**Tópicos:**

* Introdução Sistemas de Computação de Uso Geral
* A arquitetura MIPS

# Questões:

1. ***Quais são os 3 blocos fundamentais de um sistema computacional?***

Resposta: CPU, Memória, Unidades de I/O

1. ***Quais são os 3 principais blocos funcionais na arquitetura de um CPU?***

Resposta: Unidade de controlo, Secção de dados, ??????????

1. ***Qual a função do registo Program Counter?***

Resposta: Registo que contém o endereço de memória onde está armazenada a próxima instrução a executar.

1. ***Descreva de forma sucinta a função de um compilador.***

Resposta: Transformação de linguagem de programação para linguagem máquina.

1. ***Descreva de forma sucinta a função de um assembler.***

Resposta: Transformação de código em assembly para código máquina.

1. ***Quantos registos internos de uso geral tem o MIPS?***

Resposta: 32 registos.

1. ***No MIPS, qual a dimensão, em bits, que cada um dos registos internos pode armazenar?***

Resposta: 32 bits.

1. ***Qual a sintaxe, em Assembly do MIPS, de uma instrução aritmética de soma ou subtração?***

Resposta: add $1,$2,$3 # $1 endereço onde será armazenado a soma de $2 com $3

sub $1,$2,$3 # $1 endereço onde será armazenado a diferença entre $2 e $3

1. ***O que distingue a instrução srl da instrução sra do MIPS?***

Resposta: srl(SHIFT RIGHT LOGIC) – Para variável do tipo **unsigned**

sra(SHIFT RIGHT ARITMETIC)- Para variável do tipo **signed**

1. ***Se $5=0x81354AB3, qual o resultado armazenado no registo destino, expresso em hexadecimal, das instruções:***

$5 - 0x81354AB3 – 1000 0001 0011 0101 0100 1010 1011 0011

* 1. **srl $3,$5,1**

Resposta:

$3 = 0100 0000 1001 1010 1010 0101 0101 1001

$3= 0x409AA559

* 1. **sra $4,$5,1**

Resposta:

$4=1100 0000 1001 1010 1010 0101 0101 1001

$4=0xC09AA559

1. ***System calls:***
   1. ***O que é um system call?***

Resposta: Uma **system call** é uma rotina que permite a um programa do utilizador fazer pedidos de ações que requerem privilégios especiais.

* 1. ***No MIPS, qual o registo usado para identificar o system call a executar?***

Resposta: $v0

* 1. ***Qual o registo ou registos usados para passar argumentos para os systems calls?***

Resposta: $a0,$a1,$a2,$a3

* 1. ***Qual o registo usado para obter o resultado devolvido por um system call (nos casos em que isso se aplica)?***

Resposta: $v0

1. ***Em Arquitetura de Computadores, como definiria o conceito de endereço?***

Resposta: Um endereço é um identificador único para um local de memória que pode ser acessado.

1. ***Defina o conceito de espaço de endereçamento de um processador?***

Resposta: Gama total de endereços que o CPU consegue referenciar.

1. ***Como se organiza internamente um processador? Quais são os blocos fundamentais da secção de dados? Para que serve a unidade de controlo?***

Resposta à pergunta 1: Um processador é organizado através dos blocos: Secção de dados(datapath) e unidade de controlo.

Resposta à pergunta 2: Multiplexers, Unidade Aritmética e Lógica(ALU) e Registos Internos.

Resposta à pergunta 3: A unidade de controlo é responsável pela coordenação dos elementos do datapath durante a execução do programa .

Uma imagem com texto

Descrição gerada automaticamente

1. ***Qual é o conceito fundamental por detrás do modelo de arquitetura "stored-program"?***

Resposta: As instruções são representadas da mesma forma que os números.

Os programas são armazenadas em memória, para serem lidos e escritos, assim como a memória.

1. ***Como se codifica uma instrução? Que informação fundamental deverá ter o código máquina de uma instrução?***

Resposta: A codificação de uma instrução para ser realizada deverá ser levado em conta os seguintes aspetos: a operação a realizar; qual a localização dos operandos(se existirem); onde colocar o resultado; qual a próxima instrução a executar.

Uma imagem com texto

Descrição gerada automaticamente

1. ***Descreva pelas suas próprias palavras o conceito de ISA (Instruction Set Architecture).***

Resposta: A parte que um programador consegue controlar/manipular.

