INE5441 - Compêndio de Avaliações de 2020

Questão 1 - P1

Mostre a sequência de instruções nativas do MIPS32 que implementa o procedimento abaixo (escrito em linguagem C), usando a convenção de chamadas.

```
int procedimento (int a, int b)
{
        return((a%b)*4)
}
```

Procedimento:

```
div     $a0, $a1
mfhi    $v0
sll     $v0, $v0, 2
jr      $ra
```

```
int64_t procedimento (int a, int b)
{
        return(a*b)
}
```

Procedimento:

```
mult    $a0, $a1
mfhi    $v0
mflo    $v1
jr      $ra
```

Questão 2 - P1 & P2

Suponha que as seguintes modificações tenham sido feitas na especificação original do MIPS32, cujos formatos de instrução estão descritos no MIPS Reference Data, disponível na penúltima página (ES81) da versão eletrônica do livro-texto:

· **Extensão do número de registradores:** de 32 para 128;
· **Extensão do comprimento dos registradores:** de 32 para 128 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas de forma que o número de instruções do tipo R seja o dobro do original;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.

Nestas condições, quais das seguintes afirmações são verdadeiras?

O número de bits do campo opcode é 7.
O número de bits do campo shamt é 7.
O número de bits do campo funct é 7.
O número de bits total da instrução no formato R é 42.
O número de bits do campo immediate é 21.

...
· **Extensão do número de registradores:** de 32 para 128;
· **Extensão do comprimento dos registradores:** de 32 para 128 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas preservandoo número original de instruções do tipo R;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 8.
O número de bits do campo shamt é 7.
O número de bits do campo funct é 6.
O número de bits total da instrução no formato R é 42.
O número de bits do campo immediate é 20.

...
· **Extensão do número de registradores:** de 32 para 128;
· **Extensão do comprimento dos registradores:** de 32 para 128 bits;
· **Extensão da ISA:** o número de instruções do tipo R foi quadriplicado;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 6.
O número de bits do campo shamt é 6.
O número de bits do campo funct é 8.
O número de bits total da instrução no formato R é 41.
O número de bits do campo immediate é 21.

...
· **Extensão do número de registradores:** de 32 para 128;
· **Extensão do comprimento dos registradores:** de 32 para 64 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas preservando o número original de instruções do tipo R;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 8.
O número de bits do campo shamt é 6.
O número de bits do campo funct é 6.
O número de bits total da instrução no formato R é 41.
O número de bits do campo immediate é 19.

...
· **Extensão do número de registradores:** de 32 para 128;
· **Extensão do comprimento dos registradores:** de 32 para 64 bits;
· **Extensão da ISA:** o número de instruções do tipo R foi quadriplicado;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 6.
O número de bits do campo shamt é 6.
O número de bits do campo funct é 8.
O número de bits total da instrução no formato R é 41.
O número de bits do campo immediate é 21.

...
· **Extensão do número de registradores:** de 32 para 64;
· **Extensão do comprimento dos registradores:** de 32 para 128 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas preservando o número original de instruções do tipo R;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 8.
O número de bits do campo shamt é 7.
O número de bits do campo funct é 6.
O número de bits total da instrução no formato R é 39.
O número de bits do campo immediate é 19.

...
· **Extensão do número de registradores:** de 32 para 64;
· **Extensão do comprimento dos registradores:** de 32 para 64 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas de forma que o número de instruções do tipo R seja o dobro do original;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 7.
O número de bits do campo shamt é 6.
O número de bits do campo funct é 7.
O número de bits total da instrução no formato R é 38.
O número de bits do campo immediate é 19.

...
· **Extensão do número de registradores:** de 32 para 64;
· **Extensão do comprimento dos registradores:** de 32 para 64 bits;
· **Extensão da ISA:** o número total de instruções foi quadriplicado, mas preservando o número original de instruções do tipo R;;
· **Comprimento do Formato I ditado pelo comprimento do Formato R:** O número total de bits do Formato R estendido foi adotado para o Formato I.
...

O número de bits do campo opcode é 8.
O número de bits do campo shamt é 6.
O número de bits do campo funct é 6.
O número de bits total da instrução no formato R é 38.
O número de bits do campo immediate é 18.

Questão 3 - P1

Os processadores X e Y são ambos compatíveis com ARMv8. Suponha que ummesmo programa escrito na linguagem C foi compilado para executar em X e Y,usando o mesmo compilador e exatamente as mesmas opções (flags) de compilação. Sabe-se que as frequências de X e Y são, respectivamente, 1GHz e 2GHz. Nestas condições, qual(is) das seguintes afirmações é (são) verdadeira(s)?

O número de instruções executadas é o mesmo em X e Y.
Nada se pode afirmar sobre o número de instruções por segundo.
O número de ciclos por segundo é 2 vezes maior em Y do que em X.
Nada se pode afirmar sobre o número de ciclos por instrução

O processadores X e Y são compatíveis com ARMv7 e ARMv8, respectivamente. Suponha que um mesmo programa escrito na linguagem C foi compilado para executar em X e Y, usando o mesmo compilador e exatamente as mesmas opções (flags) de compilação. Sabe-se que as frequências de X e Y são, respectivamente, 500MHz e 1GHz. Nestas condições, qual(is) das seguintes afirmações é (são)verdadeira(s)

Nada se pode afirmar sobre o número de instruções executadas.
Nada se pode afirmar sobre o número de instruções por segundo.
O número de ciclos por segundo é 2 vezes maior em Y do que em X.
Nada se pode afirmar sobre o número de ciclos por instrução.

O processadores X e Y são compatíveis com ARMv8 e x86-64, respectivamente. Suponha que um mesmo programa escrito na linguagem C foi compilado para executar em X e Y, usando o mesmo compilador e exatamente as mesmas opções (flags) de compilação. Sabe-se que as frequências de X e Y são, respectivamente, 500 MHz e 1 GHz. Nestas condições, qual(is) das seguintes afirmações é (são) verdadeira(s)?

O número de instruções executadas é o mesmo em X e Y.
Nada se pode afirmar sobre o número de instruções por segundo.
O número de ciclos por segundo é 2 vezes maior em Y do que em X.
Nada se pode afirmar sobre o número de ciclos por instrução.

