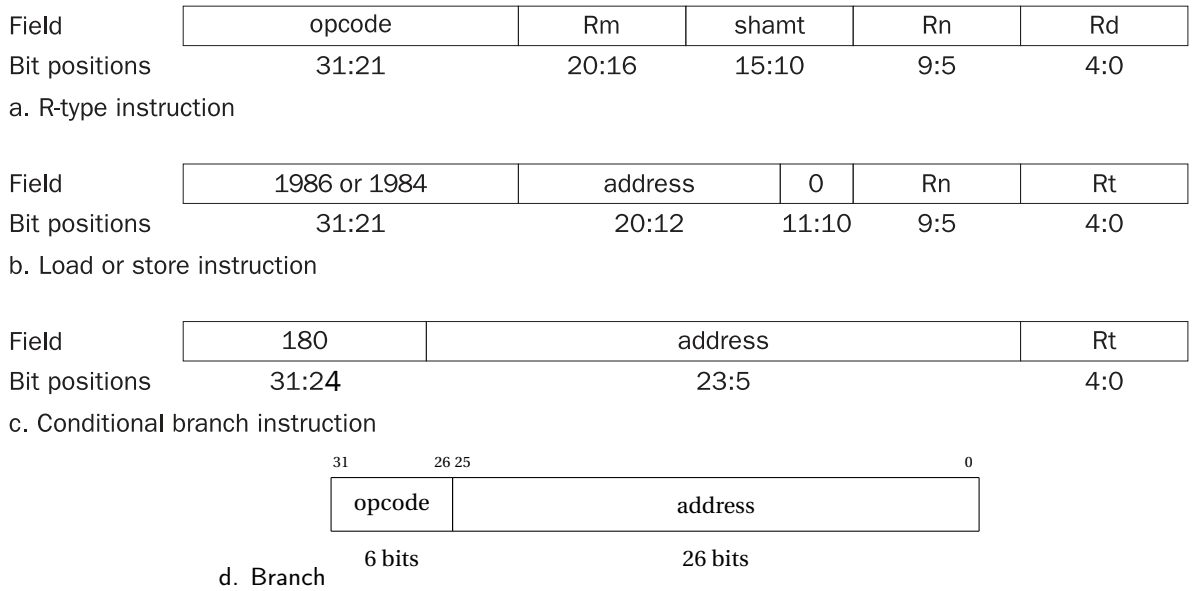
AOCO: Questões e exercícios adicionais (soluções)

Parte II

As questões de escolha múltipla (secção 1) e os problemas de resposta aberta (secção 2) foram retirados ou adaptados de testes de AOCO de anos anteriores.

Informação de referência



Instrução	Opcode
ADD	100 0101 1000
SUB	110 0101 1000
AND	100 0101 0000
ORR	101 0101 0000
LDUR	111 1100 0010
STUR	111 1100 0000
CBZ	101 1010 0
B	000 101

ALU trabalha em 3 contextos diferentes.

1. instruções lógico-aritméticas: `ALUOp[1:0]=10`
2. cálculo de endereços: `ALUOp[1:0]=00`
3. comparação: `ALUOp[1:0]=01`

Para programação, usar também **a folha de consulta** com as instruções mais comuns.

