UFSM
Universidade Federal de Santa Maria

Departamento de Eletrônica e Computação
ELC 1011 – Organização de Computadores

15/02/2021

Exame

Nome: Gabarito

1 – (1,0 ponto) Considere duas máquinas, P1 e P2, que usam o mesmo conjunto de instruções mas possuem implementações diferentes. O conjunto de instruções possui 4 classes de instruções: A, B, C e D. A frequência de relógio e o CPI em cada uma das implementações é dada pela seguinte tabela:

	Frequência do relógio (GHz)	CPI das classes de Instruções			
		A	B	C	D
P1	2,0	1	4	4	2
P2	2,5	2	2	2	2

Executamos um programa X, com 10^6 instruções, dividida nas classes da seguinte forma: 10% na classe A, 20% na classe B, 50% na classe C e 20% na classe D. Qual das máquinas possui melhor desempenho? Por quê?

Resposta:

Usaremos o tempo de execução t_{exe} como medida do desempenho.

$$t_{exe} = \frac{N_{clk}}{f_{clk}}$$

O número de ciclos de relógio N_{clk} é dado por:

$$N_{clk} = \sum_{i=A}^{D} N_{inst}^{i} \cdot CPI^{i}$$

Nas equações anteriores, N_{clk} é o número de ciclos de relógio, f_{clk} a frequência de relógio do processador, i a classe da instrução, N_{int}^{i} o número de instruções da classe i e CPI^{i} o número e ciclos de relógio por instrução da classe i.

Os dois processadores, P1 e P2, executam o mesmo número de instruções:

$$N_{inst}^{A} = 0,1 \cdot 10^6 = 10^5 \text{ instruções}$$

$$N_{inst}^{B} = 0,2 \cdot 10^6 = 2 \cdot 10^5 \text{ instruções}$$

$$N_{inst}^{C} = 0,5 \cdot 10^6 = 5 \cdot 10^5 \text{ instruções}$$

$$N_{inst}^{D} = 0,2 \cdot 10^6 = 2 \cdot 10^5 \text{ instruções}$$

Calculamos o número de ciclos de relógio para a execução do programa, para cada um dos processadores. Para o processador P1:

$$
\begin{aligned}
N_{clk} =& \sum_{i=A}^{D} N_{inst}^{i} \cdot CPI^{i} \\
=& (10^5 \cdot 1) + (2 \times 10^5 \cdot 4) + (5 \times 10^5 \cdot 4) + (2 \times 10^5 \cdot 2) \\
=& 3{,}3 \times 10^6 \,\text{ciclos de relógio} \qquad (1)
\end{aligned}
$$

Para o processador P2:

$$
\begin{aligned}
N_{clk} =& \sum_{i=A}^{D} N_{inst}^{i} \cdot CPI^{i} \\
=& (10^5 \cdot 2) + (2 \times 10^5 \cdot 2) + (5 \times 10^5 \cdot 2) + (2 \times 10^5 \cdot 2) \\
=& 2{,}0 \times 10^6 \,\text{ciclos de relógio} \qquad (1)
\end{aligned}
$$

Calculamos o tempo de execução do programa X, para cada um dos processadores. Para o processador P1:

$$
t_{exe} = \frac{N_{clk}}{f_{clk}} = \frac{3{,}3 \times 10^6}{2{,}0 \times 10^9} = 1{,}65\,\text{ms}
$$

Para o processador P2:

$$
t_{exe} = \frac{N_{clk}}{f_{clk}} = \frac{2{,}0 \times 10^6}{2{,}5 \times 10^9} = 0{,}8\,\text{ms}
$$

O processador P2 possui um melhor desempenho, quando comparado ao processador P1, se utilizamos o tempo de execução como medida de desempenho.