1. ***Independentemente do modelo da arquitetura, identifique quantas e quais as classes de instruções que compões o conjunto de instruções executáveis por um processador dessa arquitetura?***

Resposta: 3 classes de instruções: Processamento, transferência de informação e controlo de fluxo de execução.

Uma imagem com texto

Descrição gerada automaticamente

1. ***O que carateriza e distingue as arquiteturas do tipo "register-memory" e "load-store"? De que tipo é a arquitetura MIPS?***

Resposta: A arquitetura “register-memory” pode realizar operações com registos internos e/ou memória, a arquitetura “load-store” apenas realiza com registos internos.

Uma imagem com texto

Descrição gerada automaticamente

1. ***O ciclo de execução de uma instrução é composto por uma sequência ordenada de operações. Quantas e quais são essas operações (passos de execução)?***

Resposta: Um ciclo de execução de instrução é composto por um conjunto de 5 operações: Instruction fetch; Instruction decode; operand fetch; execute; store result, respetivamente.

1. ***Como se designa o barramento que permite identificar, na memória, a origem ou o destino da informação transferida?***

Resposta: Address Bus

1. ***Qual a finalidade do barramento normalmente designado por Data Bus?***

Resposta: O data bus tem como objetivo a transferência de informação( CPU – Memória ; CPU – Input/Output)

1. ***Os processadores da arquitetura hipotética ZWYZ possuem 4 registos internos e todas as instruções são codificadas usando 24 bits. Num dos formatos de codificação existem 5 campos: um OpCode com 5 bits, três campos para identificar registos internos em operações aritméticas e lógicas e um campo para codificar valores constantes imediatos em complemento para dois. Qual a gama de representação destas constantes?***

Resposta: 13 bits

Uma imagem com texto

Descrição gerada automaticamente

1. ***A arquitetura hipotética ZPTZ tem um barramento de endereços de 32 bits e um barramento de dados de 16 bits. Se a memória desta arquitetura for bit\_addressable:***
   1. ***Qual a dimensão do espaço de endereçamento desta arquitetura?***

Resposta: A dimensão do espaço de endereçamento é (2^32) bits.

* 1. ***Qual a dimensão máxima da memória, expressa em bytes, suportada por esta arquitetura?***

Resposta:

2^32 X 2^4 = 2 ^36 bits = 2^36 bits / 8 = 2 ^ 36 / 2 ^ 3 = 2 ^ 33 bytes

1. ***Considere agora uma arquitetura em que o respetivo ISA especifica uma organização de memória do tipo word-addressable, em que a dimensão da word é 32 bits. Tendo o espaço de endereçamento do processador 24 bits, qual a dimensão máxima de memória que este sistema pode acomodar se expresso em bytes?***

Resposta: Número de endereços: 2^24

Número de bits em cada endereço: 32 bits = 4 bytes

Espaço da memória: 2^24 \* 4 = 2^24 \* 2^2 = 2^26 bytes

1. ***Relativamente à arquitetura MIPS:***
   1. ***Com quantos bits são codificadas as instruções no MIPS?***

Resposta: 32 bits

* 1. ***O que diferencia o registo $0 dos restantes registos de uso geral?***

Resposta: O registo $0 tem sempre o valor 0 e é inalterável.

* 1. ***Qual o endereço do registo interno do MIPS a que corresponde a designação lógica $ra?***

Resposta: Registo número 31

1. ***No MIPS, um dos formatos de codificação de instruções é designado por R:***
   1. ***Quais os campos em que se divide este formato de codificação?***

Resposta: OPCODE,RS,RT,RD,SHAMT,FUNCT.

