

Ficha 4 – CPU com *pipelining*

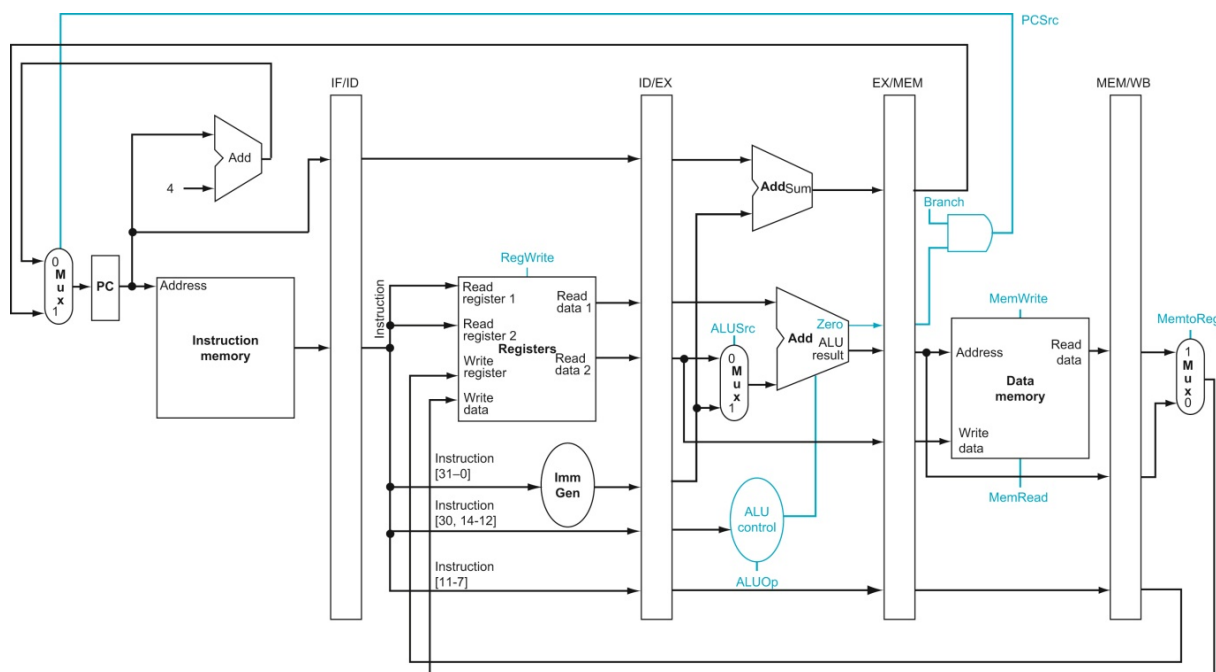


Figura 1: CPU com *pipelining*.

Exercícios resolvidos

1. Indique quais os componentes do caminho de dados do CPU que produzem valores úteis em cada etapa do processamento da instrução sw. Use a figura 1 como referência.

1. A instrução sw usa os componentes de cada andar da *pipeline* da seguinte forma:

- Andar 1 (IF): Todos os componentes são úteis. Neste andar nada depende da instrução, uma vez que esta ainda não é conhecida (está a ser obtida de memória).
- Andar 2 (ID): As operações deste andar também não dependem da instrução. Todos os componentes são úteis à exceção do porto de escrita (Write register e Write data) que não é usado nesta fase pela instrução.
- Andar 3 (EX): O componente Add não produz resultado útil, o mesmo acontecendo com a saída Zero da ALU. Os restantes componentes são úteis.
- Andar 4 (MEM): O porto de leitura da memória (Read data) não produz um valor útil. Os restantes componentes são úteis.
- Andar 5 (WB): O multiplexador controlado por MemtoReg não produz um resultado útil. O porto de escrita dos registos não está ativo (esta instrução não leva a uma alteração dos registos).