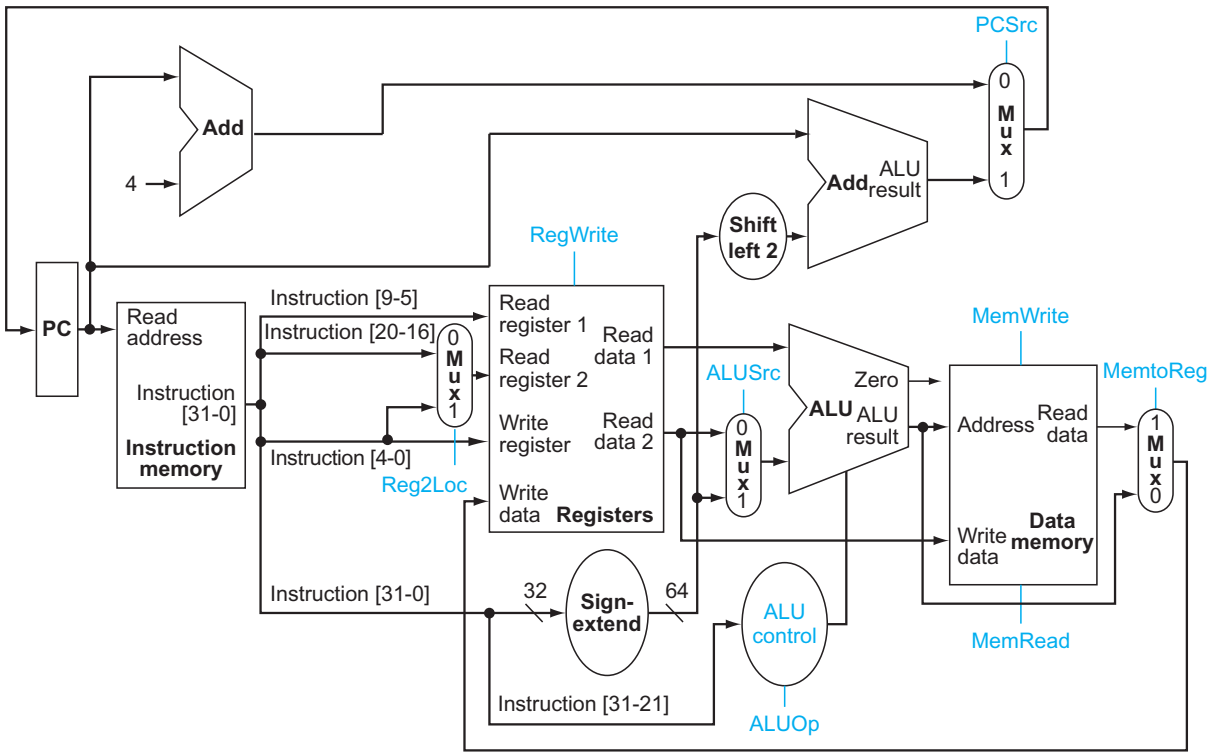


Figura 1: CPU com "multiplexers" e sinais de controlo

1 Questões de escolha múltipla

1. Assuma que a saída `Read data 2` do banco de registos é sempre 0. Que instrução ARMv8 não é afetada por esta anomalia?

   A. `STUR` B. `ADD` C. `CBZ` D. `LDUR`

2. Que instrução ARMv8 poderá ser executada se `MemtoReg`=0, `Reg2Loc`=1 e `ALUSrc`=1?

   A. `ORR` B. `CBZ` C. `STUR` D. `LDUR`

3. Que instrução ARMv8 tem o código `0xCB0201E8`?

   A. `ADD X15,X8,X2` B. `SUB X2,X15,X8` C. `SUB X15,X2,X8` D. `SUB X8,X15,X2`

4. Relativamente a sub-rotinas, qual das seguintes afirmações é falsa?

   A. Sub-rotinas terminais devem preservar o valor de `X30` antes de invocarem outras sub-rotinas.

   B. Uma sub-rotina do tipo função devolve um valor como resultado.

   C. Uma sub-rotina do tipo procedimento não devolve resultados.

   D. Na invocação de uma sub-rotina, o endereço da instrução seguinte é guardado no registo `X30`.

5. Um programa gasta 75 % do tempo em transferências de dados para outro computador via rede sem fios. Quantas vezes é preciso aumentar a velocidade de transferência para obter uma redução do tempo de execução do programa *(speedup)* de duas vezes?

   A. 4 B. 1,5 C. 3 D. 2

6. Para um dado programa, o processador P1 com $F_1 = 1\,\text{GHz}$ apresenta o mesmo tempo de execução que o processador P2 com $F_2 = 1{,}25\,\text{GHz}$. O tempo de execução de P1 fica maior que o de P2 se:
   A. passar a usar $F_2 = 1\,\text{GHz}$;
   B. aumentar o valor do CPI médio de P2;
   C. reduzir o valor do CPI médio de P1;
   D. aumentar 1,3 vezes o período do relógio de P1.

7. O tempo de execução de um programa está repartido entre a execução de instruções da classe A (60 % do tempo) e da classe B (40 % do tempo). Qual das seguintes alterações leva ao melhor desempenho?
   A. diminuir para metade o tempo de execução das instruções de classe A;
   B. diminuir o tempo de execução das instruções de classe B para um quarto do tempo original;
   C. reduzir o tempo de execução das instruções de classe A para um terço e aumentar o tempo de execução das instruções de classe B para o dobro;
   D. reduzir 1,5 vezes o tempo de execução das instruções de classe A e reduzir o tempo de execução das instruções de classe B para metade.

8. Um programa de cálculo científico gasta 80 % do seu tempo de execução em operações numéricas. Este tempo está repartido da seguinte forma:
   - operações aritméticas: 40 %
   - operações trigonométricas: 60 %

   Um novo método de cálculo das funções trigonométricas reduzirá o respetivo tempo de execução em 4×. Qual dos valores indicados se aproxima mais da melhoria de desempenho (*speedup*) global que esta medida produzirá?