* 1. ***Qual o significado de cada um desses campos?***

Resposta: OPCODE – Sempre zero em instruções tipo R

RS – Endereço do registo do 1º operando-fonte

RT – Endereço do registo do 2º operando-fonte

RD – Endereço do registo destino

SHAMT – Apenas útil para instruções de deslocamento

FUNCT – Código da operação a realizar

* 1. ***Qual o valor do campo opCode nesse formato?***

Resposta: ZERO

* 1. ***O que faz a instrução cujo código máquina é: 0x00000000?***

Resposta:

OPCODE : 000000 ; RS – 00000 ; RT – 00000 ; RD – 00000 ; SHAMT – 00000 ; FUNCT – 000000

sll $0,$0,0

1. ***O símbolo ”>>“ da linguagem C significa deslocamento à direita e é traduzido em assembly por srl ou sra (no caso do MIPS). Dê exemplos de casos em linguagem C em que o compilador gera um srl e exemplos em que gera um sra.***

Resposta:

Exemplo para srl:

**unsigned** **int** **var**;  
**var** = **var** >> 2;

Exemplo para sra:

**int** **var**;  
 **var** = **var** >> 2 ;

1. ***Qual a instrução nativa do MIPS em que é traduzida a instrução virtual "move $4,$15"?***

Resposta: addu $4,$0,$15

1. ***Determine o código máquina das seguintes instruções (verifique a tabela na última página):***
   1. ***xor $5,$13,$24***

Resposta: xor é formato R , logo o opcode é 0

OPCODE – 000000

RS - $13 – 01101

RT - $24 – 11000

RD - $5 – 00101

SHAMT – 00000 – não é uma instrução de deslocamento

FUNCT – 100110 - código da função(última página)

Juntando os elementos – 000000 01101 11000 00101 00000 100110

0000 0001 1011 1000 0010 1000 0010 0110

Em hexadecimal – 0x01B82826

* 1. ***sub $25,$14,$8***

Resposta: sub é formato R , logo o opcode é 0

OPCODE – 000000

RS - $14 – 10010

RT - $8 – 01000

RD - $25 – 11101

SHAMT – 00000 – não é uma instrução de deslocamento

FUNCT – 100010 - código da função(última página)

Juntando os elementos – 000000 10010 01000 11101 00000 100010

0000 0010 0100 1000 1110 1000 0010 0010

Em hexadecimal – 0x0248E822

* 1. ***sll $3,$9,7***

Resposta: sll é formato R , logo o opcode é 0

OPCODE – 000000

RS – 00000

RT - $9 – 01001

RD - $3– 00011

SHAMT – 00111 –instrução de deslocamento – valor a deslocar

FUNCT – 000000 - código da função(última página)

Juntando os elementos – 000000 00000 01001 00011 00111 000000

0000 0000 0000 1001 0001 1001 1100 0000

Em hexadecimal – 0x000919C0

1. ***Traduza para instruções Assembly do MIPS a seguinte expressão aritmética, supondo que x e y são inteiros e residentes em $t2 e $t5, respetivamente (apenas pode usar instruções nativas e não deverá usar a instrução de multiplicação):***

**y = -3 \* x + 5;**

Resposta:

add $t0,$t2,$t2

add $t2,$t2,$t0

add $t2,$t2,-1

addi $t5,$t2, 5

1. ***Traduza para instruções assembly do MIPS o seguinte trecho de código:***

**int a, b, c; //a:$t0, b:$t1, c:$t2 unsigned int x, y, z; //x:$a0, y:$a1, z:$a2 z = (x >> 2) + y;**

**c = (a >> 5) – 2 \* b;**

Resposta:

srl $a0,$a0,2

add $a2,$a0,$a1

sra $t0,$t0,5

mul $t1,$t1,2

sub $t2,$t0,$t1

1. ***Considere que as variáveis g, h, i e j são conhecidas e podem ser representadas por uma variável de 32 bits num programa em C. Qual a correspondência, em linguagem C, às seguintes instruções:***
   1. **add h, i, g #**

Resposta: h = i + g;

* 1. **addi j, j, 1 #**

**add h, h, j #**

Resposta: j = j+1;

h = h + j ;

1. ***Assumindo que g=1, h=2, i=3 e j=4 qual o valor destas variáveis no final das sequências das alíneas da questão anterior?***

Resposta para alínea a: g=1; h=4;i=3; j=4.

Resposta para alínea b: g=1; h=7; i=3; j=5.

1. ***Descreva a operação realizada pela instrução assembly "slt" e quais os resultados possíveis?***

Resposta: A instrução slt vai adicionar o valor 1 ao registo destino se o operando 1 for menor que o operando 2, caso contrário , o registo destino fica com o valor 0.

