

① $0x25A1C \rightarrow$

| | | | | |
|-----------|-------------|-------------|-------------|-------------|
| <u>10</u> | <u>0101</u> | <u>1010</u> | <u>0001</u> | <u>1100</u> |
| 2 | 4 | 4 | 4 | 4 |

18 bits

Gama de endereços = 0x00000 a 0x3FFF

Logo, 18 bits \rightarrow barramento de endereços

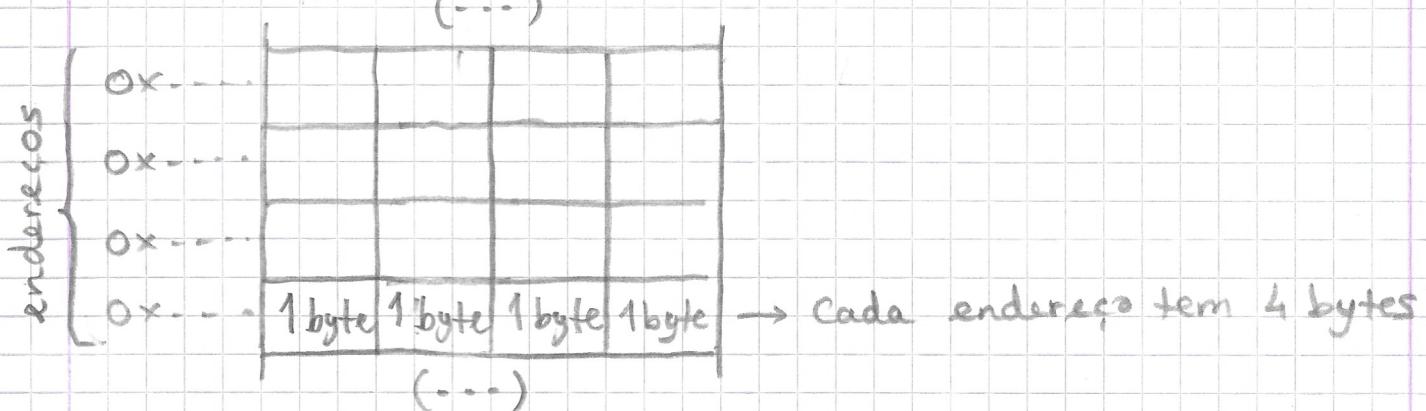
$2^{18} \rightarrow$ espaço de endereçamento

Resposta: b)

② byte-addressable \rightarrow cada endereço tem 1 byte (8 bits)

word-addressable \rightarrow cada endereço tem 4 bytes (32 bits)

memória
(...)



Como o barramento de endereços é 24 bits



o espaço de endereçamento é 2^{24} bytes

\rightarrow Se fosse byte-addressable a resposta era 2^{24} bytes

\rightarrow Como é word-addressable, temos de fazer:

$$2^{24} \times 2^2 = 2^{26} \text{ bytes}$$

4 (porque cada endereço tem 4 bytes)

Resposta: c)

③ Single-cycle \rightarrow 2 memórias \rightarrow Harvard

Operandos residem em registos internos ou na memória externa \rightarrow "Register-Memory"

Resposta: a)

④ adição em comp/para 2 \leftrightarrow adição signed

\rightarrow Quando a soma é signed, o bit do carry-out é ignorado, pois não tem significado

↓
Logo, não é possível tirar nenhuma conclusão

Nota:

\rightarrow Se fosse adição unsigned, o bit do carry-out simboliza a ocorrência de overflow

Resposta: c)