2. Indique os atalhos necessários para executar as sequências de instruções seguintes, de maneira a minimizar o número de ciclos de relógio.

a)

```
add s3, s4, s6
sub s5, s3, s2
lw s7, 100(s5)
add s8, s2, s7
```

b)

```
add s1, s1, s2
add s1, s1, s3
add s1, s1, s4
add s1, s1, s5
```

2. Existe uma dependência de dados entre as instruções A e B (B depende de A), quando A produz um valor que é usado por B. Existe um conflito entre as duas instruções se estas não puderem “partilhar” o caminho de dados por falta de recursos (conflito estrutural) ou se uma instrução dependente não puder obter atempadamente os valores corretos dos seus operandos.

a) A instrução sub depende da primeira instrução via s3. Este conflito pode ser resolvido pelo atalho ALU-ALU que encaminha a saída da ALU, armazenada em EX/MEM, para a entrada superior da ALU.

A instrução lw depende de sub via s5. Este conflito pode ser resolvido da mesma maneira que o anterior.

A última instrução (add) depende de lw via s7. Este conflito exige um ciclo de protelamento, porque lw apenas disponibiliza o valor a colocar em s7 no andar MEM. O protelamento é de apenas 1 ciclo: o atalho MEM-ALU leva o valor lido de memória, armazenado temporariamente em MEM/WB, para a entrada inferior da ALU (segundo operando de add). Se não existisse este atalho, seria necessário que o CPU protelasse durante 2 ciclos.

b) Cada instrução depende da precedente via s1. Todos os conflitos podem ser resolvidos pelo atalho entre a saída de EX/MEM e a entrada superior da ALU (primeiro operando de cada add).

3. A tabela mostra a latência de componentes do percurso de dados do CPU estudado nas aulas. Assuma que todos os outros elementos têm latência nula.

Unidade	I-Mem	Banco de registos	ALU	D-Mem
Tempo de atraso (ps)	200	100	200	200

a) Suponha que o tempo de execução da ALU é encurtado de 25%. Esta alteração afeta o desempenho da *pipeline*? Em caso afirmativo, determine a alteração de desempenho.

b) Responda à questão anterior, supondo que o tempo de execução da ALU aumenta 25%.

3.

- a) Cada um dos componentes determina o período mínimo necessário para funcionamento correto de cada um dos quatro andares iniciais da *pipeline*. O período mínimo do 5º andar (WB) é determinado pelo multiplexador MemtoReg, pelo que é 0.

Expressão do desempenho:

$$D = \frac{1}{t_{exec}} = \frac{1}{N \times CPI \times T}$$

A alteração indicada não afeta o valor de N nem de CPI, pelo que eventualmente o período T é a única grandeza a alterar neste caso. T deve ser igual, ou superior, ao máximo dos períodos mínimos de cada andar. Portanto, o melhor valor é $T_a = 200$ ps.

Encurtar o tempo de execução da ALU não melhora o valor de T_a , porque este continua a ser de 200 ps, conforme imposto pelos componentes I-Mem e D-Mem.

O aumento de rapidez (*speedup*) é 1 (i.e., não existe alteração).

- b) Neste caso, a ALU passa a ser o componente mais demorado (250 ps), aumentando o período do sinal de relógio do CPU que passa a ser $T_b = 250$ ps.

O "aumento" de rapidez (*speedup*) é inferior a 1, porque o desempenho piora:

$$s = \frac{D_b}{D_a} = \frac{N \times CPI \times T_a}{N \times CPI \times T_b} = \frac{200}{250} = 0,8$$

4. Um engenheiro de computadores necessita de projetar um caminho de dados para um novo processador. Para a avaliação do desempenho usa um programa de aferição com 10^6 instruções. Cada instrução demora 100 ps.

- a) Determine quanto tempo demora a execução do programa num processador sem *pipelining*.
- b) Um CPU atual tem cerca de 20 andares de *pipeline*. Assumindo que a *pipeline* funciona em condições ideais, determine a melhoria de desempenho que pode ser obtida face à situação da alínea anterior.
- c) Na realidade, a implementação de *pipelines* introduz sempre algum *overhead* por andar. Este *overhead* afeta a latência ou o débito das instruções (ou ambos)? Porquê?

4.

- a) Basta multiplicar o número de instruções pelo tempo de execução de cada uma:

$$100 \text{ ps} \times 10^6 = 100 \mu\text{s}$$

- b) Em circunstâncias ideais, o período de relógio vem reduzido de um fator de 20, i.e., existe um aumento de rapidez (*speedup*) de 20. O novo tempo de execução é:

$$\frac{100 \mu\text{s}}{20} = 5 \mu\text{s}$$

- c) Se na situação ideal, o período for T , na situação mais realista será $T + t_{\text{overhead}}$.