1. ***Qual o valor armazenado no registo $1 na execução da instrução "slt $1, $3, $7", admitindo que:***
   1. ***$3=5 e $7=23***

Resposta: $1 = 1

* 1. ***$3=0xFE e $7=0x913D45FC***

Resposta : $1 = 0

1. ***Com que registo implícito comparam as instruções "bltz", "blez", "bgtz" e "bgez"?***

Resposta: Com o registo $0.

1. ***Decomponha em instruções nativas do MIPS as seguintes instruções virtuais:***
   1. **blt $15,$3,exit**

Resposta:

slt $1, $15, $3  
 bne $1, $0, **exit**

* 1. **ble $6,$9,exit**

Resposta:

slt $1, $9, $6  
 beq $1, $0, **exit**

* 1. **bgt $5,0xA3,exit**

Resposta:

addi $1, $0, 0x000000A3  
 slt $1, $1, $5  
 bne $1,$0,**exit**

* 1. **bge $10,0x57,exit**

Resposta:

slti $1,$10,0x57  
 beq $1,$0,**exit**

* 1. **blt $19,0x39,exit**

Resposta:

slt $1,$9,0x39  
 bne $1,$0,**exit**

* 1. **ble $23,0x16,exit**

Resposta:

addi $1, $23, 0xFFFFFFFF  
slti $1, $1, 0x16  
bne $1, $0, **exit**

1. ***Na tradução de C para assembly, quais as principais diferenças entre um ciclo "while(…){…}" e um ciclo "do{…}while(…);" ?***

Uma imagem com texto

Descrição gerada automaticamente

Como em C, o **while** verifica a condição no início e o **do while**

Verifica no fim do trecho de código.

1. ***Traduza para assembly do MIPS os seguintes trechos de código de linguagem C (admita que a, b e c***

***residem nos registos $4, $7 e $13, respetivamente):***

**a. if(a > b && b != 0) c = b << 2;**

**else**

**c = (a & b) ^ (a | b);**

**Uma imagem com texto

Descrição gerada automaticamente**

**b. if(a > 3 || b <= c) c = c – (a + b);**

**else**

**c = c + (a – 5);**

**Uma imagem com texto

Descrição gerada automaticamente**

1. ***Como se designa o modo de endereçamento usado pelo MIPS para ter acesso a palavras residentes na memória externa?***

Resposta: Load – Store, ou seja, não é possível operar diretamente com a memória.

1. ***Na instrução "lw $3,0x24($5)" qual a função dos registos $3, $5 e da constante 0x24?***

Resposta: O $3 irá guardar o valor que está no endereço de memória igual a $5 + 0x24.

1. ***Qual é o tipo de codificação das instruções de acesso à memória no MIPS? Descreva o seu formato e o significado de cada um dos seus campos?***

Resposta: A codificação de instruções de acesso à memória é de formato I.

Vejamos as componentes e o seu significado no formato I.

OPCODE: Código da função a executar – 6 bits

RS – Endereço do registo fonte – 5 bits

RT – Endereço do registo destino – 5 bits

OFFSET – Valor que será somado ao registo fonte para acesso à memória – 16 bits

1. ***Qual a diferença entre as instruções "sw" e "sb"?***

Resposta:

SW – Armazena 4 bytes

SB – Armazena 1 byte

1. ***O que distingue as instruções "lb" e "lbu"?***

Resposta:

LB – load byte ( offset signed)

LBU – load byte unsigned (offset unsigned)

1. ***O que acontece quando uma instrução lw/sw tenta aceder a um endereço que não é múltiplo de 4?***

Resposta: Quando o MIPS tentar acessar à memória verifica que o endereço é inválido e gera uma execeção terminando o programa.

Uma imagem com texto

Descrição gerada automaticamente

1. Traduza para *assembly* do MIPS os seguintes trechos de código de linguagem C (atribua registos internos para o armazenamento das variáveis **i** e **k** ) :
   1. **int i, k;**