O processadores X e Y são compatíveis com ARMv8 e x86-64, respectivamente. Suponha que um mesmo programa escrito na linguagem C foi compilado para executar em X e Y, usando o mesmo compilador e exatamente as mesmas opções (flags) de compilação. Sabe-se que as frequências de X e Y são, respectivamente, 2GHz e 4GHz. Nestas condições, qual(is) das seguintes afirmações é (são) verdadeira(s)?

Nada se pode afirmar sobre o número de instruções executadas.
Nada se pode afirmar sobre o número de instruções por segundo.
O número de ciclos por segundo é 2 vezes maior em Y do que em X.
Nada se pode afirmar sobre o número de ciclos por instrução

Questão 4 - P1

Suponha que se pretenda implementar a função `lock (char *sem)` para disponibilizá-la em uma biblioteca de sincronização compatível com o MIPS32, onde o parâmetro **sem** representa o endereço de memória associado a um semáforo binário. Uma programadora e um programador desenvolveram independentemente os códigos A e B abaixo para implementar aquela função. Nestas condições, qual(is) das seguintes afirmações é (são) verdadeira(s)?

```
                A                              B
lock:  addi  $t0,$zero,1      lock  ll    $t1,0($a0)
       ll    $t1,0($a0)             beq   $t1,$zero,lock
       sc    $t0,0($a0)             addi  $t0,$zero,1
       beq   $t0,$zero,lock         sc    $t0,0($a0)
       bne   $t1,$zero,lock         bne   $t0,$zero,lock
       jr    $ra                    jr    $ra
```

O código **A** implementa corretamente a função.

...

```
                A                              B
lock:  addi  $t0,$zero,1      lock  ll    $t1,0($a0)
       ll    $t1,0($a0)             bne   $t1,$zero,lock
       sc    $t0,0($a0)             addi  $t0,$zero,1
       bne   $t0,$zero,lock         sc    $t0,0($a0)
       beq   $t1,$zero,lock         beq   $t0,$zero,lock
       jr    $ra                    jr    $ra
```

O código **B** implementa corretamente a função.

...

```
                A                              B
lock:  addi  $t0,$zero,1      lock  ll    $t1,0($a0)
       ll    $t1,0($a0)             beq   $t1,$zero,lock
       sc    $t0,0($a0)             addi  $t0,$zero,1
       bne   $t0,$zero,lock         sc    $t0,0($a0)
       beq   $t1,$zero,lock         bne   $t0,$zero,lock
       jr    $ra                    jr    $ra
```

Nenhum dos códigos implementa corretamente a função.

Questão 5 - P1

A instrução **beq $s1, $s2, label** reside no endereço `0FF8 0018`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FF6 001C.

A instrução **beq $s1, $s2, label** reside no endereço `0FF9 0014`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FF7 0018.

A instrução **beq $s1, $s2, label** reside no endereço `0FFA 0010`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FF8 0014.

A instrução **beq $s1, $s2, label** reside no endereço `0FFB 000C`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FF9 0010.

A instrução **beq $s1, $s2, label** reside no endereço `0FFC 0008`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FFA 000C.

A instrução **beq $s1, $s2, label** reside no endereço `0FFE 0004`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FFC 0008.

A instrução **beq $s1, $s2, label** reside no endereço `0FFF 0000`. Para o MIPS32, qual o endereço-efetivo **mínimo** atingível por esta instrução?

A resposta correta é: 0x 0FFD 0004.

Questão 6 - P1

Em linguagens orientadas a objeto, os métodos são chamados através de uma tabela, denominada tabela de métodos, que contém os endereços (M0, M1, M2, ...) onde residem os métodos (método 0, método 1, método 2, ...) de um objeto (ela é similar à jump address table que você aprendeu no Lab 2). No código abaixo, suponha que o endereço-base da tabela de métodos esteja representado pelo rótulo met. Complete o código de forma que o **método n** seja chamado. Consulte o Apêndice A.10 do livro-texto para entender como funciona a instrução `jalr`, que aparece no final do código.

```
      .data
met:  .word M0, M1, M2, M3, M4, M5, M6, M7, M8, M9...
      .text
      .globl main
main:
      ...
```

Método 3

```
la      $s0, met
lw      $t0, 12($s0)
jalr    $t0
```

Método 4

```
la      $s0, met
lw      $t0, 16($s0)
jalr    $t0
```

Método 5

```
la      $s0, met
lw      $t0, 20($s0)
jalr    $t0
```

Método 6

```
la      $s0, met
lw      $t0, 24($s0)
jalr    $t0
```

Questão 7 - P1 & P2

As duas linhas de código executável abaixo correspondem a uma sequência de duas instruções nativas do MIPS32 (representadas em hexadecimal) que implementam uma pseudoinstrução suportada pelo montador. Consulte o manual de referência e deduza que pseudoinstrução é essa.

```
0x 3C01 1001
0X 8C29 0000
```

A pseudoinstrução é `lw $t1, FACE2020` ($t1 ← MEM[endereço correspondente a FACE2020]). [ERRO NA QUESTÃO]

```
0x 3C01 FACE
0X 3429 0000
```

A pseudoinstrução é `la $t1, FACE2020` ($t1 ← endereço correspondente a FACE2020) `lt $t1, $t2, FACE2020` (se $t1 < $t2, desvie para FACE2020).

```
0x 3C01 FACE
0X 8C29 2020
```

A pseudoinstrução é `lw $t1, 0xFACE2020` ($t1 ← MEM[`0xFACE2020`]).

```
0x 0149 082A
0X 1020 000F
```

A pseudoinstrução é `ble $t1, $t2, FACE2020` (se $t1 ≤ $t2, desvie para FACE2020).

```
0x 0149 082A
0X 1420 000D
```

A pseudoinstrução é `bgt $t1, $t2, FACE2020` (se $t1 > $t2, desvie para FACE2020).

```
0x 012A 082A
0X 1020 000B
```

A pseudoinstrução é `bge $t1, $t2, FACE2020` (se $t1 ≥ $t2, desvie para FACE2020).

```
0x 012A 082A
0X 1420 0011
```

A pseudoinstrução é `blt $t1, $t2, FACE2020` (se $t1 < $t2, desvie para FACE2020).