A latência L da instrução é o tempo que demora a ser completada (20 ciclos). No caso ideal:

$$L_i = 20 \times T$$

No caso mais realista tem-se:

$$L_r = 20 \times (T + t_{\text{overhead}}) = 20 \times T + 20 \times t_{\text{overhead}} = L_i + 20 \times t_{\text{overhead}}$$

O débito d é definido pelo número de instruções terminadas por unidade de tempo. Assumindo que $\text{CPI}=1$, o débito ideal é:

$$d_i = \frac{1}{T}$$

O débito com *overhead* será de:

$$d_r = \frac{1}{T + t_{\text{overhead}}} < \frac{1}{T} = d_i \rightarrow d_r < d_i$$

Tanto a latência como o débito são afetados negativamente pelo *overhead*. Contudo, a latência é mais afetada (fator de 20 associado a t_{overhead}).

5. Assuma a seguinte distribuição de instruções a serem executadas no CPU com *pipeline*.

add	beq	lw	sw
50%	25%	15%	10%

Assuma também que não existem conflitos ou protelamentos ($\text{CPI}=1$).

- Determine a taxa de utilização da memória de dados (i.e., a percentagem de ciclos em que é usada).
- Determine a taxa de utilização do porto de escrita do banco de registos.

5.

- A memória de dados é usada apenas pelas instruções *lw* e *sw*. Logo, a respetiva taxa de utilização é $15\%+10\%=25\%$.
- O porto de escrita é usado pelas instruções que modificam um registo, isto é, pelas instruções *add* e *lw*. Portanto, a taxa de utilização é $50\%+15\%=65\%$.

6. Num determinado programa executado no CPU de referência (RISC-V), as instruções de salto condicional têm uma taxa de ocorrência de 15%. Na situação ideal, o processador tem um $\text{CPI}=1$. Sabendo que a previsão de saltos está correta 75% das vezes, determine o CPI efetivo.

6. É necessário calcular a média pesada do CPI associado a cada situação possível.

- Instrução não é de salto condicional: 85%.
- Instrução é de salto condicional e a previsão é correta: $15\% \times 75\%$.
- Instrução é de salto condicional e a previsão é incorreta: $15\% \times 25\%$.

A situação mais penalizadora corresponde à ocorrência de salto condicional com previsão incorreta. Para esta ocorrência o número de ciclos é 4. Note-se que, por omissão, é na etapa MEM que fica a conhecer-se se a condição de salto é verdadeira ou falsa. Caso a avaliação da condição fosse antecipada para a etapa ID, então o número de ciclos era 2 (em vez de 4).

Conclusão: $CPI_{\text{médio}} = 1 \times 0,85 + 1 \times 0,75 \times 0,15 + 4 \times 0,25 \times 0,15 = 1,1125$.

7. Considere o seguinte fragmento de código *assembly* a ser executado num processador RISC-V com *pipeline* e atalhos, estudado nas aulas. O processador suporta prognóstico de saltos condicionais e avalia a condição de salto no andar ID.

```
lw    s2, 0(s3)
add   t0, s2, t0
andi  t6, s2, 1
add   t2, t2, t6
sub   t1, t1, s2
addi  s3, s3, 4
```

- Indique todas as dependências de dados existentes no fragmento.
- Para cada dependência, indique se provoca um conflito e, em caso afirmativo, como é que ele é resolvido: por protelamento e/ou por utilização de atalhos. Especifique claramente todos os atalhos envolvidos.
- Determine o CPI deste fragmento.
- O fragmento de código é incluído num ciclo, conforme se indica a seguir:

```
L: lw    s2, 0(s3)
    add   t0, s2, t0
    andi  t6, s2, 1
    add   t2, t2, t6
    sub   t1, t1, s2
    addi  s3, s3, 4
    bne   s3, s0, L
```

Considere que são executadas três iterações deste novo fragmento, ou seja, na terceira execução da instrução *bne* o salto não é tomado. Determine o seu novo CPI.