**for(i=5, k=0; i < 20; i++, k+=5);**

**Uma imagem com mesa

Descrição gerada automaticamente**

**b. int i=100, k=0; for( ; i >= 0; )**

**{**

**i--;**

**k -= 2;**

**}**

**Uma imagem com mesa

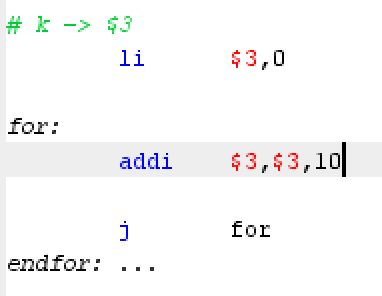
Descrição gerada automaticamente**

**c. unsigned int k=0; for( ; ; )**

**{**

**k += 10;**

**}**

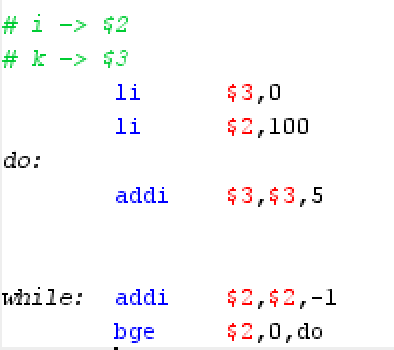
****

**d. int k=0, i=100; do**

**{**

**k += 5;**

**} while(--i >= 0);**



1. ***Sabendo que o OpCode da instrução "lw" é 0x23, determine o código máquina, expresso em hexadecimal, da instrução "lw $3,0x24($5)".***

Resposta: 0x8CA30024

OPCODE : 0x23 -> 10 0011

RS – 00101

RT – 00011

OFFSET – 0000 0000 0010 0100

Juntando : 10 0011 00101 00011 0000 0000 0010 0100 =

1000 1100 1010 0011 0000 0000 0010 0100 =

0x8CA30024

1. ***Suponha que o conteúdo da memória externa foi inicializada, a partir do endereço 0x10010000, com os valores 0x01,0x02,0x03,0x04,0x05,0x06,0x07 e assim sucessivamente. Suponha ainda que $3=0x1001 e $5=0x10010000. Qual o valor armazenado no registo destino após a execução da instrução "lw $3,0x24($5)" admitindo uma organização de memória little endian?***

Resposta: 0x28272625

Uma imagem com texto

Descrição gerada automaticamente

1. Considere as mesmas condições da questão anterior. Qual o valor armazenado no registo destino pelas instruções:
   1. **lbu $3,0xA3($5)**

Resposta: Utilizando o mesmo procedimento , $3 = 0xA4

**b. lb $4,0xA3($5)**

1. Quantos *bytes* são reservados no segmento de dados da memória por cada uma das seguintes diretivas:
   1. **L1: .asciiz "Aulas5&6T"**

Resposta: 10 bytes , 9 caracteres + caracter terminal. Cada caracter ocupa 1 byte.

* 1. **L2: .byte 5,8,23**

Resposta: 3 bytes

* 1. **L3: .word 5,8,23**

Resposta: Cada word ocupa 4 bytes . 4 bytes X 3 words = 12 bytes.

* 1. **L4: .space 5**

Resposta: 5 bytes

1. ***Desenhe esquematicamente a memória e preencha-a com o resultado das diretivas anteriores admitindo que são interpretadas sequencialmente pelo Assembler.***

Resposta: Admitindo que a memória começa no endereço 0x10010000

|  |  |
| --- | --- |
| 0x10010000 – 0x10010003 | a l u A |
| 0x10010004  -  0x10010007 | 6 & 5 s |
| 0x10010008  -  0x100100B | 8 5 /0 T |
| 0x1001000C  -  0X1001000F | 0 0 0 23 |
| 0x10010010  -  0x10010013 | 5 |
| 0x100100014  -  0x10010017 | 8 |
| 0x10010018  -  0x1001001B | 23 |

1. ***Supondo que "L1:" corresponde ao endereço inicial do segmento de dados, e que esse endereço é***

***0x10010000, determine os endereços a que correspondem os labels "L2:", "L3:" e "L4:".***

Resposta: L2: 0x100100A

L3: 0x1001010

L4: 0x100101C

1. ***Suponha que "b" é um array declarado como "int b[25];":***
   1. ***Como é obtido, em C, o endereço inicial do array, i.e., o endereço a partir do qual está armazenado o seu primeiro elemento?***

Resposta: &b[0]

* 1. ***Supondo uma memória "byte-addressable", como é obtido, em assembly o endereço do elemento "b[6]"?***

Resposta: la $t0,a

lb $t1,6($t0)

1. ***O que é codificado no campo offset do código máquina das instruções "beq/bne" ?***

Resposta: Endereço-alvo – Program Counter

1. ***A partir do código máquina de uma instrução "beq/bne", como é formado o endereço-alvo (Branch Target Address)?***

Resposta: ?????