   A. 1,82 B. 2,40 C. 1,56 D. 2,62

9. Uma memória *cache* com 64 B/bloco contém 32 KiB de dados. Quantos blocos tem?

   A. 512 B. 2048 C. 256 D. 1024

10. Qual das seguintes afirmações sobre uma memória *cache* do tipo *write-through* é falsa?
    A. O conteúdo da memória principal está sempre atualizado.
    B. No caso de uma falta num acesso de leitura (*read miss*) o valor é lido da memória principal e colocado na *cache* atualizando a etiqueta e alterando o valor de **v** para 1.
    C. No caso de uma falta num acesso de escrita (*write miss*) o valor é escrito na memória principal e na memória *cache* atualizando a etiqueta e alterando o valor de **v** para 1.
    D. No caso de um acerto num acesso de leitura (*read hit*), o valor é lido da memória *cache*.

11. Um CPU está equipado com uma memória *cache* unificada com taxa de faltas $t_f = 0{,}1$. Em média, 20 % das instruções de um programa acedem a dados. Qual das seguintes alternativas de *split cache* apresenta o mesmo desempenho?
    A. I-*cache* com $t_f = 20\,\%$ e D-*cache* com $t_f = 8\,\%$;
    B. I-*cache* com $t_f = 8\,\%$ e D-*cache* com $t_f = 20\,\%$;
    C. I-*cache* com $t_f = 5\,\%$ e D-*cache* com $t_f = 5\,\%$;

D. I-*cache* com $t_f = 10\,\%$ e D-*cache* com $t_f = 20\,\%$.

12. A memória principal usada com um CPU de 2 GHz tem um tempo de acesso de 60 ns. O acesso à memória *cache* demora 0,5 ns. Qual deve ser o valor máximo da taxa de faltas para que o tempo médio de acesso a memória seja 10 ciclos?

    A. 5 % B. 7,5 % C. 10 % D. 2,5 %

13. Um programa gasta 50 %do tempo a executar cálculos de vírgula flutuante. Qual é o ganho de rapidez *(speedup)* que se poderia obter se a unidade de vírgula flutuante fosse 5 vezes mais rápida?

    A. 2,5 B. 5/3 C. 10/3 D. 2

14. Qual das seguintes afirmações sobre uma memória *cache* do tipo *write-back* é verdadeira?

    A. Existe a possibilidade de serem feitos 2 acessos a memória principal para apenas um acesso a memória *cache*.
    B. O conteúdo da memória principal está sempre atualizado.
    C. Pode haver um acesso a memória principal mesmo que o acesso a memória *cache* seja um acerto (*hit*).
    D. Não pode ser usada como *D-cache*.

15. Considere duas versões do mesmo programa a correrem no mesmo processador. A versão A foi produzida pelo compilador C_A e executa 2×10^{10} instruções; a versão B foi produzida pelo compilador C_B e executa $1{,}25 \times 10^{10}$ instruções. Para a versão A, o valor de CPI_A é 2. A versão A é 25 % mais rápida que a versão B. Qual é o valor de CPI da versão B?

    A. $CPI_B = 25/16$ B. $CPI_B = 4$ C. $CPI_B = 57/16$ D. $CPI_B = 3$

16. Um programa científico despende 80 % do seu tempo a executar operações de vírgula flutuante. Para tornar o programa 4 vezes mais rápido pretende-se usar uma nova uma unidade de vírgula flutuante. Quanto mais rápida que a anterior deve ser essa unidade?

    A. 16; B. 8; C. 12; D. É impossível.

17. Assuma as seguintes condições iniciais: `X0=0x8abc7520` e `X2=0x11110000` Determine o valor do registo `X0` após ser executada a instrução `ORR X0,X0,X2`.

    A. `0x8abc0000` B. `0x9bbd5702` C. `0x9bbd7520` D. `0x8bbc5720`

18. Uma memória *cache* tem 32 blocos com 8 palavras por bloco e etiquetas de 16 bits. Quantos bits tem cada endereço?

    A. 24 B. 21 C. 26 D. 32

19. Um processador funciona a 3 GHz. Nesse processador, o programa P1 apresenta um CPI médio de 2,4 e o programa P2 apresenta um CPI médio de 2. Sabendo que o tempo de execução de cada um dos programas é 2 s, indique a afirmação verdadeira.

    A. O programa P2 executa menos instruções que o programa P1.
    B. O programa P2 executa 3×10^9 instruções.
    C. Se a frequência baixar para 2 GHz, o programa P2 passa a ser mais rápido que P1.
    D. Se o CPI de P1 aumentar, P2 fica mais lento que P1.

