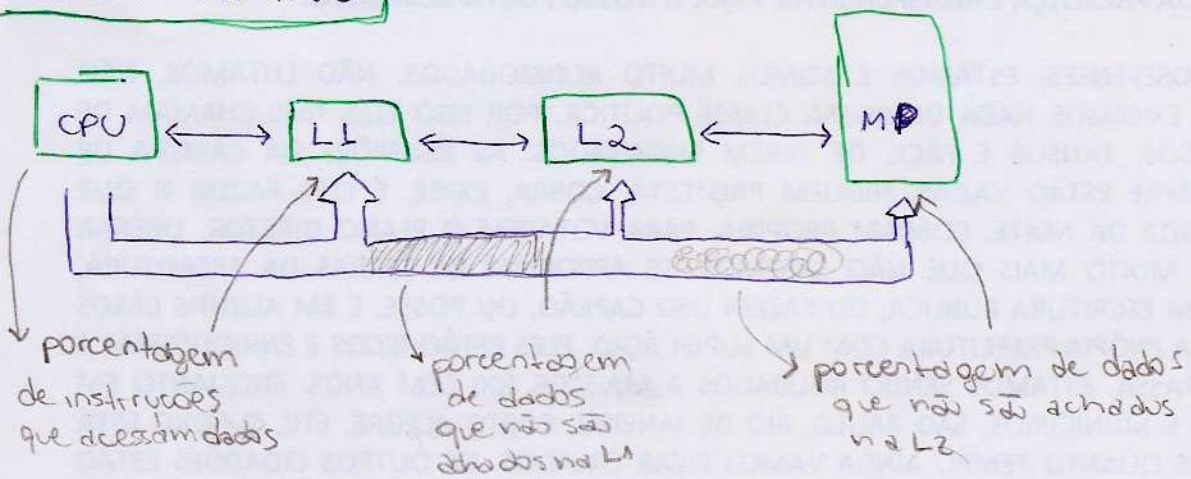
P1	P2	P1	P2

(2,1) (1,4) (2,1) (1,1)

sem coerência

Revisão

Cache multinível



*nesse caso as porcentagens são globais

Tipos de cache

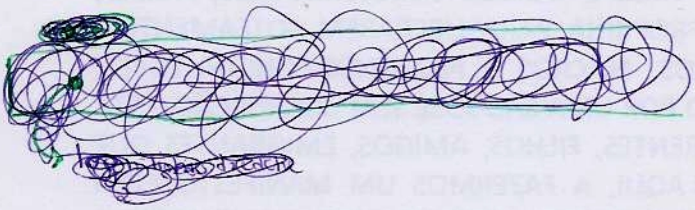
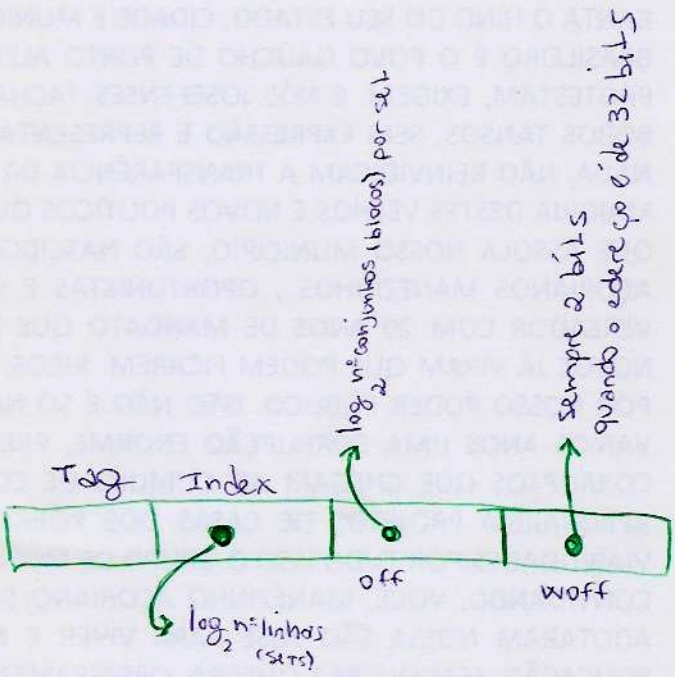
Mapeamento Direto (one-way set associative)

- 1 bloco por linha, muitas linhas.
- cada bloco tem sua tag.

N-way set associative (N > 1)

N = número de blocos por linhas.

- cada bloco tem sua tag.

Fully associative

- Muitos blocos, uma só linha.
- cada bloco tem sua tag.