2 – (4,0 pontos) Traduza o seguinte programa de C para o *assembly* do processador MIPS.

```c
int valor1 = 10;

int procedimento3(int x) {
  int tmp;

  if (x > 5) {
    tmp = x;
  } else {
    tmp = x + valor1;
  }

  return x;
}

int procedimento2(int v1[], int n) {
  int i;

  int acc;
  acc = 0;

  for (i = 0; i < n; i++) {
    acc = acc + procedimento3(v1[i]);
  }

  return acc;
}

int procedimento1(int x, int y) {
  int vetorA[10];
  int vetorB[10];
  int i;
  int resultado;

  resultado = 0;
  for (int i = 0; i < 10; i++) {
    vetorA[i] = x + i;
    vetorB[i] = y + i;
  }
  resultado = procedimento2(vetorA, 10) - procedimento2(vetorB, 10);
  return resultado;
}

int main(void) {
  int n;
  int m;
  int resultado;

  n = 5;
  m = 3;

  resultado = procedimento1(n, m);
  return 0;
}
```

Resposta:

A seguir apresentamos um programa em *assembly* para o processador MIPS.

```
.data
#int valor1 = 10;
valor1:     .word 10                # variável global valor 1 igual a 10

.text
            jal     main            # executamos o procedimento main
            move    $a0, $v0        # retornamos o valor de retorno de main
            li      $v0, 17         # serviço exit2
            syscall                 # executamos o serviço, encerramos o programa

################################################################################
#int procedimento3(int x) {
procedimento3:
# mapa da pilha
# |  tmp | $sp + 0
################################################################################
## prólogo ##
            addiu   $sp, $sp, -4    # ajustamos a pilha
 #int tmp;
## corpo do programa ##
p3_if:
  #if (x > 5) {
            li      $t0, 5          # $t0 <- 5
            slt     $t1, $t0, $a0   # $t1 = 1 se x > 5
            bne     $t1, $zero, p3_if_verdadeiro
  #} else {
    #tmp = x + valor1;
            la      $t2, valor1     # $t2 <- endereço de valor1
            lw      $t3, 0($t2)     # $t3 <- valor1
            add     $t4, $a0, $t3   # $t4 <- x + valor1
            sw      $t4, 0($sp)     # tmp = x + valor1
            j       p3_if_fim       # saímos do if
  #}
p3_if_verdadeiro:
    #tmp = x;
            sw      $a0, 0($sp)     # tmp = x;
p3_if_fim:                          # fim da estrutura if
## epílogo ##
  #return x;
            move    $v0, $a0        # retornamos x
            addiu   $sp, $sp, 4     # restauramos a pilha
            jr      $ra             # retornamos ao procedimento chamador
#}

################################################################################
#int procedimento2(int v1[], int n) {
procedimento2:
# mapa da pilha
# | $ra  | $sp + 16
# | v1[] | $sp + 12
# |  n   | $sp + 8
# |  i   | $sp + 4
# | acc  | $sp + 0
################################################################################
## prólogo ##
            addiu   $sp, $sp -20    # ajustamos a pilha
            sw      $ra, 16($sp)    # armazenamos o endereço de retorno na pilha
            sw      $a0, 12($sp)    # armazenamos v1[] na pilha
            sw      $a1, 8($sp)     # armazenamos n na pilha
  #int i;
  #int acc;
## corpo do programa ##
  #acc = 0;
            sw      $zero, 0($sp)   # acc = 0
p2_for:
  #for (i = 0; i < n; i++) {
p2_for_inicializacao:
            li      $t0, 0          # $t0 <- i
            sw      $t0, 4($sp)     # i = 0
p2_for_condicao:
            j       p2_for_testa_condicao
```

```
p2_for_codigo:
    #acc = acc + procedimento3(v1[i]);
            # o valor de i está em $t0, no teste da condição
            sll     $t3, $t0, 2     # $t3 <- 4*i
            lw      $t4, 12($sp)    # $t4 <- endereço base de v1
            addu    $t5, $t4, $t3   # $t5 <- endereço efetivo de v1[i]
            lw      $a0, 0($t5)     # $a0 <- valor de v1[i]
            jal     procedimento3   # chamamos o procedimento3
            lw      $t6, 0($sp)     # $v0 <-acc
            add     $v0, $t6, $v0   # $v0 <- acc + procedimento3(v1[i])
            sw      $v0, 0($sp)     # acc = acc + procedimento3(v1[i])
p2_for_incremento:
  #for (i = 0; i < n; i++) {
            lw      $t8, 4($sp)     # $t8 <- i
            addi    $t0, $t8, 1     # $t0 <- i + 1
            sw      $t0, 4($sp)     # i = i + 1
p2_for_testa_condicao:
  #for (i = 0; i < n; i++) {
            # a variável i está em $t0
            lw      $t1, 8($sp)     # $t1 <- n
            slt     $t2, $t0, $t1   # $t2 = 1 se i<n
            bne     $t2, $zero, p2_for_codigo
  #}

## epílogo ##
  #return acc;

            lw      $ra, 16($sp)    # restauramos o endereço de retorno