Questão 8 - P1

Uma instrução j label reside no endereço **0xF000 00C4**. Para o MIPS32, quais os endereços-efetivos **mínimo** e **máximo** atingíveis por esta instrução?

[min, max] = [[0xF000 0000, 0xFFFF FFFC]]

Uma instrução j label reside no endereço **0x0000 00C4**. Para o MIPS32, quais os endereços-efetivos **mínimo** e **máximo** atingíveis por esta instrução?

[min, max] = [[0x0000 0000, 0x0FFF FFFC]]

Questão 9 - P1 & P2

O código abaixo mostra a implementação para o MIPS 32 de uma função que foi assim declarada: `int f (int n)`. Sabe-se que o montador utilizado sempre sintetiza todas as pseudo-instruções com um número mínimo de instruções nativas. Na primeira coluna, cada número identifica uma instrução diferente. Nestas condições, responda:

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bne     $a0,$zero,test
6           add     $v0,$zero,$zero
7           addi    $sp,$sp,12
8           jr      $ra
9   test:   bne     $a0,1,gen
10          addi    $v0,$zero,1
11          addi    $sp,$sp,12
12          jr      $ra
13  gen:    addi    $a0,$a0,-1
14          jal     f
15          add     $s0,$v0,$zero
16          addi    $a0,$a0,-1
17          jal     f
18          add     $v0,$v0,$s0
19          lw      $a0,0($sp)
20          lw      $s0,4($sp)
21          lw      $ra,8($sp)
22          addi    $sp,$sp,12
23          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6 a 8), (9), (10 a 12), (13,14), (15 a 17).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 8.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 10.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 36.

O código abaixo mostra a implementação para o MIPS 32 de uma função que foi assim declarada: `int f (int n)`. Sabe-se que o montador utilizado sempre sintetiza todas as pseudo-instruções com um número mínimo de instruções nativas. Na primeira coluna, cada número identifica uma instrução diferente. Nestas condições, responda:

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bgt     $a0,$zero,test
6           add     $v0,$zero,$zero
7           addi    $sp,$sp,12
8           jr      $ra
9   test:   addi    $t0,$zero,1
10          bne     $a0,$t0,gen
11          addi    $v0,$zero,1
12          jr      $ra
13  gen:    addi    $a0,$a0,-1
14          jal     f
15          add     $s0,$v0,$zero
16          addi    $a0,$a0,-1
17          jal     f
18          add     $v0,$v0,$s0
19          lw      $a0,0($sp)
20          lw      $s0,4($sp)
21          lw      $ra,8($sp)
22          addi    $sp,$sp,12
23          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6 a 8), (9,10), (11 a 13), (14,15), (16 a 18).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 9.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 11.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 39.

...

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bgt     $a0,$zero,test
6           add     $v0,$zero,$zero
7           j       rst
8   test:   bne     $a0,1,gen
9           addi    $v0,$zero,1
10          j       rst
11  gen:    addi    $a0,$a0,-1
12          jal     f
13          add     $s0,$v0,$zero
14          addi    $a0,$a0,-1
15          jal     f
16          add     $v0,$v0,$s0
17  rst:    lw      $a0,0($sp)
18          lw      $s0,4($sp)
19          lw      $ra,8($sp)
20          addi    $sp,$sp,12
21          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6,7), (8), (9,10), (11,12), (13 a 15), (16).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 13.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 15.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 47.

...

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bgt     $a0,$zero,test
6           add     $v0,$zero,$zero
7           addi    $sp,$sp,12
8           jr      $ra
9   test:   addi    $t0,$zero,1
10          bne     $a0,$t0,gen
11          addi    $v0,$zero,1
12          addi    $sp,$sp,12
13          jr      $ra
14  gen:    addi    $a0,$a0,-1
15          jal     f
16          add     $s0,$v0,$zero
17          addi    $a0,$a0,-1
18          jal     f
19          add     $v0,$v0,$s0
20          lw      $a0,0($sp)
21          lw      $s0,4($sp)
22          lw      $ra,8($sp)
23          addi    $sp,$sp,12
24          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6,8), (9,10), (11 a 13), (14,15), (16 a 18).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 9.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 11.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 39.

...

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bgt     $a0,$zero,test
6           add     $v0,$zero,$zero
7           j       rst
8   test:   addi    $t0,$zero,1
9           bne     $a0,$t0,gen
10          addi    $v0,$zero,1
11          j       rst
12  gen:    addi    $a0,$a0,-1
13          jal     f
14          add     $s0,$v0,$zero
15          addi    $a0,$a0,-1
16          jal     f
17          add     $v0,$v0,$s0
18  rst:    lw      $a0,0($sp)
19          lw      $s0,4($sp)
20          lw      $ra,8($sp)
21          addi    $sp,$sp,12
22          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6,7), (8,9), (10,11), (12,13), (14 a 16), (17).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 13.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 15.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 47.

...

```
1   f:      addi    $sp,$sp,-12
2           sw      $ra,8($sp)
3           sw      $s0,4($sp)
4           sw      $a0,0($sp)
5           bne     $a0,$zero,test
6           add     $v0,$zero,$zero
7           j       rst
8   test:   bne     $a0,1,gen
9           addi    $v0,$zero,1
10          j       rst
11  gen:    addi    $a0,$a0,-1
12          jal     f
13          add     $s0,$v0,$zero
14          addi    $a0,$a0,-1
15          jal     f
16          add     $v0,$v0,$s0
17  rst:    lw      $a0,0($sp)
18          lw      $s0,4($sp)
19          lw      $ra,8($sp)
20          addi    $sp,$sp,12
21          jr      $ra
```

As seguintes sequências de instruções formam blocos básicos: (6,7), (8), (9,10), (11,12), (13 a 15), (16).
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=0 é: 13.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=1 é: 15.
O número de instruções **nativas** executadas quando f é chamada com o parâmetro n=2 é: 47.