7.

- add* (linha 2) depende de *lw* via *s2*
 - andi* depende de *lw* via *s2*
 - add* (linha 4) depende de *andi* via *t6*
 - sub* depende de *lw* via *s2*
- causa protelamento (1 ciclo); usa atalho MEM→ALU (entrada superior);
 - sem protelamento; não necessita de atalho;
 - sem protelamento; usa atalho ALU→ALU (entrada inferior);
 - sem protelamento; não necessita de atalho.

c) Número de instruções é 6.

A primeira instrução (lw) termina após 5 ciclos, a segunda (add) é protelada 1 ciclo, necessitando de mais um ciclo para concluir a execução de cada uma das 5 instruções após lw. O número de ciclos é pois $5 + 1 + 5 = 11$.

$$\text{CPI} = 11/6 = 1,833$$

d) Após lw ocorre um protelamento. Como a decisão da condição de salto é conhecida na etapa ID, existe um (apenas) protelamento adicional sempre que o salto bne é tomado. Portanto, ocorrem dois protelamentos por iteração, exceto na última em que só ocorre um (imposto por lw).

Número de ciclos em cada iteração:

- 1ª iteração: $5 + 1 + 6 + 1 = 13$
- 2ª iteração: $1 + 1 + 6 + 1 = 9$
- 3ª iteração: $1 + 1 + 6 = 8$

Número de ciclos para a execução completa é $13 + 9 + 8 = 30$.

Número de instruções executadas é $3 \times 7 = 21$.

Logo, $\text{CPI} = 30/21 = 10/7 \approx 1,43$.

Exercícios propostos

8. Indique quais os componentes do caminho de dados do CPU que produzem valores úteis em cada etapa do processamento da instrução `lw`. Use a figura 1 como referência.

9. Considere a execução do seguinte fragmento de código numa versão do CPU que não deteta conflitos de dados na *pipeline*, obrigando o programador a resolver tais conflitos por protelamento, ou seja, inserindo instruções `nop` (*no operation*). `NOP` executa sem alterar nada no CPU (`addi x0, x0, 0`).

```
addi x11, x12, 5
add  x13, x11, x12
addi x14, x11, 15
add  x15, x11, x11
```

O banco de registos é escrito no início do ciclo e lido no final do mesmo ciclo. Portanto, na etapa ID está acessível o resultado da etapa WB que ocorra durante o mesmo ciclo.

Determine o valor final de `x15` assumindo que os valores iniciais de `x11` e `x12` são, respetivamente, 11 e 22.

10. Identifique todas as dependências de dados da sequência de código apresentada a seguir. Indique que conflitos podem ser resolvidos por atalhos e quais provocam protelamentos.

```
add  s3, s4, s2
sub  s5, s3, s1
lw   s6, 200(s3)
add  s7, s3, s6
```

11. Uma sub-rotina com 10^3 instruções tem o seguinte formato: "`lw, add, lw, add, . . .`". A instrução de adição depende apenas da instrução de *load* imediatamente anterior. Cada instrução de *load* depende apenas da instrução de adição imediatamente anterior.

Considerando que o programa deve ser executado pelo CPU com *pipelining* apresentado nas aulas, determine:

- CPI desta sub-rotina se executada com atalhos.
- CPI desta sub-rotina na ausência de atalhos.

12. Assuma que os andares do caminho de dados têm as seguintes latências:

IF	ID	EX	MEM	WB
300 ps	400 ps	350 ps	500 ps	100 ps

- Qual é o período mínimo de relógio para um caminho de dados sem e com *pipelining*?
- Qual é a latência total da instrução `lw` em processadores sem e com *pipelining*?
- Suponha que era possível dividir andares de *pipeline* em dois novos andares, cada um com metade da latência do andar original. Que andar deveria ser dividido e qual seria o período de relógio correspondente?

13. Considere a seguinte sequência de instruções:

```
lw    t1, 40(t6)
add   t6, t2, t2
sw    t6, 50(t1)
```

- Indique todas as dependências existentes.
- Assuma que o CPU (com *pipelining*) não implementa atalhos. Indique os conflitos existentes e onde devem ser acrescentadas instruções *nop* (*no operation*) para os evitar.
- Assuma que o CPU implementa todos os atalhos (ALU-ALU e MEM-ALU). Indique os conflitos existentes e onde devem ser acrescentadas instruções *nop* para os evitar.