20. O desempenho de uma memória *cache* unificada usada com um CPU de 1 GHz foi considerado insuficiente. Sabendo que o CPI de base é 1, indique a alteração que leva à maior redução do CPI efetivo.

A. Baixar a taxa de instruções *load/store* de 50 % para 20 %.

B. Baixar o tempo de acesso à memória externa de 100 ns para 80 ns.

C. Baixar a taxa de faltas de 15 % para 10 %.

D. Nenhuma das outras alterações indicadas reduz o CPI efetivo.

2 Problemas de resposta aberta

Nota: Justificar todas as respostas e apresentar todos os cálculos.

1. O fragmento de código ARMv8 abaixo aplica a sub-rotina `calc` aos elementos de uma sequência de "double words" e acumula os resultados das invocações em `X21`. O endereço-base da sequência está inicialmente em `X19` e o número de elementos em `X20`.

   (a) Completar o fragmento.

```
        mov     X21, XZR
ciclo:  cbz     X20, L1          // terminar ciclo
        ldur    X0, [X19]
        bl      calc             // invocar sub-rotina
        add     X21, X21, X0     // usar o resultado
        add     X19, X19, 8
        add     X20, X20, -1
        b       ciclo
L1:     ....    // fim da execução do fragmento


calc:   eor     X1, X1, X1       // inicializar X1 com zero
LC1:    cbz     X0, LC2          // terminar?
        and     X2, X0, 1
        add     X1, X1, X2
        lsr     X0, X0, 1
        b       LC1
LC2:    mov     X0, X1
        ret                      // fim da sub-rotina
```

   ▶ Considerar que a sequência processada tem 3 valores: {170, 42, 450}.

   (b) Determinar o número de instruções executadas pela sub-rotina `calc` quando é chamada pela primeira vez.

   Assumindo que as instruções de alteração do fluxo de execução (condicional ou incondicional) têm CPI=2 e todas as outras têm CPI=1, determinar também o valor de CPI médio para este fragmento ao processar a sequência indicada. Mostrar todos os cálculos.

   > A sub-rotina determina quantos bits 1 compõem a representação binária do argumento que recebe em `X0`. Para isso possui um ciclo no qual é determinado o valor do bit menos

significativo. Em cada iteração o conteúdo de `X0` é deslocado um bit para a direita. O ciclo termina quando `X0` é zero.
Da primeira vez que a sub-rotina é chamada o argumento é 170 (0..00 1010 1010), pelo que, o ciclo é repetido 8 vezes e o número de instruções executadas é:

$1(\texttt{eor}) + 8 \times 5(\texttt{cbz até b}) + 1(\texttt{cbz}) + 1(\texttt{mov}) + 1(\texttt{ret}) = 1 + 40 + 3 = 44$

O número de ciclos correspondente é $1 + 8 \times (2 + 1 + 1 + 1 + 2) + 2 + 1 + 2 = 1 + 56 + 5 = 62$.

CPI = nº ciclos / nº instruções = 62/44 = 1,41

(c) Considerar agora o funcionamento da sub-rotina `calc` quando recebe argumentos de valor 2^k (k inteiro, $0 \leq k \leq 63$). Explicar o valor do resultado da sub-rotina e determinar o número de instruções executadas (em função de k).

A representação binária de um argumento com o valor 2^k tem um só bit 1. Assim sendo, o resultado da sub-rotina é 1.

O número de iterações realizadas é $k+1$ e o número de instruções executadas é $1 + (k+1) \times 5 + 3 = 5 \times k + 9$.

2. A sub-rotina `substitui` procura a primeira ocorrência de um número N numa sequência de "double words", substituindo esse número por 0 (zero). Os parâmetros da sub-rotina são, por ordem, os seguintes:1) endereço-base da sequência; 2) número de elementos da sequência; 3) valor de N.

(a) Completar o código da sub-rotina tendo em atenção as convenções relacionadas com o uso de registos.

```
substitui:  cbz    __X1__, final      // terminar?
            ldur   X5, [X0]           // obter um valor da sequência
            cmp    __X2__, X5         // é o valor procurado?
            b.eq   LS1
            add    X0, __X0__, 8       // preparar próxima iteração
            sub    X1, X1, __1__
            b      substitui
L1:         stur   __XZR__, [X0]      // substituir valor na sequência
final:      ret                       // retornar
```

(b) Supondo que a sequência é {12, 56, 17, 21, 72, 7} e que N=21, determinar quantas instruções são executadas pela sub-rotina `substitui`.