Questão 10 - P1

A sequência de instruções abaixo mostra a implementação de uma função `f` (originalmente escrita em linguagem C) compatível com a arquitetura MIPS32. Sabe-se que a função `f` invoca uma outra função que foi assim declarada: `int func(int a, int b)`. No código abaixo, cada número identifica uma dentre as 22 instruções. Nestas condições, responda:

```
1   f:      add     $sp,$sp,-20
2           sw      $ra,16($sp)
3           sw      $s0,12($sp)
4           sw      $s1,8($sp)
5           sw      $s2,4($sp)
6           sw      $s3,0($sp)
7           move    $s0,$a0
8           move    $s1,$a1
9           move    $s2,$a2
10          move    $s3,$a3
11          move    $a0,$a3
12          jal     funct
13          move    $a0,$v0
14          add     $a1,$s0,$s2
15          jal     funct
16          lw      $ra,16($sp)
17          lw      $s0,12($sp)
18          lw      $s1,8($sp)
19          lw      $s2,4($sp)
20          lw      $s3,0($sp)
21          addi    $sp,$sp,20
22          jr      $ra
```

A implementação acima corresponde à função:

```
int f(int a, int b, int c, int d)
{
     return func(func(d,b),a+c)
}
```

Suponha agora que se queira otimizar a implementação acima através da remoção de instruções, mas de forma que a implementação otimizada ainda funcione corretamente e obedeça à convenção de chamadas. Nestas condições, responda: as seguintes instruções poderiam ser removidas com pequenos ajustes no código remanescente: [(4, 6, 8, 10, 18, 20)]

A sequência de instruções abaixo mostra a implementação de uma função `f` (originalmente escrita em linguagem C) compatível com a arquitetura MIPS32. Sabe-se que a função `f` invoca uma outra função que foi assim declarada: `int func(int a, int b)`. No código abaixo, cada número identifica uma dentre as 22 instruções. Nestas condições, responda:

```
1   f:      add     $sp,$sp,-20
2           sw      $ra,16($sp)
3           sw      $s0,12($sp)
4           sw      $s1,8($sp)
5           sw      $s2,4($sp)
6           sw      $s3,0($sp)
7           move    $s0,$a0
8           move    $s1,$a1
9           move    $s2,$a2
10          move    $s3,$a3
11          move    $a0,$a2
12          jal     funct
13          move    $a0,$v0
14          add     $a1,$s0,$s3
15          jal     funct
16          lw      $ra,16($sp)
17          lw      $s0,12($sp)
18          lw      $s1,8($sp)
19          lw      $s2,4($sp)
20          lw      $s3,0($sp)
21          addi    $sp,$sp,20
22          jr      $ra
```

A implementação acima corresponde à função:

```
int f(int a, int b, int c, int d)
{
     return func(func(c,b),a+d)
}
```

Suponha agora que se queira otimizar a implementação acima através da remoção de instruções, mas de forma que a implementação otimizada ainda funcione corretamente e obedeça à convenção de chamadas. Nestas condições, responda: as seguintes instruções poderiam ser removidas com pequenos ajustes no código remanescente: [(4, 5, 8, 9, 18, 19)]

A sequência de instruções abaixo mostra a implementação de uma função `f` (originalmente escrita em linguagem C) compatível com a arquitetura MIPS32. Sabe-se que a função `f` invoca uma outra função que foi assim declarada: `int func(int a, int b)`. No código abaixo, cada número identifica uma dentre as 22 instruções. Nestas condições, responda:

```
1   f:      add     $sp,$sp,-20
2           sw      $ra,16($sp)
3           sw      $s0,12($sp)
4           sw      $s1,8($sp)
5           sw      $s2,4($sp
6           sw      $s3,0($sp)
7           move    $s0,$a0
8           move    $s1,$a1
9           move    $s2,$a2
10          move    $s3,$a3
11          move    $a1,$a3
12          jal     funct
13          move    $a0,$v0
14          add     $a1,$s1,$s2
15          jal     funct
16          lw      $ra,16($sp)
17          lw      $s0,12($sp)
18          lw      $s1,8($sp)
19          lw      $s2,4($sp)
20          lw      $s3,0($sp)
21          addi    $sp,$sp,20
22          jr      $ra
```

A implementação acima corresponde à função:

```
int f(int a, int b, int c, int d)
{
     return func(func(a,d),b+c)
}
```

Suponha agora que se queira otimizar a implementação acima através da remoção de instruções, mas de forma que a implementação otimizada ainda funcione corretamente e obedeça à convenção de chamadas. Nestas condições, responda: as seguintes instruções poderiam ser removidas com pequenos ajustes no código remanescente: [(3, 6, 7, 10, 17, 20)]

A sequência de instruções abaixo mostra a implementação de uma função `f` (originalmente escrita em linguagem C) compatível com a arquitetura MIPS32. Sabe-se que a função `f` invoca uma outra função que foi assim declarada: `int func(int a, int b)`. No código abaixo, cada número identifica uma dentre as 22 instruções. Nestas condições, responda:

```
1   f:      add     $sp,$sp,-20
2           sw      $ra,16($sp)
3           sw      $s0,12($sp)
4           sw      $s1,8($sp)
5           sw      $s2,4($sp
6           sw      $s3,0($sp)
7           move    $s0,$a0
8           move    $s1,$a1
9           move    $s2,$a2
10          move    $s3,$a3
11          move    $a1,$a2
12          jal     funct
13          move    $a0,$v0
14          add     $a1,$s1,$s3
15          jal     funct
16          lw      $ra,16($sp)
17          lw      $s0,12($sp)
18          lw      $s1,8($sp)
19          lw      $s2,4($sp)
20          lw      $s3,0($sp)
21          addi    $sp,$sp,20
22          jr      $ra
```

A implementação acima corresponde à função:

```
int f(int a, int b, int c, int d)
{
     return func(func(a,c),b+d)
}
```

Suponha agora que se queira otimizar a implementação acima através da remoção de instruções, mas de forma que a implementação otimizada ainda funcione corretamente e obedeça à convenção de chamadas. Nestas condições, responda: as seguintes instruções poderiam ser removidas com pequenos ajustes no código remanescente: [(3, 5, 7, 9, 17, 19)]