Assuma de seguida que o período de relógio das diferentes implementações do CPU é:

Sem atalhos	Com todos os atalhos	Apenas com atalhos ALU-ALU
300 ps	400 ps	360 ps

- Determine o tempo total necessário para executar corretamente a sequência de instruções indicada com atalhos completos e sem atalhos. Determine o aumento de rapidez obtido pela implementação com atalhos.
- Acrescente instruções *nop* à sequência para evitar conflitos na implementação que dispõe apenas de atalhos ALU-ALU.
- Determine o tempo total necessário para executar corretamente a sequência de instruções numa implementação que tenha apenas atalhos ALU-ALU. Determine o aumento de rapidez obtido face a uma implementação sem atalhos.

14. Considere a instrução `lw t2, 40(t6)`. Note que o registo t2 é x7 e o registo t6 é x31. Assumir que o conteúdo de t6 é 200.

- Indique os valores de cada registo da *pipeline* ao longo da execução da instrução.
- Quais são as atividades associadas à execução desta instrução nos andares EX e MEM?

15. Considere a seguinte sequência de instruções:

```
loop: lw    t1, 40(t6)
      add   t5, t5, t3
      add   t6, t6, t3
      sw    t1, 20(t5)
      beq   t1, zero, loop
```

Assuma que os prognósticos de salto são sempre corretos, ou seja, saltos para trás são sempre tomados (exceto na última iteração). Assuma também que o processador contém todos os atalhos estudados e que um elevado número de iterações é realizado antes de a execução do ciclo terminar.

- Apresente um diagrama da execução da *pipeline* para a terceira iteração deste ciclo. Indique todas as instruções presentes na *pipeline*, mesmo as que não pertencem a essa iteração.
- No início do ciclo em que o CPU obtém a primeira instrução da terceira iteração, qual é o conteúdo do registo IF/ID?
- Determine a percentagem de ciclos em que todos os andares executam trabalho útil.

16. Considere o seguinte fragmento de código a ser executado pelo CPU com *pipelining* estudado nas aulas.

```
add    t2, s1, s2
addi   t3, t2, -18
lw     t5, 12(t2)
or     t5, t5, t3
lw     s1, 0(t3)
```

Assuma que o CPU implementa todos os atalhos: ALU-ALU e MEM-ALU.

- Indique, justificando, as dependências de dados existentes e como podem ser evitados os correspondentes conflitos de forma a minimizar o tempo de execução.
- Assuma que os andares do CPU têm as seguintes latências:

IF	ID	EX	MEM	WB
300 ps	400 ps	350 ps	500 ps	100 ps

Determine o período mínimo do sinal de relógio. Determine também o tempo de execução do fragmento de código nas condições da alínea anterior.

- Considere que o fragmento de código é executado 10^6 vezes seguidas. Determine o respetivo CPI.

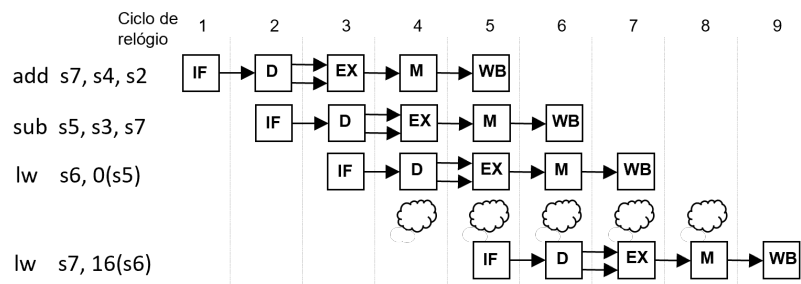
17. Considere o seguinte fragmento de código *assembly* RISC-V.

```
sw     x29, 12(x16)
lw     x29, 8(x16)
sub    x17, x15, x14
beq    x17, x0, L
add    x15, x11, x14
sub    x15, x30, x14
```

Suponha que a *pipeline* do CPU é alterada passando a ter apenas uma memória, armazenando instruções e dados. Como consequência, ocorrerá um conflito estrutural sempre que num programa seja extraída uma instrução (etapa IF) durante o mesmo ciclo no qual outra instrução acede a dados (etapa MEM).