            addiu   $sp, $sp, 20    # restauramos a pilha
            jr      $ra             # retornamos ao procedimento chamador
#}


################################################################################
#int procedimento1(int x, int y) {
procedimento1:
# mapa da pilha
# |    $ra    | $sp + 92
# |     $s0     | $sp + 88 usamos o registradores $s0 para guardar procedimento2(vetorB,
10)
# |   vetorA  | $sp + 48
# |   vetorB  | $sp + 8
# |     i     | $sp + 4
# | resultado | $sp + 0
################################################################################
## prólogo ##
            addiu   $sp, $sp, -96   # ajustamos a pilha para o quadro do procedimento
            sw      $ra, 92($sp)    # guardamos o endereço de retorno na pilha
            sw      $s0, 88($sp)    # guardamos o valor do registrador $s0
  #int vetorA[10];
  #int vetorB[10];
  #int i;
  #int resultado;
## corpo do programa ##
  #resultado = 0;
            sw      $zero, 0($sp)   # resultado = 0
p1_for:
  #for (int i = 0; i < 10; i++) {   # Erro. retirar int
p1_for_inicializacao:
            addiu   $t0, $zero, 0   # $t0 = 0
            lw      $t0, 4($sp)     # i = 0
p1_for_condicao:
            j       p1_for_testa_condicao
p1_for_codigo:
    #vetorA[i] = x + i;
            # a variável i está no registrador $t0, após o teste da condição
            addi    $t1, $sp, 48    # $t1 <- endereço base de vetorA
            sll     $t2, $t0, 2     # $t2 <- 4*i
            add     $t3, $t1, $t2   # $t3 <- endereço efetivo de vetorA[i]
            add     $t4, $a0, $t0   # $t4 <- x + i
            sw      $t4, 0($t3)     # vetorA[i] = x + i
```

```
    #vetorB[i] = y + i;
            addi    $t1, $sp, 8     # $t1 <- endereço base de vetorB
            # 4*i está em $t2
            add     $t3, $t1, $t2   # $t3 <- endereço efetivo de vetorB[i]
            add     $t4, $a1, $t0   # $t4 <- y + i
            sw      $t4, 0($t3)     # vetorA[i] = y + i
p1_for_incrementa:
            addi    $t0, $t0, 1     # $t0 <- i + 1
            sw      $t0, 4($sp)     # i = i + 1
p1_for_testa_condicao:
  #for (int i = 0; i < 10; i++) {
            # Uma cópia da variável i está em $t0
            slti    $t1, $t0, 10    # $t1 = 1 se i < 10
            bne     $t1, $zero, p1_for_codigo
  #}

    #resultado = procedimento2(vetorA, 10) - procedimento2(vetorB, 10);
            addi    $a0, $sp,  8    # $a0 <- endereço do vetorB
            li      $a1, 10         # $a1 < - 10
            jal     procedimento2   # $v0 <- procedimento2(vetorB, 10)
            move    $s0, $v0        # $s0 <- procedimento2(vetorB, 10)
            addi    $a0, $sp, 48    # $a0 <- endereço de vetorA
            li      $a1, 10         # $a1 <- 10
            jal     procedimento2   # $v0 <- procedimento2(vetorA, 10)
                    sub         $v0,  $v0,  $s0     #  $v0  <-  procedimento2(vetorA,  10)  -
procedimento2(vetorB, 10)
                sw          $v0,  0($sp)        # resultado  =  procedimento2(vetorA,  10)  -
procedimento2(vetorB, 10)
## epílogo ##
  #return resultado;
            lw      $s0, 88($sp)    # restauramos o registrador $s0
            lw      $ra, 92($sp)    # restauramos o endereço de retorno
            addiu   $sp, $sp, 96    # restauramos a pilha
            jr      $ra             # retornamos ao procedimento chamador
#}


################################################################################
#int main(void) {
main:

# mapa da pilha
# |    $ra    | $sp + 12
# |     n     | $sp + 8
# |     m     | $sp + 4
# | resultado | $sp + 0
################################################################################
## prólogo ##
  #int n;
  #int m;
  #int resultado;
            addiu   $sp, $sp, -16           #
            sw      $ra, 12($sp)            #
## corpo do programa ##
  #n = 5;
            li      $a0, 5                  # $a0 <- 5
            sw      $a0, 8($sp)             # n = 5
  #m = 3;
            li      $a1, 3                  # $a1 <- 3
            sw      $a1, 4($sp)             # m = 3
  #resultado = procedimento1(n, m);
            # n e m já estão nos registradores $a0 e $a1
            jal     procedimento1           # chamamos o procedimento1
## epílogo ##
  #return 0;
            lw      $ra, 12($sp)            # restauramos o endereço de retorno
            li      $v0, 5                  # retornamos 0
            addiu   $sp, $sp, 16            # restauramos a pilha
            jr      $ra                     # retornamos ao procedimento chamador
#}
```