Questão 11 - P2 & P3

Suponha que o MIPS tenha sido estendido com instruções e registradores similares às da extensão AVX do x86, mas de forma que as novas instruções utilizem a ordem original de operandos do assembly do MIPS, conforme a tabela abaixo, onde y0, y1 e y2 são registradores de 256 bits:

Tipo	Nova instrução	Similar no AVX	Fluxo de dados
Load vetorial	vload.pd $y0,($s1)	vmovapd	yo ← memória[$s1]
	vload.ps $y0,($s1)	vmovaps	
Load múltiplo	vbroad.sd $y0,($s1)	vbroadcastsd	
	vbroad.ss $y0,($s1)	vbroadcastss	
Store vetorial	vstore.pd $y0,($s1)	vmovapd	memória[$s1] ← y0
	vstore.ps $y0,($s1)	vmovapd	
Soma vetorial	vadd.pd $y2,$y1,$y0	vaddpd	$y2 ←$y1 + $y0
	vadd.ps $y2,$y1,$y0	vaddps	
Produto vetorial	vmul.pd $y2,$y1,$y0	vmulpd	$y2 ←$y1 x $y0
	vmul.ps $y2,$y1,$y0	vmulpd	

O código abaixo implementa a seguinte operação vetorial: $\mathbf{Y} = \mathbf{a} \times \mathbf{X} + \mathbf{Y}$, onde **X** e **Y** são vetores e **a** é uma variável escalar, ou seja, $\mathbf{Y[i]} = \mathbf{a} \times \mathbf{X[i]} + \mathbf{Y[i]}$ para cada componente i dos vetores. A variável escalar **a** reside em memória. O registrador $s0 contém o endereço da variável **a** e os registradores $s5 e $s6 contêm os endereços-base dos arranjos **X** e **Y**, respectivamente. O nome **value** representa uma constante cujo valor é **16384**. O nome **increment** representa uma constante cujo valor você terá que deduzir.

```
        vbroad.sd   $y0,0($s0)
        addi        $s1,$s5,value
Loop:   vload.pd    $y1,($s5)
        vmul.pd     $y1,$y1,$y0
        vload.pd    $y2,($s6)
        vadd.pd     $y2,$y2,$y1
        vstore.pd   $y2,($s6)
        addi        $s5,$s5,increment
        addi        $s6,$s6,increment
        bne         $s1,$s5,Loop
```

Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 4098

... uma constante cujo valor é **128**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 34

... uma constante cujo valor é **256**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 66

... uma constante cujo valor é **512**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 130

... uma constante cujo valor é **1024**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 258

... uma constante cujo valor é **4096**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 1026

... uma constante cujo valor é **8192**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 2050

... uma constante cujo valor é **65536**...
Quantas instruções são executadas para realizar essa operação vetorial?

Resposta: 16386

Questão 12 - P2

Para um mesmo programa, dois compiladores distintos, C1 e C2, produziram arquivos executáveis diferentes, X1 e X2, respectivamente, para a arquitetura ARMv7. Pretende-se avaliar o desempenho desse programa – quando executado com exatamente os mesmos dados de entrada – em dois chips distintos e sob diferentes frequências de operação, conforme a tabela abaixo.

	Chip 1: Cortex A8		**Chip 2: Cortex A15**	
Frequência	1 GHz	0,6 GHz	1,5 GHz	2 GHz

Mediram-se o tempo de execução e o número de instruções em apenas um dos quatro cenários e os resultados estão na tabela abaixo. A partir destes resultados experimentais, quer-se estimar o que ocorrerá nos demais cenários.

	A8 (1 GHz)	
Executável	**Tempo de execução**	**Instruções executadas**
X1	**1 s**	10^9
X2	**1,5 s**	$1,2 \times 10^9$

Para o A8 a 0,6 GHz, o tempo de execução de X1 será 10/6 s.
Para o A15 a 1,5 GHz, nada se pode afirmar sobre o tempo de execução de X1.
Para o A15 a 2 GHz, nada se pode afirmar sobre o tempo de execução de X1
Para o A15 a 2 GHz, nada se pode afirmar sobre o tempo de execução de X2.

... Mediram-se o tempo de execução e o número de instruções em apenas um dos quatro cenários e os resultados estão na tabela abaixo. A partir destes resultados experimentais, quer-se estimar o que ocorrerá nos demais cenários.

	A8 (1 GHz)	
Executável	**Tempo de execução**	**Instruções executadas**
X1	**2/3 s**	10^9
X2	**1 s**	$1,2 \times 10^9$

Para o A8 a 0,6 GHz, nada se pode afirmar sobre o tempo de execução de X1.
Para o A8 a 0,6 GHz, nada se pode afirmar sobre o tempo de execução de X2
Para o A15 a 1 GHz, nada se pode afirmar sobre o tempo de execução de X1
Para o A15 a 1 GHz, nada se pode afirmar sobre o tempo de execução de X2.
Para o A15 a 2 GHz, o tempo de execução de X1 será 0,5 s.

... Mediram-se o tempo de execução e o número de instruções em apenas um dos quatro cenários e os resultados estão na tabela abaixo. A partir destes resultados experimentais, quer-se estimar o que ocorrerá nos demais cenários.

	A8 (1 GHz)	
Executável	**Tempo de execução**	**Instruções executadas**
X1	**10/6 s**	10^9
X2	**2,5 s**	$1,2 \times 10^9$

Para o A8 a 1 GHz, o tempo de execução de X1 será 1 s.
Para o A15 a 1,5 GHz, nada se pode afirmar sobre o tempo de execução de X1
Para o A15 a 1,5 GHz, nada se pode afirmar sobre o tempo de execução de X2
Para o A15 a 2 GHz, nada se pode afirmar sobre o tempo de execução de X1
Para o A15 a 2 GHz, nada se pode afirmar sobre o tempo de execução de X2

Questão 13 - P2

Lembre que o MIPS usa um pipeline de 5 estágios (IF, ID, EX, ME, WB), que permite a escrita e a leitura de registradores em semiciclos distintos de um mesmo ciclo. Suponha que o pipeline original seja assim modificado:

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

Sabe-se que o datapath é capaz de detectar hazards de dados para provocar pausas (quando necessário), **mas não possui suporte a forwarding**. Cada código da tabela abaixo será executado (individualmente) no novo pipeline.