- Desenhe um diagrama da *pipeline* que permita identificar os protelamentos necessários à resolução dos conflitos estruturais.
- É em geral possível reduzir o número de protelamentos/NOPs resultantes deste conflito estrutural por reordenação do código?
- Este conflito estrutural deve ser tratado em *hardware*? Nós temos visto que os conflitos de dados podem ser eliminados pela inserção de NOPs no código. Pode fazer-se o mesmo em relação a este conflito estrutural? Explique.
- Num programa típico quantos protelamentos ocorrerão para resolver este tipo de conflito? Como exemplo, considere as seguintes taxas de ocorrência de instruções: tipo-R \rightarrow 52%, lw \rightarrow 25%, sw \rightarrow 11% e beq \rightarrow 12%.

18. Considere o seguinte diagrama de evolução temporal resultante da execução de um fragmento de código no CPU com *pipeline* e atalhos estudado nas aulas.

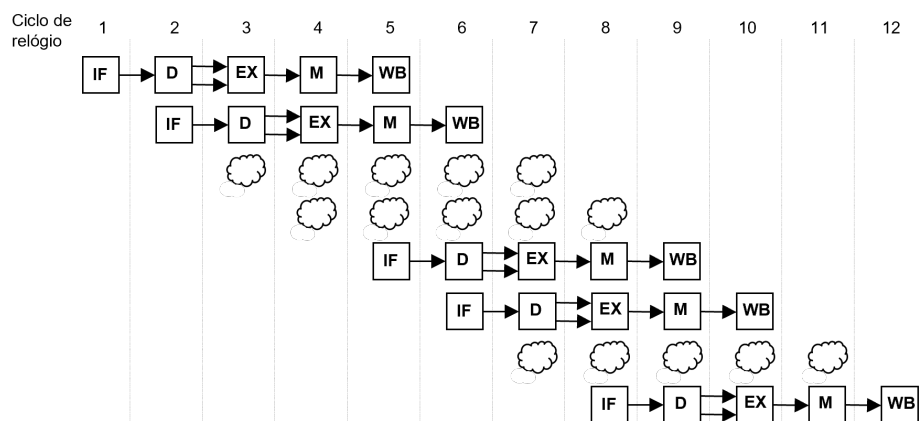


- Indique, justificando, que atalhos são usados e entre que instruções são utilizados.
- Determine quantos ciclos de relógio seriam necessários para executar corretamente o fragmento de código se o CPU não implementasse atalhos.
- Considere que o CPU apenas implementa atalhos ALU-ALU. Indique se existe aumento de rapidez face à implementação sem atalhos, considerando que o período de relógio no CPU sem atalhos é 80% do período de um CPU com apenas atalhos ALU-ALU.

19. A sequência de instruções incompletas apresentada a seguir foi executada numa versão do processador com *pipeline* que não implementa atalhos.

- 1 `lw t6, 0(t1)`
- 2 `lw ____, 4(t1)`
- 3 `add ____, ____, ____`
- 4 `sw ____, 4(____)`
- 5 `sub ____, ____, ____`

A figura mostra o diagrama temporal correspondente.



- Atendendo aos dados, complete as instruções com os argumentos em falta, fundamentando as suas escolhas.
- Indique que instruções estão a ser executadas durante o 9º ciclo de relógio e explique as operações em curso no caminho de dados (*pipeline*) do processador.
- Supondo agora que o processador implementa atalhos, redesenhe o diagrama temporal evidenciando os atalhos utilizados.
- Assumindo que o período do relógio é o mesmo na versão com atalhos e na versão sem atalhos, calcule o aumento de rapidez conseguido pela versão com atalhos.

20. O seguinte código *assembly* RISC-V permite calcular a soma de 50 valores inteiros de uma sequência armazenada em memória.

```
sub    t0, t0, t0
addi   t1, zero, 50
loop:  lw    t2, 0(a0)
add    t0, t0, t2
addi   a0, a0, 4
addi   t1, t1, -1
bne    t1, zero, loop
add    a0, zero, t0
```

Considere que o código é executado no CPU com *pipeline*. Tenha presente que, por omissão, não é usada predição e que a decisão de salto (tomado/não tomado) é conhecida na etapa 4 (MEM).