3 – (1,0 ponto) Converta os números A = 20,3 e B = 30,4 para ponto fixo. Use 12 bits: 6 bits para a parte inteira e 6 bits para a parte fracionária. Os números são representados sem sinal. Faça a soma binária de (A+B) e a subtração (A-B). Mostre as operações com os vem-uns. Houve estouro (overflow) na operação?

Resposta:

Para A = 20,3.

Separamos a parte inteira I=20 da parte fracionária F=0,3. Usamos o algoritmo da divisão longa com a parte inteira I do número.

20 ÷ 2	=	10	resto	0
10 ÷ 2	=	5	resto	0
5 ÷ 2	=	2	resto	1
2 ÷ 2	=	1	resto	0
1 ÷ 2	=	0	resto	1

Tomamos os restos da última para a primeira divisão. Usamos 6 bits para representar a parte inteira em binário.

$I = 20 = 010100_2$

Usamos o algoritmo da multiplicação longa para a parte fracionária F=0,3 do número decimal.

0,3	× 2	=	0,6
0,6	× 2	=	1,2
0,2	× 2	=	0,4
0,4	× 2	=	0,8
0,8	× 2	=	1,6
0,6	× 2	=	1,2

Usamos como critério de parada deste algoritmo, o número de bits da parte fracionária da representação em ponto fixo.

$F = 0,3 \cong 0,010011_2$

Concatenamos a parte inteira binária com a parte fracionária binária[1].

$20,3 \cong 01\ 0100,0100\ 11_2$

Para B = 30,4.

Separamos a parte inteira I=30 da parte fracionária F=0,4. Usamos o algoritmo da divisão longa com a parte inteira I do número.

30 ÷ 2	=	15	resto	0
15 ÷ 2	=	7	resto	1
7 ÷ 2	=	3	resto	1
3 ÷ 2	=	1	resto	1

1 Não encontramos uma representação exata do número 20,3 com um número finito de bits. O valor encontrado é uma aproximação deste valor. O número $01\ 0100,0100\ 11_2$ equivale ao valor decimal 20,296875.