A	B	C
`lw $s1,8($s0)`	`lw $s1,8($s0)`	`sw $s1,8($s0)`
`sw $s2,8($s1)`	`sw $s1,8($s2)`	`lw $s2,8($s0)`

Nestas condições, **A** requer 10 ciclos, **B** requer 10 ciclos e **C** requer 8 ciclos.

...

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 10 ciclos, **B** requer 10 ciclos e **C** requer 7 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, **A** requer 9 ciclos, **B** requer 9 ciclos e **C** requer 7 ciclos.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 12 ciclos, **B** requer 12 ciclos e **C** requer 9 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, **A** requer 11 ciclos, **B** requer 11 ciclos e **C** requer 9 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 13 ciclos, **B** requer 13 ciclos e **C** requer 10 ciclos.

Questão 14 - P2

Lembre que o MIPS usa um pipeline de 5 estágios (IF, ID, EX, ME, WB), que permite a escrita e a leitura de registradores em semiciclos distintos de um mesmo ciclo. Suponha que o pipeline original seja assim modificado:

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o datapath é capaz de detectar hazards de dados para provocar pausas (quando necessário) e **possui suporte a full forwarding**. Cada código da tabela abaixo será executado (individualmente) no novo pipeline.

A		B		C	
`lw`	`$s1,8($s0)`	`lw`	`$s1,8($s0)`	`sw`	`$s1,8($s0)`
`sw`	`$s2,8($s1)`	`sw`	`$s1,8($s2)`	`lw`	`$s2,8($s0)`

Nestas condições, **A** requer 9 ciclos, **B** requer 7 ciclos e **C** requer 7 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, **A** requer 8 ciclos, **B** requer 7 ciclos e **C** requer 7 ciclos.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, **A** requer 9 ciclos, **B** requer 8 ciclos e **C** requer 8 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, **A** requer 10 ciclos, **B** requer 9 ciclos e **C** requer 9 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 10 ciclos, **B** requer 8 ciclos e **C** requer 8 ciclos.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 11 ciclos, **B** requer 9 ciclos e **C** requer 9 ciclos.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, **A** requer 12 ciclos, **B** requer 10 ciclos e **C** requer 10 ciclos.

Questão 15 - P2

Lembre que o MIPS usa um pipeline de 5 estágios (IF, ID, EX, ME, WB), que permite a escrita e a leitura de registradores em semiciclos distintos de um mesmo ciclo. Suponha que o pipeline original seja assim modificado:

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo). Suponha que o código abaixo seja executado em duas variantes diferentes de datapath:

Variante A: Resultado do teste de condição disponível na saída da ALU (ao final do ciclo).
Variante B: Resultado do teste de condição disponível na saída do estágio ID (ao final do ciclo.

```
            beq    $s0,$s1,L
            add    $s2,$s3,$s4
            add    $t0,$t1,$t2
            add    $t3,$t4,$t5
            add    $t6,$t7,$t8
      L:    sw     $s5,8($s6)
```

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [10] ciclos na variante A e [9] ciclos na variante B.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [9] ciclos na variante A e [8] ciclos na variante B.

...

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [10] ciclos na variante A e [8] ciclos na variante B.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

...

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [11] ciclos na variante A e [10] ciclos na variante B.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [12] ciclos na variante A e [10] ciclos na variante B.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

...

Nestas condições, supondo o cenário em que a **previsão resulte incorreta**, o código abaixo requer [11] ciclos na variante A e [9] ciclos na variante B.

Questão 16 - P2

Um programa paralelo executa em 100 segundos em um único core. Sabe-se que 20% é a percentagem do tempo gasto executando instruções que estão encadeadas por dependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 40 cores, qual será o tempo de execução do programa em segundos?

Resposta: 22

Um programa paralelo executa em 200 segundos em um único core. Sabe-se que 10% é a percentagem do tempo gasto executando instruções que estão encadeadas por dependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 90 cores, qual será o tempo de execução do programa em segundos?

Resposta: 22

Um programa paralelo executa em 150 segundos em um único core . Sabe-se que 10% é a percentagem do tempo gasto executando instruções que estão encadeadas pordependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 90 cores, qual será o tempo de execução do programa emsegundos?

Resposta: 16,5

Um programa paralelo executa em 200 segundos em um único core. Sabe-se que 10% é a percentagem do tempo gasto executando instruções que estão encadeadas por dependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 9 cores , qual será o tempo de execução do programa em segundos?

Resposta: 40

Um programa paralelo executa em 250 segundos em um único core. Sabe-se que 10% é a percentagem do tempo gasto executando instruções que estão encadeadas por dependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 90 cores, qual será o tempo de execução do programa em segundos?

Resposta: 27,5

Um programa paralelo executa em 250 segundos em um único core. Sabe-se que 20% é a percentagem do tempo gasto executando instruções que estão encadeadas por dependências de dados. Quer-se aumentar o desempenho do programa através do aumento do número de cores, sendo que cada um deles é idêntico ao core original. Em um processador com 4 cores, qual será o tempo de execução do programa em segundos?

Resposta: 100

Questão 17 - P3

Um processador permite endereçar 2^{32} bytes. Um programa compilado para esse processador requer um espaço de endereçamento de 2^{30} bytes. Sabe-se que o sistema de gerência de memória virtual aloca páginas somente na faixa de endereços correspondente aos primeiros **1073741824** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **65536** bytes. Suponha que se use o mínimo número de bytes para acomodar uma entrada da tabela de páginas (não se esqueça que esse número precisa ser inteiro). Sabe-se que são necessários 3 bits para codificar o estado de uma página. Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 49152.