⑤ Codificação do JAL = codificação do J

| opcode | offset |
|--------|---------|
| 6 bits | 26 bits |

$$\bullet \$ra = PC + 4 = 0x0000A034 + 0x04 = \boxed{0x0000A038}$$

$$\bullet PC = offset \ll 2 = 0x00000D4 \ll 2 = \boxed{0x00000350}$$

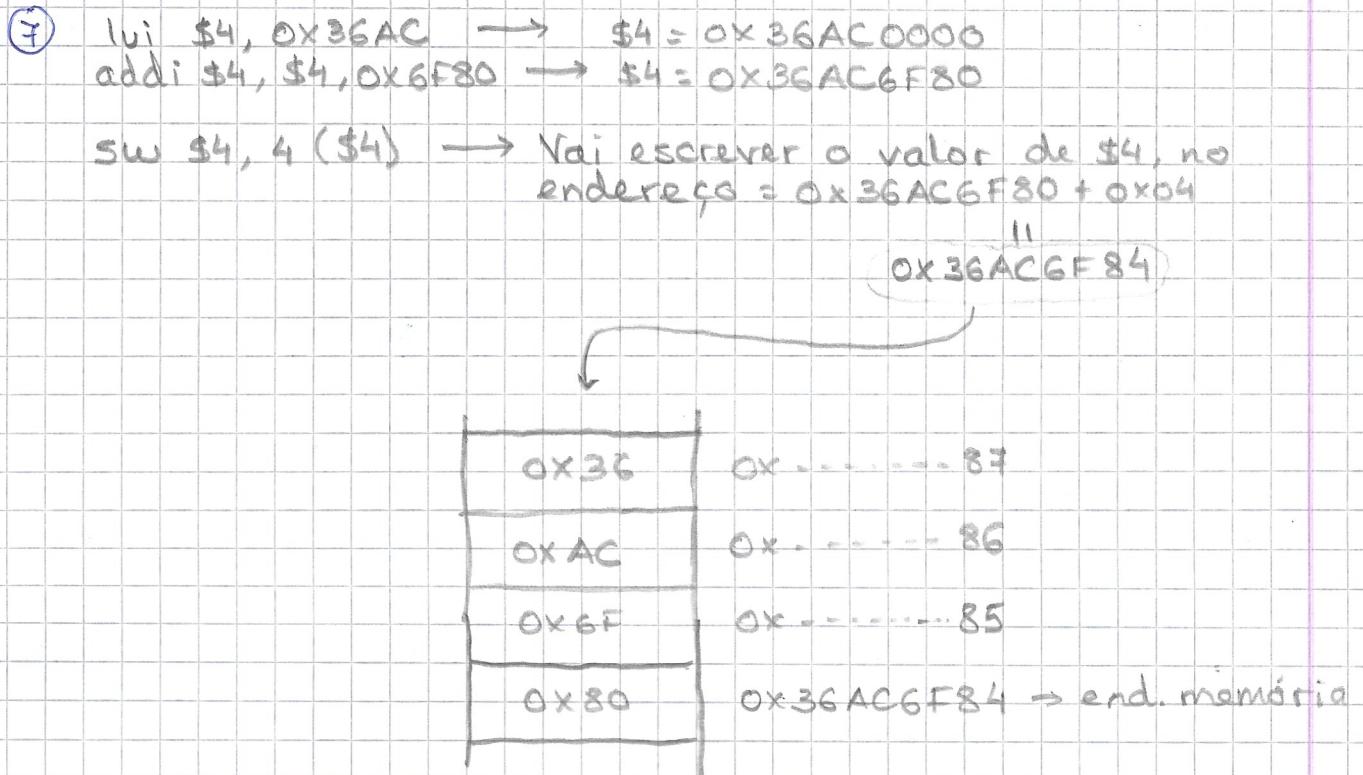
\downarrow
 $\ll 2$ (0000 1101 0100
 \underbrace{0011}_{3} \quad \underbrace{0101}_{5} \underbrace{0000}_0)

Resposta: d)

⑥ Salto condicional \rightarrow branches \rightarrow "beq"

Logo, é endereçamento relativo ao PC com deslocamento

Resposta: c)



lb \$4, 6(\$4) → Vai escrever em \$4 o valor que está no endereço

$$0x36AC6F80 + 0x06 = 0x36AC6F86$$

valor de \$4
antes da execução
da instrução lw

→ Olhando para o "desenho" da memória o valor que está no endereço 0x36AC6F86 é 0xAC

Como a instrução é "lb", temos de fazer a extensão de sinal

→ Logo, \$4 = 0xFFFFFFFFAC

Resposta: d)

⑧ Gama para operações com sinal → $[-2^{n-1}, 2^{n-1} - 1]$

Na instrução branch → offset tem 16 bits

$$BTA = (PC + 4) + (\text{offset} \ll 2)$$

↓ ↓
instrução é o mesmo que acrescentar 2 zeros
seguinte à direita do offset

↓
Logo, offset "passa a ter" 18 bits

$$\text{Por isso, } n=18 \implies \text{Logo, a gama} = [-2^{17}, 2^{17} - 1]$$

Resposta: a)

9) LO → fica com os 32 bits menos significativos da multiplicação

$$\$2 = 0xFFFFFFFFFE = -2$$

$$\$3 = 0x00000003 = 3$$

$$\$2 * \$3 = -2 \times 3 = -6 = \boxed{0xFFFFFFFFFA}$$

Resposta: d)