1 ÷ 2 = 0 resto 1

Tomamos os restos da última para a primeira divisão. Usamos 6 bits para representar a parte inteira em binário.

I = 30 = 11110_2

Usamos o algoritmo da multiplicação longa para a parte fracionária F=0,4 do número decimal.

0,4	× 2	=	0,6
0,8	× 2	=	1,6
0,6	× 2	=	1,2
0,2	× 2	=	0,4
0,4	× 2	=	0,8
0,8	× 2	=	1,6

Usamos como critério de parada deste algoritmo, o número de bits da parte fracionária da representação em ponto fixo.

F = 0,4 ≅ 0,0110 11_2

Concatenamos a parte inteira binária com a parte fracionária binária[2].

30,4 ≅ 01 1110,0110 01_2

A soma binária de A e B é realizada com os passos apresentado no texto numeros_ponto_fixo.pdf.

```
          011 1000 1001 10
  20,3     01 0100 0100 11
+ 30,4   + 01 1110 0110 01
  ------------------------
  50,7     11 0010 1011 00
```

O vai-um da coluna dos bits mais significativos é igual a 0. Não houve *overflow* ou estouro na operação[3].

No texto numeros_ponto_fixo.pdf temos os passos para a realização da subtração binária A – B.

```
          000 0000 0011 11
  20,3     01 0100 0100 11
- 30,4   + 10 0001 1001 10
  ------------------------
- 10,1 ?   11 0101 1110 10
```

O vai-um da coluna dos bits mais significativos é igual a 0. Nesta operação de subtração (realizada como a soma do minuendo com o subtraendo com todos os bits complementados e o vem-um inicial igual a 1) houve o estouro ou *overflow* na operação[4].

2 Não encontramos uma representação exata do número 30,4 com um número finito de bits. O valor encontrado é uma aproximação deste valor. O número 01 1110,0110 01_2 equivale ao valor decimal 30,390625.

3 A soma é igual a 50,687500 ≅ 50,7.

4 Observamos que o resultado da operação de subtração é negativo, não podendo ser representado como um número sem sinal.

4 – (2,0 pontos) Converta os números A = 20,3 e B = 30,4 para ponto fixo. Use 12 bits: 6 bits para a parte inteira e 6 bits para a parte fracionária. Os números são representados em complemento de 2. Faça a soma binária de (A+B) e a subtração (A-B). Mostre as operações com os vem-uns. Houve estouro (overflow) na operação? Escreva um trecho de código em assembly MIPS, para verificar se houve o estouro na operação de soma.

Resposta

Os números decimais A e B deste problema são iguais ao problema anterior. A representação dos números em ponto fixo em complemento de dois é igual ao problema anterior porque os números binários encontrados são positivos e possuem o bit mais significativo igual a 0.

A = 20,3 = 01 0100,0100 11_2
B = 30,4 = 01 1110,0110 01_2

Os números binários A e B são iguais neste problema e no problema anterior. As operações de soma e de subtração e os valores são iguais ao problema anterior: a operação de soma e de subtração não depende da representação do número, se é sem sinal ou complemento de 2. O resultado das operações são os mesmos mas a verificação do estouro ou overflow depende da representação. Usaremos a soma e a subtração do problema anterior.

Soma A + B.

```
          011 1000 1001 10
  20,3     01 0100 0100 11
+ 30,4   + 01 1110 0110 01
  ------------------------
  50,7     11 0010 1011 00
```

Na soma de A + B temos o estouro ou overflow da operação porque os bits vem-um e vai-um da coluna com os bits mais significativos são diferentes[5].

Subtração A - B.

```
          000 0000 0011 11
  20,3     01 0100 0100 11
- 30,4   + 10 0001 1001 10
  ------------------------
 -10,1     11 0101 1110 10
```

Na subtração binária A – B não houve estouro na operação: os bits vem-um e vai-um da coluna dos bits mais significativos são iguais[6].