Quando o valor do elemento da sequência é diferente de N, uma iteração do ciclo executa 7 instruções. Quando esse valor é igual a N são executadas 6 instruções: 4 para terminar a iteração e as 2 instruções a seguir à etiqueta `LS1`.

Para N=21, tem-se 3 elementos diferentes de N no início da sequência. Logo, o número

de instruções executadas é $3 \times 7 + 6 = 27$.

(c) Suponha que o programa a que pertence a sub-rotina anterior é executado em dois computadores A e B. O período do sinal do relógio dos computadores A e B é 300 ps e 400 ps respetivamente. O número de ciclos de relógio consumidos por instrução (CPI) é 4 no computador A e 2,5 no computador B. Determinar qual dos computadores é o mais rápido a executar o programa.

$$t_{\text{exec\_A}} = N_{\text{instr}} \times 4 \times 300 = 1200 \times N_{\text{instr}} \text{ ps}$$

$$t_{\text{exec\_B}} = N_{\text{instr}} \times 2{,}5 \times 400 = 1000 \times N_{\text{instr}} \text{ ps}$$

Então,

$$\frac{t_{\text{exec\_A}}}{t_{\text{exec\_B}}} = \frac{12}{10} = 1{,}2$$

O computador *B* é o mais rápido.

3. A sub-rotina `sumsel` retorna a soma dos elementos de uma sequência (de N "double words") que pertencem ao intervalo $[a; b]$.Os parâmetros da sub-rotina são, por ordem, os seguintes: 1) endereço-base da sequência; 2) número de elementos da sequência; 3) valor de a; 4) valor de b.

(a) Completar a sub-rotina tendo em atenção as convenções relacionadas com o uso de registos.

```
sumsel:     eor     X5, X5, __X5__        // inicializar acumulador
loop:       cbz     X1, __fim__           // terminar?
            ldur    X6, [X0]
            cmp     __X6__, X2            // limite inferior
            b.lt    cont
            cmp     X6, X3                 // limite superior
            __b.gt__   cont
            add     X5, X5, __X6__
cont:       add     X0, __X0__, 8
            add     X1, X1, __-1__
            __b__   loop
fim:        mov     __X0__, X5
            ret
```

(b) Para a sequência {-3, 3, 6, 5, 0, -5, 8, 2, -1} e intervalo $[-1; 6]$, determinar quantas instruções são executadas pela sub-rotina `sumsel` e qual o resultado.

- Iterações para elementos da sequência inferiores a a executam 7 instruções.
- Iterações para elementos da sequência superiores a b executam 9 instruções.

- Iterações para elementos da sequência dentro do intervalo executam 10 instruções.

Neste caso, temos N=9: Existem 2 elementos na primeira condição, 1 na segunda e 6 na terceira. A primeira e duas últimas instruções do código são executadas apenas 1 vez cada. A instrução `cbz` é executada ainda uma vez após as 9 iterações.
No total: $2\times7+1\times9+6\times10+4=87$ instruções executadas.
O resultado (valor final de `X0`) é 15.

(c) O modelo do processador usada para a execução da sub-rotina emprega um sinal de relógio com a frequência de 1 GHz. O tempo de execução da sub-rotina com os dados da alínea (b) é de 170 ns. Determinar o valor médio de ciclos por instrução (CPI).

Usando a fórmula para o desempenho: $T_{exec} = N \times CPI \times \frac{1}{F}$, pode determinar-se CPI por $CPI = F \times T_{exec} \times \frac{1}{N}$.
Como $N = 87$, obtém-se: $CPI = 1\times10^{9}\times170\times10^{-9}\times\frac{1}{87} = \frac{87}{43}$.

4. Considere o CPU ARMv8 simplificado, apresentado na figura 1, e que o valor em cada registo $\mathtt{X}_i$ é $i+2$. A latência de componentes usados no CPU é a seguinte (componentes não indicados têm latência nula):

I-Mem	Add	Mux	ALU	Regs	D-Mem	Control	ALU control	
400	100	30	130	220	350	80	40	(ps)

(a) Indique o valor dos seguintes sinais de entrada/saída de componentes e sinais de controlo para a execução da instrução `CBZ X1, fim`:

`Read register 2` = 1 ; `Write register` = 1 ; `Write data` de D-Mem = 3

`ALUSrc` = 0 ; `PCSrc` = 0 ; `MemtoReg` = X

(b) Determine o caminho crítico da instrução `STUR X7, [X2, #-4]` e a respetiva latência.