10) HI → fica com o resto da divisão

$$\$2 = 0xFFFFFFFFFE = -2$$

$$\$3 = 0x00000003 = 3$$

$$\frac{\$2}{\$3} = \frac{-2}{3}$$

→ resto = -2, pois o resto
fica sempre com o sinal.
do dividendo

↓
Logo

$$-2 = \boxed{0xFFFFFFFF}$$

Resposta: b)

11) bge \$5, 0xA3, target → $(\$5 \geq 0xA3) = \sim (\$5 < 0xA3)$

Slti \$1, \$5, 0xA3 → Se $\$5 < 0xA3$, $\$1 = 0x01$
Senão $\$1 = 0x00$

Logo, se $\$1 = 0x00$, então $\$5 \geq 0xA3$



Quando se verifica, é realizado
o salto

Por isso, a segunda instrução é beq \$1, \$0, target

Resposta: a)

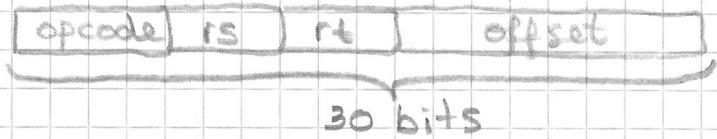
12) jr → não tem nenhuma restrição, logo pode saltar para qualquer endereço múltiplo de 4

Ou seja, de 0x00000000 a 0xFFFFFFFFC

↓
acaba em "C" porque tem
de ser múltiplo de 4

Resposta: d)

⑬ Codificação → Formato I



• opcode

→ Qual o espaço de endereçamento para 130 instruções?

$$2^{10} = 1024$$

$$2^9 = 512$$

$$2^8 = 256 \leftarrow \text{Tem de ser } 2^8 \text{ bytes}$$

$$2^7 = 128$$

↓
barramento de endereços

↓
8 bits

Logo, o opcode tem 8 bits também

• offset

$$\rightarrow \text{Gama} = [-32768, 32767] = \pm 32\text{K}$$

↓
Logo, o offset tem 16 bits como no MIPS

• rs e rt

Sendo que,

| | | | |
|--------|----|----|---------|
| opcode | rs | rt | offset |
| 8 bits | | | 16 bits |

$$30 - 8 - 16 = 6 \text{ bits} \rightarrow \text{Logo rs tem 3 bits e rt também tem 3 bits}$$

→ Como rs e rt têm 3 bits, o número de registos internos é $2^3 = 8$ registos

Resposta: b)

⑭ O que se pretende

↓
0000 0000 0000 0000 0000 xxxx xxxx
negar

$$\text{addi } \$1, \$0, 0xFFFF \rightarrow \$1 = 0xFFFFFFFF$$

nor \$3, \$3, \$1 → os 24 primeiros bits ficam a 0
os 8 últimos bits são negados

| nor |
|--------|
| 00 → 1 |
| 01 → 0 |
| 10 → 0 |
| 11 → 0 |

Resposta: d)

15) L1: .word 0, 1, 2 → reserva espaço para 3 words,
 L2: .ascii "NATAL" → ou seja, para 12 bytes
 .align 3 → reserva espaço do tamanho da string + o terminador '\0', sendo que cada caractere é 1 byte
 L3: .space 20 → o próximo endereço será múltiplo de 8 (2^3)
 → reserva espaço para 20 bytes

| | | |
|------------|------|-----------|
| 0x00000100 | 0x00 | → L1 |
| 01 | 0x00 | |
| 02 | 0x00 | |
| 03 | 0x00 | |
| 04 | 0x01 | |
| 05 | 0x00 | |
| 06 | 0x00 | |
| 07 | 0x00 | |
| 08 | 0x10 | |
| 09 | 0x00 | |
| 0A | 0x00 | |
| 0B | 0x00 | |
| 0C | 'N' | → L2 |
| 0D | 'A' | |
| 0E | 'T' | |
| 0F | 'A' | |
| 10 | 'L' | |
| 11 | 'O' | |
| 12 | ? | → align 3 |
| 13 | ? | |
| 14 | ? | |
| 15 | ? | |
| 16 | ? | |
| 17 | ? | |
| 18 | ? | → L3 |

Logo, L3 está no endereço 0x00000118

Resposta: a)

16) int *p; → &p → aponta para o endereço
 *p → aponta para o conteúdo

Resposta: c)

17) precisão dupla → excesso de 1023

Resposta: b)