5 Observamos que estamos somado dois números positivos e o resultado é um número negativo (o bit mais significativo do resultado é um).

6 O resultado da subtração é igual a 11 0101,1110 10_2 = -10,093750 ≅ -10,1

4 – (2,0 pontos) (a) Converta a instrução em linguagem de montagem **beq $s0**, **$s1**, **loop** para linguagem de máquina e (b) explique detalhadamente, com a ajuda da figura 2 e as tabelas 7 e 8, como a instrução em linguagem de máquina é executada pelo processador monociclo. A instrução **beq** está no endereço 0x0040000C. **Loop** é um rótulo para o endereço 0x00400000.

Resposta:

Vamos explicar como a instrução beq $s0, $s1, loop é executada por um processador monociclo.
O endereço da instrução é 0x0040000C. O endereço do desvio condicional é 0x00400000. Utilizaremos o diagrama de blocos do processador da figura 1 e as tabelas dos sinais gerados pela unidade de controle do processador e a tabela com os sinais de controle da operação da ULA e respectivas operações. Traduzimos a instrução de *assembly* para linguagem de máquina, usando o apêndice B do livro texto (Patterson e Hennessy, 2014).

beq $s0, $s1, loop = beq $16, $17, loop → 0x1211FFFC

op	rs	rt	imm
00 0100	1 0000	1 0001	1111 1111 1111 1100
0x04	$16	$17	0xFFFC
Beq	$s0	$s1	-4

Para calcularmos o valor imediato imm da instrução, subtraímos o valor de PC+4 do endereço de desvio (loop) e dividimos o valor por 4.

$$\frac{loop-(PC+4)}{4}=\frac{0x00400000-0x00400010}{4}=\frac{-16}{4}=-4=0xFFFC$$