A instrução `STUR X7, [X2,-4]` guarda o conteúdo do registo `X7` no endereço de memória dado por `X2-4`.
A unidade de controlo do CPU recebe o código da instrução ao fim de 400 ps. A sua latência (80 ps) determina que os sinais de controlo ficam disponíveis aos 480 ps.
Para esta instrução há a considerar três caminhos constituídos por componentes importantes para a sua execução.

- Atualização de `PC`: O cálculo do endereço da próxima instrução utiliza `Add` e `Mux` (controlado por `PCSrc`). A entrada 0 de `Mux` fica disponível aos 100 ps, mas `PCSrc` (igual a 0 para esta instrução) só fica pronto aos 480 ps. Logo, o novo valor de `PC` fica disponível ao fim de 480+30=510 ps.
- Cálculo do endereço de acesso à memória: Este caminho inclui a ALU, pelo que deve verificar-se ao fim de quanto tempo estão disponíveis as suas três entradas:

– entrada superior (endereço-base): o valor em `Read data 1` fica disponível aos 400+220=620 ps;

– entrada inferior (valor imediato): a entrada 1 de `Mux` controlado por `ALUSrc` está pronta aos 400 ps mas `ALUSrc`=1 só surge aos 480 ps, pelo que a entrada inferior de `ALU` fica pronta em 480+30=510 ps;

– entrada proveniente de `ALU control`: como `ALUOp` está disponível aos 480 ps então a saída de `ALU control` fica disponível aos 480+40=520 ps.

Conclui-se desta análise que a entrada mais demorada da ALU é `Read data 1`, pelo que `ALU result` é obtido do caminho a que pertencem `I-Mem` → `Regs` → `ALU` e o tempo que demora a ser calculado é 400+220+130=750 ps. Deste caminho resulta uma latência superior à do caminho que implementa a atualização de `PC`.

- Obtenção do valor a escrever em memória: O valor a escrever é obtido da saída `Read data 2`, demorando mais 220 ps que a saída de `Mux` controlado pelo sinal `Reg2Loc`. Como este está correto aos 480 ps, então `Read register 2` fica pronto aos 480+30=510 ps e `Read data 2` surge aos 510+220=730 ps. Esta latência é inferior à de `ALU result`.

Da análise apresentada conclui-se que o caminho crítico para a instrução `STUR` é

`I-Mem` → `Regs` → `ALU` → `D-Mem`

e o valor da latência é 400 ps + 220 ps + 130 ps + 350 ps = 1100 ps

(c) Determine a partir de que valor da latência da unidade de controlo o sinal `Write data` de D-Mem pertence ao caminho crítico da instrução `STUR`.

Para que o sinal `Write data` de D-Mem pertença ao caminho crítico da instrução `STUR` é necessário que demore mais a ser obtido do que o endereço. Este surge ao fim de 400+220+130=750 ps.

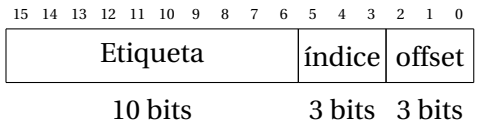
Portanto, `Read register 2` deve surgir depois de 750-220=530 ps.

Logo, $400 + t_{\text{control}} + 30 > 530$, pelo que $t_{\text{control}} > 100$ ps.

5. A tabela seguinte apresenta o conteúdo (em hexadecimal) de uma memória *cache* do tipo *write-back* com 8 blocos de 8 bytes usada como *D-cache* num CPU com endereços de 16 bits.

bloco	conteúdo 7 6 5 4 3 2 1 0	etiqueta	v	d
0	aa cc de hf 34 33 11 01	235	1	0
1	bb ad 45 4f af de 21 99	391	1	1
2	cc 34 ab 1f 56 cd ff ff	023	1	1
3	dd 67 22 2b 32 56 32 21	198	0	1
4	ee 32 11 9f aa ba ab bb	311	1	0
5	ff 10 00 04 01 02 03 04	278	0	0
6	11 03 41 32 cc dd ee ff	212	1	1
7	22 01 65 01 05 06 07 08	387	0	1

(a) Como é decomposto o endereço para acesso à memória *cache*? Justifique.



Offset: 3 bits. Como o conteúdo de cada bloco tem 8 bytes, serão necessários 3 bits para designar um byte do bloco.
Índice: 3 bits. Como a cache possui 8 blocos, são necessários 3 bits para representar os índices de 0 a 7.
Etiqueta: 10 bits. Como temos 16 bits e já gastámos 6, os restantes ($16-6=10$) serão para a etiqueta.

(b) Indique (se possível) o valor (byte) em memória principal no endereço `0xc467`. Justifique.

Decompondo o endereço temos: `0xC467` → `1100 0100 0110 0111` → `1100010001 100 111`.
Portanto, etiqueta: 311_{16}, índice: 4, *offset*: 7.