(18) menor quantidade positiva $\rightarrow 1,0000 \times 10^{-126}$

| S | E | f |
|------|-------|---------|
| 1bit | 8bits | 23 bits |

• S = 0, pois é um nº positivo

• E = expoente + 127 = -126 + 127 = 1 = 0x01

• f = 0x0000000

Logo, $0|0000\ 0001\ 000\ 0000\ 0000\ 0000\ 0000$
 $= 0x00800000$

Resposta: d)

(19) Resposta: a)

(20) Resposta: c)

(21) Resposta: a)

(22) Saída de ALUOut na 4ª fase

Saída de ALUR_{Result} na 3ª fase

\rightarrow Numa instrução "lw", na 3ª fase é calculado o endereço de memória

$$\text{end_mem} = (\text{conteúdo de rs}) + \text{offset}$$

Resposta: b)

(23) A unidade de controlo do Pipeline não é uma máquina de estados, ou seja, é um círcuito lógico combinatório

Os sinais de controlo são gerados na 2ª fase, ou seja, em ID

Resposta: d)

(24) Resposta: b)

(25) delayed-branch \rightarrow Exemplo: `beq $1, $2, end
addi $3, $3, 1`

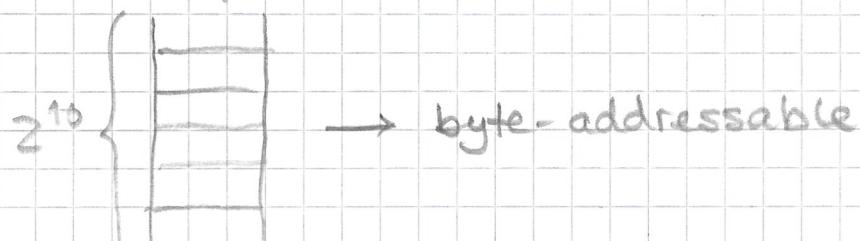


à instrução "addi" é realizada primeiro do que "beq", independentemente de saltar ou não

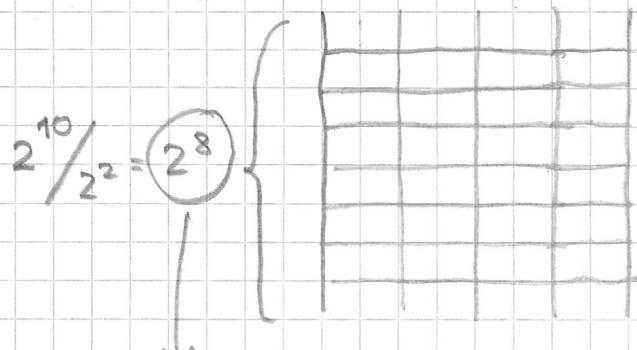
Resposta: a)

(26) $1 \text{ KByte} = 10 \text{ bits de barramento}$

\Downarrow
 $2^{10} \rightarrow \text{espaço de endereçamento}$



\rightarrow Se for word-addressable



\downarrow
espaço de endereçamento

\Downarrow
Logo, o barramento é de 8 bits (A_7 a A_0)

• Como os endereços são sempre múltiplos de 4, os bits A_1 e A_0 são sempre 0 e, por isso, são descartados



8 bits - A_7 a A_2

Resposta: c)

(27) frequência máxima de operação



tempo mais longo de todas as instruções $\rightarrow "t_{lw}"$

$$t_{lw} = t_{RM} + \underbrace{\max(t_{RFR}, t_{WC}, t_{ES})}_{\text{tempo mais longo desta}} + t_{ALU} + t_{RM} + t_{RFW}$$

3 operações, pois são executadas ao mesmo tempo

$$= 3 + \max(2, 1, 0) + 1 + 3 + 1$$

$$= 3 + 2 + 1 + 3 + 1 = 10 \text{ ns}$$

$$f = \frac{1}{t} = \frac{1}{10 \text{ ns}} = 100 \text{ MHz}$$

Resposta: b)

(22) RegDst, TorD, MemToReg são sinais de muxs, ou seja, só sabendo o valor deles, não podemos concluir qual a fase e a instrução que está a ser executada

Resposta: c)