1. ***Qual o formato de codificação de cada uma das seguintes instruções: "beq/bne", "j", "jr"?***

Resposta: BEQ/ BNE – formato I

J/JR – formato J

1. ***A partir do código máquina de uma instrução "j", como se obtém o endereço-alvo (Jump Target Address)?***

Resposta: O endereço-alvo obtém-se a partir dos 26 bits menos significativos do código máquina, acrescentando 2 bits à direita.

1. ***Dada a seguinte sequência de declarações:***

***int b[25]; int a;***

***int \*p = b;***

***Identifique qual ou quais das seguintes atribuições permitem aceder ao elemento de índice 5 do array***

***"b":***

|  |  |  |  |
| --- | --- | --- | --- |
| **a = b[5];** | **a = \*p + 5;** | **a = \*(p + 5);** | **a = \*(p + 20);** |

Resposta: Opção 1,3.

1. Assuma que as variáveis **f***,* **g***,* **h***,* **i** e **j** correspondem aos registos **$t0**, **$t1**, **$t2**, **$t3** e **$t4** respetivamente. Considere que o endereço base dos *arrays* de inteiros **A** e **B** está contido nos registos **$s0** e **$s1**. Considere ainda as seguintes expressões:

**f = g + h + B[2] j = g - A[B[2]]**

* 1. Qual a tradução para *assembly* de cada uma das instruções C indicadas?
  2. Quantas instruções *assembly* são necessárias para cada uma das instruções C indicadas? E quantos registos auxiliares são necessários?
  3. Considerando a tabela seguinte que representa o conteúdo byte-a-byte da memória, nos endereços correspondentes aos *arrays* A e B, indique o valor de cada elemento dos *arrays* assumindo uma organização *little endian*.

|  |  |
| --- | --- |
| Endereço | Valor |
| A+12 | … |
| A+11 | 0x00 |
| A+10 | 0x00 |
| A+9 | 0x00 |
| A+8 | 0x01 |
| A+7 | 0x22 |
| A+6 | 0xED |
| A+5 | 0x34 |
| A+4 | 0x00 |
| A+3 | 0x00 |
| A+2 | 0x00 |
| A+1 | 0x00 |
| A+0 | 0x12 |

|  |  |
| --- | --- |
| Endereço | Valor |
| B+12 | … |
| B+11 | 0x00 |
| B+10 | 0x00 |
| B+9 | 0x00 |
| B+8 | 0x02 |
| B+7 | 0x00 |
| B+6 | 0x00 |
| B+5 | 0x50 |
| B+4 | 0x02 |
| B+3 | 0xFF |
| B+2 | 0xFF |
| B+1 | 0xFF |
| B+0 | 0xFE |

|  |
| --- |
| A[0]= |
| A[1]= |
| A[2]= |

|  |
| --- |
| B[0]= |
| B[1]= |
| B[2]=d |

* 1. Assumindo que **g = -3** e **h = 2**, qual o valor final das variáveis **f** e **j**?

1. Pretende-se escrever uma função para a troca do conteúdo de duas variáveis (**a** e **b**).

Isto é, se, antes da chamada à função, **a=2** e **b=5**, então, após a chamada à função, os valores de **a** e **b**

devem ser: **a=5** e **b=2**

Uma solução incorreta para o problema é a seguinte:

**void troca(int x, int y)**

**{**

**int aux; aux = x; x = y; y = aux;**

**}**

Identifique o erro presente no trecho de código e faça as necessárias correções para que a função tenha o comportamento pretendido.