As etiquetas são iguais e o bloco é válido porque `v=1` (*read hit*).

O valor em memória *cache* é `0xEE` (elemento 7 do bloco 4) e corresponde ao valor em memória principal porque `d=0`.

(c) Explique quais as alterações que ocorrem na *cache* e na memória principal durante a leitura do valor (byte) residente no endereço `0xe48d`.

Decompondo o endereço temos: `0xE48D` → `1110 0100 1000 1101` → `1110010010 001 101`.
Portanto, etiqueta: 392_{16}, índice: 1, *offset*: 5.

Como as etiquetas são diferentes, ocorre uma falta de leitura *read miss*. Os dados do bloco são válidos (`v=1`), mas não correspondem à posição de memória pretendida.

Como `v=1` e `d=1` é necessário fazer *write-back*: o valor `0x45` é escrito na memória principal no endereço `0xE44D` (endereço calculado com a etiqueta residente em *cache*).

É necessário ler de memória principal o bloco que contém o valor no endereço `0xE48D`, escrevendo-o na posição 1 da *cache* (posição 5) e atualizando a etiqueta desse bloco para 392. Os indicadores devem ficar com os valores `v=1` e `d=0`.

6. Um CPU com endereços de 24 bits está equipado com uma memória *cache* de dados do tipo *write-through*. Etiqueta e índice têm, respetivamente, 14 e 6 bits de comprimento.

(a) Determinar o número de blocos e o número de bytes por bloco desta memória *cache*.

Como o índice tem 6 bits, a memória *cache* tem 2^6=64 blocos.
Como ficam 24-14-6=4 bits para o deslocamento, cada bloco tem 2^4=16 bytes por bloco.

(b) Considerar a seguinte situação. São realizadas sucessivamente leituras (de uma palavra) das seguintes posições de memória:

`0x3B7C94`, `0x3B6C90`, `0x3B6C98`, `0x3B6C94`

Quantos blocos são transferidos de memória principal para memória *cache* por causa dos três últimos acessos?

A primeira leitura (endereço `0x3B7C94`) afeta o bloco 9 (001001_2):

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23		
00111011011111	001001	1000
etiqueta	índice	desl.

Seja *hit* ou *miss*, a etiqueta do bloco 9 é 00111011011111_2 após a execução do acesso.
A segunda leitura (endereço `0x3B6C90`) afeta o mesmo bloco (9):

00111011011011	001001	0000

Como a etiqueta é diferente daquela que está em *cache*, trata-se de uma falta (*miss*) e é lido um bloco de memória principal. A etiqueta em memória *cache* é, agora, 00111011011011_2.
Os dois endereços seguintes levam sempre a *hit*, já que correspondem a partes do mesmo bloco (etiqueta e índices iguais aos do 2º acesso).

00111011011011	001001	1000
00111011011011	001001	0100

Logo, não existem mais acessos a memória principal: os últimos três acessos provocam a leitura de 1 bloco da memória principal.

(c) A memória *cache* é usada num sistema em que a penalidade de faltas é de 80 ciclos. Qual deve ser o valor máximo da taxa de faltas desta *cache* para que o número médio de ciclos de protelamento *no acesso a dados* não exceda 10?

O número de ciclos de protelamento (por operação de acesso a dados) C_p é dado pela número médio de acessos a dados que resultam em falta vezes a penalidade p_f (a dividir pelo número de acessos a dados N_d.

$$C_p = \frac{(N_d \times m_d) \times p_f}{N_d} = m_d \times p_f$$

Das condições do enunciado tem-se:

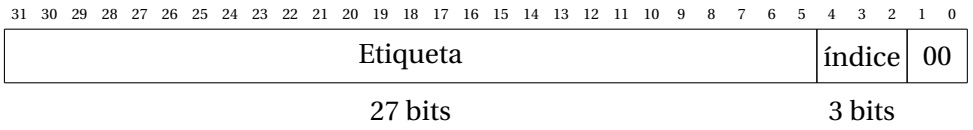
$$m_d \times 80 \leq 10 \qquad \Rightarrow \qquad m_d \leq= \frac{10}{80} = 0{,}125$$

Portanto, $m_d \leq 12{,}5\,\%$.

7. A tabela seguinte apresenta o conteúdo de uma memória *cache* do tipo *write-through* com 8 blocos de 4 bytes usada como *D-cache* num CPU com endereços de 32 bits.