- a) Determine o número de ciclos necessários para executar o código assumindo que o CPU não implementa atalhos.
- b) Determine o número de ciclos necessários para executar o código assumindo que o CPU implementa todos os atalhos.
- c) Considere que o CPU implementa todos os atalhos e que a decisão de salto é conhecida na etapa ID. Determine o número de ciclos necessários para executar o código.
- d) Considerando que o CPU implementa todos os atalhos, verifique se é possível reordenar instruções sem alterar a funcionalidade pretendida. Sendo possível, indique alterações de ordem de instruções que permitam reduzir o número de ciclos de relógio necessários à execução do código.
- e) Assuma as mesmas condições da alínea anterior e que a predição de saltos determina que os saltos para trás são tomados (técnica que acerta sempre exceto na última iteração). Determine o número de ciclos necessários para executar o código.

Fim do enunciado

Nota: Alguns dos exercícios foram extraídos ou adaptados do livro “Computer Organization and Design – The Hardware/-Software Interface”, Hennessy & Patterson, 4ª edição.

Soluções:

8. Andar 1 (IF): todos.

Andar 2 (ID): todos (porto de escrita não é usado).

Andar 3 (EX): úteis – MUX, ALU, ALU control; inútil – Add.

Andar 4 (MEM): úteis – D-Mem; inúteis – And.

Andar 5 (WB): úteis – Mux e porto de escrita dos registos (Write register, Write data).

9. $x15 = 54$

10. Existem dependências de dados entre as três últimas instruções e a primeira, via s3: podem ser resolvidas com atalhos. A última instrução também depende da instrução lw via s6: causa protelamento.

11.

a) $CPI \approx 6/4 = 1,5$ (protelamento de 1 ciclo entre lw e add).

b) $CPI \approx 12/4 = 3$ (protelamento de 2 ciclos entre lw e add, mais protelamento de 2 ciclos entre add e lw seguinte).

12.

a) Sem *pipelining*: 1650 ps. Com *pipelining*: 500 ps.

b) Sem *pipelining*: 1650 ps. Com *pipelining*: $5 \times 500 = 2500$ ps.

c) Dividir o andar MEM. $T_{clk} = 400$ ps.

13.

a) Instrução sw depende de lw via t1. Instrução sw depende de add via t6.

b) As duas dependências dão origem a conflitos. Duas instruções nop devem ser colocadas entre add e sw.

c) Os atalhos resolvem os conflitos, não sendo necessário usar instruções nop.

d) Com atalhos: $t_{exec} = 7 \times 400 = 2800$ ps. Sem atalhos: $t_{exec} = 9 \times 300 = 2700$ ps. $Speedup=0,96$.

e) Acrescentar duas instruções nop entre add e sw.

f) Com atalhos ALU-ALU: $t_{exec} = 3240$ ps. Sem atalhos: $t_{exec} = 2700$ ps. $Speedup=0,83$.

14.

a) O conteúdo de cada registo é o seguinte:

IF/ID: PC; código da instrução = $0x028FA303$.

ID/EX: PC; Read data 1 = 200; Read data 2 = ?; registo de escrita (destino t2) = 7; Imm = 40; sinais de controlo: ALUSrc=1, ALUOp=00, Branch=0, MemRead=1, MemWrite=0, MemtoReg=1, RegWrite=1.

EX/MEM: sinal Add sum = PC+40; Zero=0; Read data 2 = ?; resultado ALU = 240; registo de escrita (destino t2) = 7; sinais de controlo: Branch=0, MemRead=1, MemWrite=0, MemtoReg=1, RegWrite=1.

MEM/WB: Read data = valor da memória (endereço 240); registo de escrita (destino) = 7; sinais de controlo: MemtoReg=1, RegWrite=1.

- b) Em EX é calculado o endereço de memória. Em MEM é feita a leitura da memória de dados.