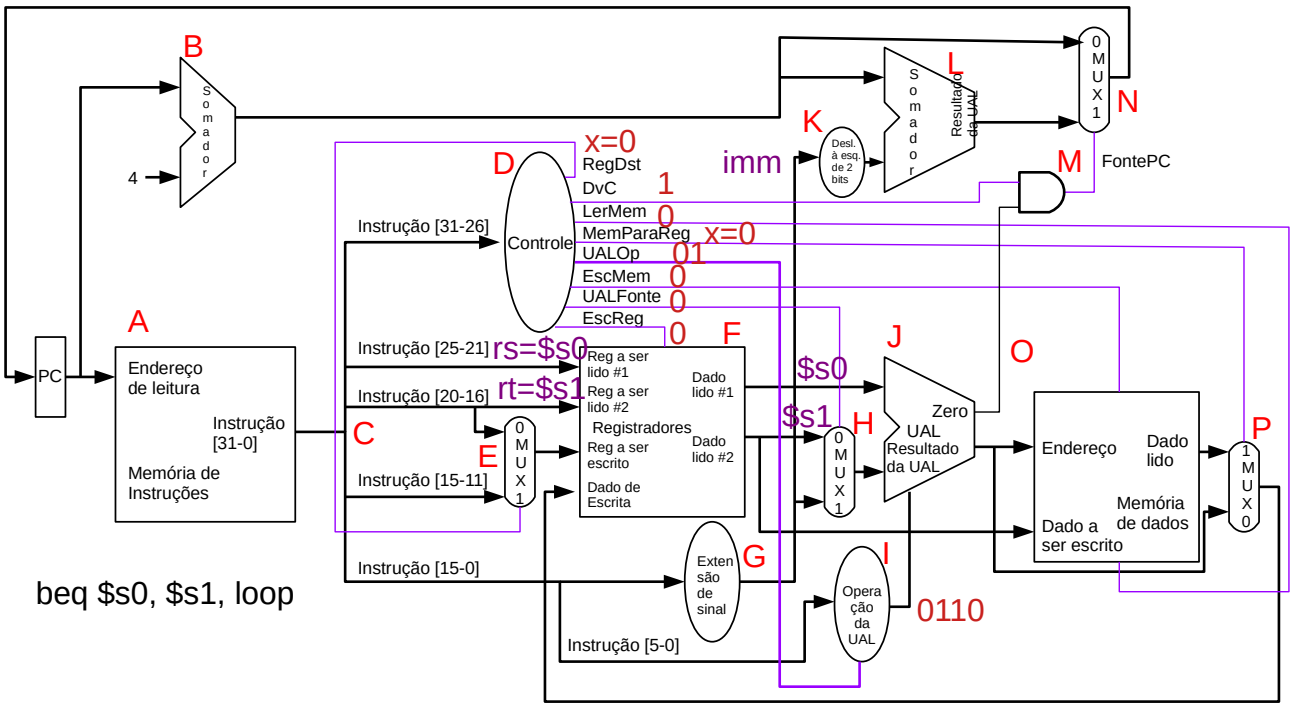

Figura 1. Ilustração com o diagrama de blocos simplificado do caminho de dados e o caminho de controle um processador MIPS.

A execução das instruções no processador monociclo ocorre no intervalo de um ciclo de relógio, entre bordas de subida.

1. Na borda de subida do sinal de relógio o novo valor do PC é carregado no registrador PC, contador de programa ou *program counter*. Após um intervalo de tempo, um tempo de propagação, o novo valor de PC aparece na saída do registrador.

2. O sinal de PC (0x0040000C) é propagado pelo barramento de endereços até a entrada de endereço da memória de instruções (A) e a entrada do somador (B).

3. Após um intervalo de tempo temos na saída da memória de instruções (A), a instrução I = 0x1211FFFC.

4. Na saída do somador (B), o valor de PC é somado com 4. Este novo endereço será chamado de PC+4 (0x00400010).

5. A instrução I na saída da memória de instruções é separada em vários campos (barramentos indicados por C na figura 1):
Instrução[31-26] = OP (código de operação da instrução, ou opcode) = 0x04 = 000100
Instrução[25-21] = rs= $s0 = $16 = 10000_2
Instrução[20-16] = rt = $s1 = $17 = 10001_2
Instrução[15-11] = rd = $ra = $31 = 11111_2
Instrução[15-0] = imm (valor imediato) = -4 = 0xFFFC = 1111111111111100_2
Instrução[5-0] = funct = 0x3C = 111100_2

6. O campo com o código de operação da instrução (opcode = 0x04, Instrução[31-26]) vai para a entrada do subsistema de controle (D). Este circuito gera os sinais de controle para executar a instrução pelo processador.

Os sinais gerados são encontrados com o auxílio da tabela 7:
RegDst = X = 0
UalFonte = 0
MemParaReg = X = 0
EscReg = 0
LerMem = 0
EscMem = 0
DvC = 1
UALOp[1:0] = 01
O valor X é um sinal don't care, um sinal que pode ter qualquer valor, ou 0 ou 1. Vamos usar para X o valor 0. Estes sinais de controle propagam-se pelo circuito.

7. O sinal de controle do multiplexador (E) é RegDst = 0. A entrada 0 com o endereço de $s1 = $17 = Instrução[20-16] vai para a saída do multiplexador e para a entrada do registrador a ser escrito do banco de registradores(F).

8. Nas entradas registrador a ser lido #1 e registrador a ser lido #2 temos os endereços dos registradores rs = $s0 e rt = $s1.

9. Na saída do banco de registradores (F), Dado Lido #1 e Dado Lido #2, temos o conteúdo do registrador rs = $s0 e rt = $s1, respectivamente.

10. O valor imediato imm (instrução[15-0]) tem o sinal estendido pelo circuito G. Este campo passa de 16 bits para 32 bits. O valor deste campo é -4.