(29) LUI \$reg, imm



escreve nos 16 bits mais significativos do \$reg, o valor de imm

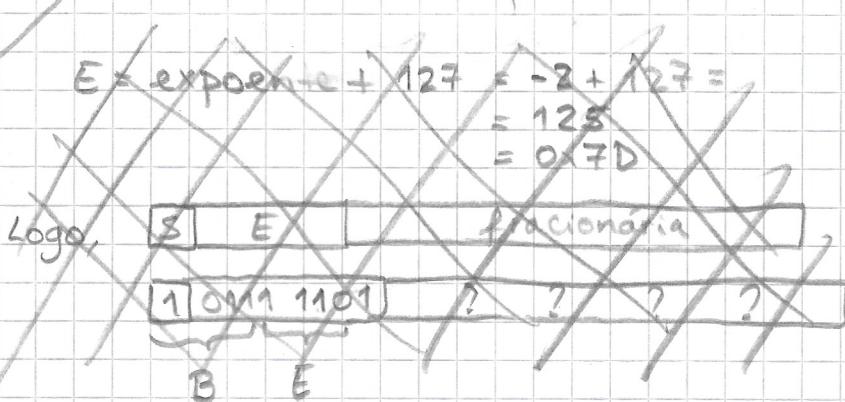
• Por isso, temos de acrescentar uma entrada no Mux3, pois é o mux que recebe o valor calculado na ALU numa instrução de "immediatos"

• E ligamos os 16 bits mais significativos aos 16 bits menos significativos do código máquina

Resposta: a)

(30) Resposta: d)

$$(31) -40,625 \times 10^{-2} = -4,0625 \times 10^{-1} = -0,40625 \times 10^0$$



• Como descobrir a parte fracionária em binário?

$$\begin{array}{r} 0,40625 \\ \times 2 \\ \hline 0,81250 \end{array}$$

$$\rightarrow \text{Parte fracionária} = 0,01101_2$$

$$\begin{array}{r} 0,81250 \\ \times 2 \\ \hline 1,62500 \end{array}$$

$$\rightarrow \text{Parte inteira: } 0$$

$$\begin{array}{r} 0,62500 \\ \times 2 \\ \hline 1,25000 \end{array}$$

$$\rightarrow 0,40625_{10} = 0,01101_2 \times 10^0 \quad \text{normalizando}$$

$$\begin{array}{r} 1,25000 \\ \times 2 \\ \hline 0,25000 \end{array}$$

$$= 1,101_2 \times 10^{-2}$$

$$\begin{array}{r} 0,25000 \\ \times 2 \\ \hline 0,50000 \end{array}$$

$$\rightarrow E = \text{exponente} + 127 = -2 + 127 = 125_{10}$$

$$\begin{array}{r} 0,50000 \\ \times 2 \\ \hline 1,00000 \end{array}$$

$$= 01111101_2$$

Logo fica, 1 0111 1101 1010000 | ...
= 0XBED00000

Resposta: b)

(32) Resposta: d)

(33) Estamos a trabalhar em base 10

↓ logo

$$\text{nº bits} = \log_2(10) = \frac{\log_{10}}{\log_2} = \frac{1}{0,3} = 3,3 \text{ bits}$$

→ Como queremos uma precisão de 4 casas decimais

$$\text{nº bits} = \lceil 4 \times 3,3 \rceil = \lceil 13,2 \rceil = 14 \text{ bits}$$

(arredonda para cima)

↓ parte
fracionária

• Sendo um formato de 20 bits,



Logo E ocupa 5 bits

→ Como queremos saber o excesso do expoente:

$$2^{n-1} - 1 = 2^{5-1} - 1 = 2^4 - 1 = \boxed{15}$$

↓ excesso do 15

Resposta: a)

(34)

| | |
|--------|-------|
| addi | |
| lw | |
| add | |
| and | |
| *(beg | |
| addi | |
| and | |
| sw | |

* como esta implementação tem um cyc-branch, o "addi" é realizado antes do "beg", ou seja está dentro do ciclo

Resposta: c)

(35) Fase WB de "add \$6,\$6,\$3" \leftrightarrow Fase ID de "beg \$3,\$2,L1"

Logo, o valor do registo PC é o valor de PC em "beg"

$$\begin{aligned} \hookrightarrow \text{PC}_{\text{beg}} &= \text{PC}_{1^{\text{a inst}}} + 5 * (\text{0x04}) && \rightarrow (\text{nº de saltos}) \\ &= \text{0x00006024} + \text{0x00000014} \\ &= \text{0x00006038} \end{aligned}$$

Resposta: b)

36) No décimo ciclo, a instrução que está na Fase EX, ou seja, que está a usar a ALU é o "Lse"
addi

Logo, o valor de saída da ALU é:

$$\begin{aligned} \$5 + 4 &= 0x7A34 + 0x04 \\ &= 0x00007A38 \end{aligned}$$

Resposta: d)