	Conteúdo	Etiqueta	v
0	aa ff cc 33	123456a	0
1	12 34 56 78	7bcd001	1
2	88 b0 3c 2b	7fffd55	1
3	71 ab 3f 6d	7fffd55	1
4	34 ff 13 aa	07f9910	0
5	78 00 9c 23	0000893	1
6	7a 10 9f a3	2900002	1
7	99 43 65 b4	7f01d12	0

(a) Como é decomposto o endereço para acesso à memória *cache*? Justifique.



Offset: 2 bits.

Índice: 3 bits. Como a cache possui 8 blocos são necessários 3 bits para representar os índices de 0 a 7.

Etiqueta: 27 bits. Como temos 32 bits e já gastamos 5, os restantes $(32-5=27)$ serão para a etiqueta.

(b) Apresente, justificando, o conteúdo da memória *cache* após a execução das seguintes operações (se em algum caso não for possívelconhecer o conteúdo escreva "indeterminado"). Cada registo Rx tem 32 bits.

```
1: R1 ← 0xfafafafa
2: R4 ← R3+R2
3: R4 ← 0x0ff32210
4: R5 ← MEM[R4]
5: R5 ← R4-R1
6: R6 ← 0xffffaaa8
7: MEM[R6+4] ← R1
```

	Conteúdo	Etiqueta	v
0	aa ff cc 33	123456a	0
1	12 34 56 78	7bcd001	1
2	88 b0 3c 2b	7fffd55	1
3	fa fa fa fa	7fffd55	1
4	indeterminado	07f9910	1
5	78 00 9c 23	0000893	1
6	7a 10 9f a3	2900002	1
7	99 43 65 b4	7f01d12	0

As únicas instruções que acedem à memória de dados são:

- Instrução 4:
  Endereço `0x0ff32210 -> 00001111111100110010001000 | 100 | 00`
  Etiqueta: `0x07f9910`, bloco 4.
  Apesar de a etiqueta ser igual, tem-se $v=0$ *(read miss)*. É necessário ler o valor da memória principal e escrevê-lo na memória cache, colocando ainda $v=1$.
- Instrução 7:
  1º passo: Calcular o endereço: `0xffffaaa8+4=0xffffaaac`.
  2º passo: Obter etiqueta e bloco: `0xffffaaac` -> 11111111111111110101010101 | 011 | 00
  Etiqueta `0x7fffd55`, bloco 3.

> 3º passo: Verificar se é para alterar a memória *cache*.
> O bloco 3 tem etiqueta igual e $v = 1$ *(write hit)*. O valor de `t1` é escrito na memória principal e também na memória *cache*: o conteúdo do bloco 3 é alterado para `0xfafafafa`.

8. Um CPU tem endereços de 20 bits e usa uma memória *cache* com 1 palavra/bloco. A memória *cache* do tipo *write-back* usa 6 bits para cada índice e 12 bits para cada etiqueta. Uma parte do conteúdo da memória *cache* está indicada (em hexadecimal) na tabela seguinte:

	conteúdo	etiqueta	v	d
0	12345678	ABC	1	1
1	6548FEAB	123	0	0
2	3C1F56FD	678	1	0
3	AFD12498	567	1	1
4	6198FA34	B7C	1	0
5	1929AAAA	8D1	0	1
...	...	...	...	...

(a) Determinar o número de blocos e o número total de bits usados na memória *cache*.

> Como o número de bits do índice é 6, o número de blocos é $2^6 = 64$.
> Cada bloco tem 32 bits de dados, 12 bits de etiqueta, 1 bit de validade e 1 bit de atualização (d), o que dá um total de 46 bits por bloco.
> Logo, o número total de bits é $64 \times 46 = 2944$.

(b) Determinar, se possível, a posição em memória do valor `0x6198FA34`.

> O valor está no bloco 4 da memória *cache* com etiqueta *0xb7c*. Este valor é válido ($v = 1$) e igual ao valor em memória principal ($d = 0$).
> Portanto, o valor `0x6198fa34` está na posição `1011 0111 1100 | 0001 00 | 00`, i.e., `0xB7C10`.

(c) Determinar, se possível, qual o valor atualmente armazenado na posição de memória `0x5670C`.

> O endereço em binário é `0101 0110 0111 | 0000 11 |00`. Logo, se a memória *cache* contiver o conteúdo desejado será no bloco 3.
> O bloco 3 é válido porque $v = 1$ e a sua etiqueta `0x567` é idêntica à do endereço indicado.
> Contudo, como $d = 1$, o seu conteúdo é (potencialmente) diferente do valor atual ainda guardado em memória.
> Logo, não é possível determinar o valor ainda armazenado na memória principal (o valor em memória *cache* é mais recente)..

Fim.