... de endereços correspondente aos primeiros **2147483648** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **32768** bytes... Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 98304

... de endereços correspondente aos primeiros **1073741824** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **2048** bytes... Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 1572864

... de endereços correspondente aos primeiros **2147483648** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **16384** bytes... Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 196608

... de endereços correspondente aos primeiros **2147483648** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **1024** bytes... Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 3145728

... de endereços correspondente aos primeiros **1073741824** bytes da memória física (o resto da memória é usado para outros propósitos como, por exemplo, manter a tabela de páginas). Sabe-se que o tamanho de página adotado é de **8192** bytes... Nestas condições, quantos bytes são necessários para armazenar a tabela de páginas desse programa?

Resposta: 393216.

Questão 18 - P3

Um *single-core chip* adota endereços de 32 bits e usa uma cache **1-way** com capacidade de **4096** bytes de dados. Cada entrada da cache armazena um bloco de dados com **8** bytes, além dos bits necessários para codificar o estado do bloco. Sabe-se que um deles é um *reference bit*, que é usado para implementar um critério LRU aproximado. Sabe-se também que a cache adota a política de *write-back* e, portanto, requer um *dirty bit*. Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 44544.

Um *single-core chip* adota endereços de 32 bits e usa uma cache **1-way** com capacidade de **8192** bytes de dados. Cada entrada da cache armazena um bloco de dados com **16** bytes..
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 76800.

Um *single-core chip* adota endereços de 32 bits e usa uma cache **1-way** com capacidade de **32768** bytes de dados. Cada entrada da cache armazena um bloco de dados com **16** bytes..
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 344064.

Um *single-core chip* adota endereços de 32 bits e usa uma cache **2-way** com capacidade de **4096** bytes de dados. Cada entrada da cache armazena um bloco de dados com **8** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 45056

Um *single-core chip* adota endereços de 32 bits e usa uma cache **2-way** com capacidade de **4096** bytes de dados. Cada entrada da cache armazena um bloco de dados com **16** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 45056

Um *single-core chip* adota endereços de 32 bits e usa uma cache **2-way** com capacidade de **16384** bytes de dados. Cada entrada da cache armazena um bloco de dados com **8** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 176128.

Um *single-core chip* adota endereços de 32 bits e usa uma cache **2-way** com capacidade de **32768** bytes de dados. Cada entrada da cache armazena um bloco de dados com **16** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 305152.

Um *single-core chip* adota endereços de 32 bits e usa uma cache **4-way** com capacidade de **16384** bytes de dados. Cada entrada da cache armazena um bloco de dados com **8** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 178176

Um *single-core chip* adota endereços de 32 bits e usa uma cache **8-way** com capacidade de **16384** bytes de dados. Cada entrada da cache armazena um bloco de dados com **8** bytes...
Neste cenário, qual o número total de bits requerido para implementar toda a cache?

Resposta: 180224.

Questão 19 - P3

Um processador adota uma hierarquia de memória com dois níveis de cache e a memória principal (MP). No primeiro nível, há uma cache de instruções (I-L1) e uma cache de dados (D-L1) separadas. No segundo nível, há uma cache unificada para dados e instruções (L2). Os parâmetros das memórias são os seguintes:

Acesso à MP = 100 ciclos.
Acesso a L2 = 10 ciclos.
Acesso a I-L1 ou D-L1 = 1 ciclo.

Ao rodar nesse processador, um programa repete – um número muito grande de vezes – as instruções de um pequeno laço, de forma que a taxa de faltas em I-L1 pode ser considerada nula para fins práticos, mas o mesmo não acontece para as demais caches, como mostram as seguintes taxas de falta (expressas como números fracionários):

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,050.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,025.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **6,0** ciclos por instrução. Qual seria o número médio de ciclos por instrução nesse processador na situação em que D-L1 também exibisse taxa de faltas nula?
(Atenção: A tolerância no valor da resposta é de 0,01. Por isso, efetue os cálculos com precisão).

Resposta: 5,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,050.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,025.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **8,0** ciclos por instrução...

Resposta: 4,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,100.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,050.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **7,0** ciclos por instrução...

Resposta: 5,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,100.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,050.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **8,0** ciclos por instrução...

Resposta: 6,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,150.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,075.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **6,0** ciclos por instrução...

Resposta: 3,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,200.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,100.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **6,0** ciclos por instrução...

Resposta: 2,0.

...

Taxa de faltas no acessos a I-L1: 0,0.
Taxa de faltas no acesso a D-L1: 0,250.
Taxa de faltas (global) no acesso aos níveis L1 e L2: 0,125.

Sabe-se que a fração de instruções que acessam dados em memória é 1/3 e que a execução do programa requer em média **8,0** ciclos por instrução...

Resposta: 3,0.

Questão 20 - P3

Lembre que o MIPS usa um pipeline de 5 estágios (IF, ID, EX, ME, WB), que permite a escrita e a leitura de registradores em semiciclos distintos de um mesmo ciclo. Suponha que o pipeline original seja assim modificado:

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída do estágio ID** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.

Um programa, cujas percentagens de instruções executadas são mostradas na tabela abaixo, resulta numa precisão de 75% para as previsões de desvios e numa taxa de faltas de 10% na cache de dados. A penalidade de falta no acesso à cache de dados é 10 ciclos. Suponha: 1) emissão dinâmica de até uma instrução por ciclo; 2) não ocorrem hazards estruturais nem hazards de dados; 3) a taxa de faltas na cache de instruções é zero.

`beq`	`jump`	`lw`	`sw`	`R`
25 %	5 %	30 %	10 %	30 %

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,525.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída da ALU** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.
...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,525.

...

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída do estágio ID** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.

...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,4625.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.
- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída da ALU** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.

...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,65.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída do estágio ID** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.
...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,525.

...

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída da ALU** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.
...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,5875.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída da ALU** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.
...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,65.

...

- O estágio IF foi particionado em dois estágios **IF1** e **IF2**, separados por uma barreira temporal (o primeiro acomoda o decodificar de endereço e o segundo as células de memória). Isto permite que o endereço de uma instrução seja decodificado em paralelo com o acesso (em memória) à instrução cujo endereço foi decodificado no ciclo anterior.

- O estágio ME foi particionado em dois estágios **ME1** e **ME2**, separados por uma barreira temporal (o primeiro acomoda o decodificador de endereços e o segundo as células de memória). Isto permite que o endereço de um dado seja decodificado em paralelo com o acesso (em memória) ao dado cujo endereço foi decodificado no ciclo anterior.