15.

a)

iter.	instr.	atividade da <i>pipeline</i>				
2	add	WB				
2	add	(MEM)	WB			
2	sw	EX	MEM	(WB)		
2	beq	ID	EX	MEM	(WB)	
3	lw	IF	ID	EX	MEM	WB
3	add		IF	ID	EX	(MEM)
3	add			IF	ID	EX
3	sw				IF	ID
3	beq					IF

Obs.: Andares em () não realizam trabalho útil.

- b) IF/ID contém o código da instrução anterior (beq) e o respetivo endereço.
c) 20%.

16.

- a)
- Instrução 1 e 2: dependência via t2 resolvida com atalho ALU-ALU;
 - Instrução 1 e 3: dependência via t2 resolvida com atalho MEM-ALU;
 - Instrução 3 e 4: dependência via t5 resolvida com um protelamento e atalho MEM-ALU;
 - Instrução 2 e 4: dependência via t3 não causa conflito (devido ao protelamento introduzido entre as instruções 3 e 4);
 - Instrução 2 e 5: dependência via t3, não provoca conflito.

- b) Período mínimo = 500 ps.

Número de ciclos = 10 (decorrente da alínea anterior).

$$t_{\text{exec}} = N_{\text{ciclos}} \times T = 10 \times 500 = 5 \text{ ns}$$

- c) Existem 2 protelamentos por cada execução do fragmento de código, um entre lw e or e outro entre lw (última linha) e add do fragmento seguinte.

$$\text{CPI} = \frac{N_{\text{ciclos}}}{N_{\text{efetivo de instr.}}} \approx \frac{7 \times 10^6}{5 \times 10^6} = 1,4$$

17.

a)

instrução	atividade da <i>pipeline</i>									
sw x29, 12(x16)	IF	ID	EX	MEM	WB					
lw x29, 8(x16)		IF	ID	EX	MEM	WB				
sub x17, x15, x14			IF	ID	EX	MEM	WB			
beq x17, zero, L				**	**	IF	ID	EX	MEM	WB
add x15, x11, x14							IF	ID	EX	MEM
sub x15, x30, x14								IF	ID	EX
									MEM	WB

Obs.: Protelamentos assinalados com **.

- b) Reordenar o código não resolve o conflito. Todas as instruções são extraídas de memória, pelo que todos os acessos a dados provocam um protelamento. A reordenação de código apenas alterará o par de instruções que estão em conflito.
- c) Não se pode resolver este risco estrutural com NOPs, porque mesmo os NOPs devem ser obtidos a partir da memória de instruções.
- d) 36% (todos os acessos a dados causam um protelamento).

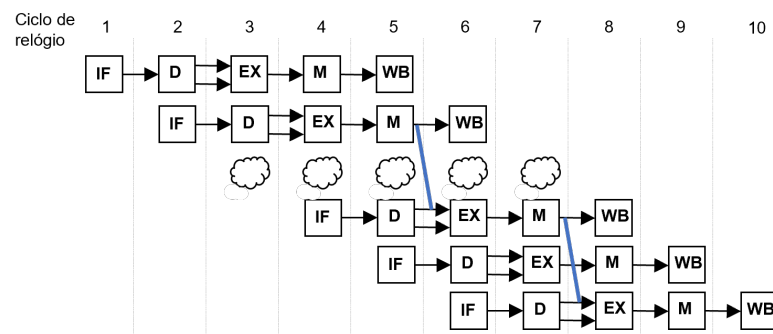
18.

- a) Instrução 1 e 2: atalho ALU-ALU.
Instrução 2 e 3: atalho ALU-ALU.
Instrução 3 e 4: atalho MEM-ALU.
- b) 14 ciclos de relógio (são necessários 2 protelamentos entre cada instrução).
- c) Apenas com atalhos ALU-ALU: $5 + 1 + 1 + 3 = 10$ ciclos de relógio.

Aumento de rapidez: $\frac{14 \times 0,8}{10} = 1,12$

19.

- a) Uma solução possível:
2: t5
3: t0, t5, t6
4: t6, t1
5: t0, t0, t5
- b) add (etapa WB), sw (etapa MEM) e sub (etapa ID).
- c)



- d) 1,2

20.

- a) 604
- b) 454
- c) 456
- d) 404
- e) 260