11. Na saída do multiplexador (H), temos o valor da saída Dado lido #2 que é o valor do registrador $s1.

12. O circuito de controle da unidade lógica e aritmética ou ULA (I) recebe o sinal UALOp[1:0] e o campo funct (Instrução[5-0]). Da tabela 8, para UALOp[1:0]=01, o código de operação enviado para o controle da UAL é 0110, representando uma subtração das entradas da ULA.

13. A UAL (J) recebe em suas entradas dois operandos, o valor do registrador rs = $s0 e o valor do registrador rt=$s1. Recebe do circuito (I) o valor 0110 na entrada de controle. A UAL realiza a operação de subtração dos operandos de entrada. A saída da UAL é igual ao valor a $s0 - $s1. A saída zero da UAL pode ser 0 ou 1, dependendo do resultado da subtração. Se $s0 é igual a $s1, a saída será 1. Nos outros casos, quando os registradores possuírem valores diferentes, a saída será 0.

14. No somador indicado por (L), temos a soma de PC+4 com o valor imediato multiplicado por 4: (PC+4) +

(imm«2). Neste caso, na saída do somador (L) temos o valor 0x00400000. A multiplicação do valor imediato por 4 é realizado pelo circuito (K), por meio de um deslocamento lógico de 2 bits para a esquerda.

15. Na porta AND (H), a saída poderá ser 1 ou 0, dependendo do valor da saída zero da UAL.

16. Na saída do multiplexador (N) podemos ter o valor de PC+4 (0x00400010) ou o endereço de desvio condicional (0x00400000), dependendo do valor da saída da porta AND (M). Se os registradores $s0 e $s1 forem iguais, a saída zero da UAL (J) e a saída da porta AND (M) serão iguais a 1 e a saída do multiplexador (N) será igual ao endereço de desvio condicional 0x00400000. Se os registradores forem diferentes, a saída zero da UAL e a saída da porta AND serão iguais a zero e a saída do multiplexador (N) será igual a PC+4 = 0x00400010. Um destes valores, ou PC+4 ou o endereço de desvio condicional, será gravado no contador de programa (PC), na próxima borda de subida do sinal de relógio.

17. Na memória de dados (O), nenhuma operação será realizada pois os sinais LerMem e EscMem são iguais a zero.

18. No banco de registradores, nenhum registrador será alterado porque o sinal EscReg = 0.

19. Na próxima borda de subida do sinal de relógio, o valor no registrador PC é atualizado para PC+4 (0x00400010) se os registradores $s0 e $s1 forem diferentes ou para o endereço de desvio (0x00400000) se estes dois registradores forem iguais. Terminamos a execução da instrução atual e uma nova instrução será executada.

Tabela 7 – Instruções e valores dos sinais na unidade de controle da figura 2.

Controle	**Nome do Sinal**	**Formato R (0)**	**Lw (35)**	**Sw (43)**	**Beq (4)**
Entradas	OP5	0	1	1	0
	OP4	0	0	0	0
	OP3	0	0	1	0
	OP2	0	0	0	1
	OP1	0	1	1	0
	OP0	0	1	1	0
Saídas	RegDst	1	0	X	X
	UalFonte	0	1	1	0
	MemParaReg	0	1	X	X
	EscReg	1	1	0	0
	LerMem	0	1	0	0
	EscMem	0	0	1	0
	DvC	0	0	0	1
	UALOp1	1	0	0	0
	UALOp0	0	0	0	1

Tabela 8 – Operação da UAL para a combinação da UALOp e campo de função.

UALOp		Campo de Função						Operação da UAL	
UALOp1	UALOp1	F5	F4	F3	F2	F1	F0		
0	0	X	X	X	X	X	X	010	soma
X	1	X	X	X	X	X	X	110	subtração
1	X	X	X	0	0	0	0	010	soma
1	X	X	X	0	0	1	0	110	subtração
1	X	X	X	0	1	0	0	000	AND
1	X	X	X	0	1	0	1	001	OR
1	X	X	X	1	0	1	0	111	Slt