- O estágio EX foi particionado em dois estágios **EX1** e **EX2**, separados por uma barreira temporal (o primeiro acomoda a propagação de carry nos bits menos significativos e o segundo a propagação de carry nos bits mais significativos). Isto permite a sobreposição de operações aritméticas no mesmo ciclo de relógio.

Sabe-se que o hardware faz previsão estática sob a **hipótese de desvio não tomado** e é **capaz de anular instruções** que tenham sido buscadas indevidamente. Sabe-se que o **endereço-alvo** de um desvio condicional está **disponível na saída do estágio ID** (ao final do ciclo).

Sabe-se também que o resultado do teste de um desvio condicional está disponível na **saída do estágio ID** (ao final do ciclo), permitindo a busca da instrução-alvo no ciclo seguinte.
...

Nestas condições, o número médio de ciclos por instrução ao se executar esse programa é 1,525.

Questão 21 - P3

Um processador usa endereços de 32 bits e uma pequena cache de mapeamento direto. A Tabela abaixo mostra uma sequência de referências à memória em ordem temporal (de cima para baixo). Cada linha da tabela representa os 16 bits menos significativos de um endereço referenciado, sendo que os 16 bits mais significativos são todos zeros. Sabe-se que a cache tem capacidade para armazenar apenas **2** blocos de memória e que cada bloco ocupa **4** bytes. Nestas condições, responda:

00000000	00	00	00	11
00000000	10	11	01	00
00000000	00	10	10	11
00000000	00	00	00	10
00000000	10	11	11	11
00000000	01	01	10	00
00000000	10	11	11	10
00000000	00	00	11	10
00000000	10	11	01	01
00000000	00	10	11	00
00000000	10	11	10	10
00000000	11	11	11	01
00000000	10	11	10	11

O número de acessos à memória que resultam em acertos em cache é 2
O número de acessos à memória que resultam em substituição de bloco em cache é 9

... capacidade para armazenar apenas **2** blocos de memória e que cada bloco ocupa **8** bytes...
O número de acessos à memória que resultam em acertos em cache é 1
O número de acessos à memória que resultam em substituição de bloco em cache é 10

... capacidade para armazenar apenas **4** blocos de memória e que cada bloco ocupa **4** bytes...
O número de acessos à memória que resultam em acertos em cache é 4
O número de acessos à memória que resultam em substituição de bloco em cache é 5

... capacidade para armazenar apenas **4** blocos de memória e que cada bloco ocupa **8** bytes...
O número de acessos à memória que resultam em acertos em cache é 2
O número de acessos à memória que resultam em substituição de bloco em cache é 7

... capacidade para armazenar apenas **8** blocos de memória e que cada bloco ocupa **4** bytes...
O número de acessos à memória que resultam em acertos em cache é 4
O número de acessos à memória que resultam em substituição de bloco em cache é 3

... capacidade para armazenar apenas **16** blocos de memória e que cada bloco ocupa **4** bytes...
O número de acessos à memória que resultam em acertos em cache é 4
O número de acessos à memória que resultam em substituição de bloco em cache é 1

... capacidade para armazenar apenas **16** blocos de memória e que cada bloco ocupa **8** bytes...
O número de acessos à memória que resultam em acertos em cache é 4
O número de acessos à memória que resultam em substituição de bloco em cache é 0

Questão 22 - P3

A figura abaixo mostra a **ordem temporal** em que as instruções de um programa paralelo são executadas num *dual-core*, onde **X** e **Y** são endereços de variáveis compartilhadas e as reticências representam instruções aritméticas. Cada núcleo tem suas próprias caches de dados e instruções (separadas). Há um único nível de cache. Todas as caches de dados têm blocos de 16 bytes e usam *write allocate*. O protocolo de coerência codifica cada estado em dois bits: bit de validade e *dirty* bit. Nestas condições, para os eventos induzidos pelo trecho observado do programa, qual(is) das seguintes afirmações é (são) verdadeira(s)?

Tempo	Núcleo A	Núcleo B
0	`sw $s0,X`	...
1	...	...
2	`lw $s1,X`	...
3	...	`lw $s2,Y`
4	`sw $s3,X`	...
5	...	`lw $s4,Y`

Cenário 1: X = `0xCCCCCCC0`, **Y** = `0xCCCCCCCA`
Cenário 2: Y = `0xBBBBBBBC`, **Y** = `0xAAAAAAAC`

... Todas as caches de dados têm blocos de 16 bytes...

Cenário 1: X = `0xF0F0F0FA`, **Y** = `0xF0F0F0F0`
Cenário 2: Y = `0xF000000B`, **Y** = `0xDDDDDDDB`

...Todas as caches de dados têm blocos de 256 bytes...

Cenário 1: X = `0xF0F0F0FA`, **Y** = `0xF0F0F0C0`
Cenário 2: Y = `0x00000FBB`, **Y** = `0xF00000BB`

...Todas as caches de dados têm blocos de 16 bytes...

Cenário 1: X = `0xBBBBBBCA`, **Y** = `0xBBBBBBC0`
Cenário 2: Y = `0xAAAAAAAB`, **Y** = `0xCCCCCCCB`

...Todas as caches de dados têm blocos de 16 bytes...

Cenário 1: X = `0xAAAAAAA0`, **Y** = `0xAAAAAAAA`
Cenário 2: Y = `0xBBBBBBB0`, **Y** = `0xCCCCCCC0`

...Todas as caches de dados têm blocos de 256 bytes...

Cenário 1: X = `0xAAAAAAA0`, **Y** = `0xAAAAAABA`
Cenário 2: Y = `0xBBBBBBD0`, **Y** = `0xCCCCCCD0`

No Cenário 1, o número total de **acertos** nas caches de dados é 2.
No Cenário **1**, o número total de **leituras** da memória principal é 1.
No Cenário **1**, o número total de **escritas** na memória principal é 2.
No Cenário **2**, o número total de **acertos** nas caches de dados é 3.
No Cenário **2**, o número total de **leituras** da memória principal é 2.
No Cenário **2**, o número total de **escritas** na memória principal é 0.