1. Na instrução "**jr $ra**", como é obtido o endereço-alvo?
2. Qual é o menor e o maior endereço para onde uma instrução "**j**", residente no endereço de memória

**0x5A18F34C**, pode saltar?

1. Qual é o menor e o maior endereço para onde uma instrução "**beq**", residente no endereço de memória

**0x5A18F34C**, pode saltar?

1. Qual é o menor e o maior endereço para onde uma instrução "**jr**", residente no endereço de memória

**0x5A18F34C** pode saltar?

1. Qual a gama de representação da constante nas instruções aritméticas imediatas (e.g. **addi**)?
2. Qual a gama de representação da constante nas instruções lógicas imediatas (e.g. **andi**)?
3. Por que razão não existe, no ISA do MIPS, uma instrução nativa que permita manipular diretamente uma constante de 32 bits?
4. Como é que, no *assembly* do MIPS, se podem manipular constantes de 32 bits?
5. Apresente a decomposição em instruções nativas das seguintes instruções virtuais:
   1. **li $6,0x8B47BE0F**

lui $1,0Xffff8B47

ori $6, $1, 0x0000BE0F

**b. xori $3,$4,0x12345678**

lui $1, 0x00001234

ori $1, $1, 0x00005678

xor $3,$4,$1

**c. addi $5,$2,0xF345AB17**

lui $1, 0xFFFFF345

ori $1,$1,0x0000AB17

add $5, $2, $1

**d. beq $7,100,L1**

addi $1, $0, 100

beq $1,$7,L1

**e. blt $3,0x123456,L2**

lui $1, 0x00000012

ori $1, $1, 0x00003456

slt $1, $3, $1

bne $1,$0,L2

1. ***O que é uma sub-rotina?***

Sub-rotinas são funções existentes num determinado programa que poderão ser chamadas durante o programa uma ou mais vezes.

1. ***Qual a instrução do MIPS usada para evocar uma sub-rotina?***

JAL – jump and link

1. ***Por que razão não pode ser usada a instrução "j" para evocar uma sub-rotina?***

Porque o jal armazena o valor atual do Program Counter no registo $ra, para que o programa saiba retornar ao endereço onde ficou depois de a sub-rotina terminar .

1. ***Quais as operações que são sequencialmente realizadas na execução de uma instrução de evocação de uma sub-rotina?***
2. Qual o número e nome virtual do registo associado à execução dessa instrução?
3. No caso de uma sub-rotina ser simultaneamente chamada e chamadora (sub-rotina intermédia) que operações é obrigatório realizar nessa sub-rotina?
4. ***Qual a instrução usada para retornar de uma sub-rotina?***

jr $ra

1. Que operação fundamental é realizada na execução dessa instrução?
2. O que é uma *stack* e qual a finalidade do *stack pointer*?
3. Como funcionam as operações de ***push*** e ***pop***?
4. Por que razão as *stacks* crescem normalmente no sentido dos endereços mais baixos?
5. Quais as regras para a implementação em software de uma *stack* no MIPS?
6. Qual o registo usado, no MIPS, como *stack pointer*?
7. De acordo com a convenção de utilização de registos no MIPS:
   1. Que registos são usados para passar parâmetros e para devolver resultados de uma sub-rotina?
   2. Quais os registos que uma sub-rotina pode livremente usar e alterar sem necessidade de prévia salvaguarda?
   3. Quais os registos que uma sub-rotina chamadora tem a garantia que a sub-rotina chamada não altera?
   4. Em que situação devem ser usados registos “**$sn**”?
   5. Em que situação devem ser usados os restantes registos: **$tn**, **$an** e **$vn**?
8. De acordo com a convenção de utilização de registos do MIPS:
   1. Que registos podem ter de ser copiados para a stack numa sub-rotina intermédia?
   2. Que registos podem ter de ser copiados para a stack numa sub-rotina terminal?
9. Para a função com o protótipo seguinte indique, para cada um dos parâmetros de entrada e para o valor devolvido, qual o registo do MIPS usado para a passagem dos respetivos valores:

**char fun(int a, unsigned char b, char \*c, int \*d)**;

1. Para uma codificação em complemento para 2, apresente a gama de representação que é possível obter com **3**, **4**, **5**, **8** e **16** bits (indique os valores-limite da representação em binário, hexadecimal e em decimal com sinal e módulo).
2. Traduza para *assembly* do MIPS a seguinte função “**fun1()**”, aplicando a convenção de passagem de parâmetros e salvaguarda de registos:

**char \*fun2(char \*, char);**

**char \*fun1(int n, char \*a1, char \*a2)**

**{**

**int j = 0; char \*p = a1;**

**do**

**{**

**if((j % 2) == 0)**

**fun2(a1++, \*a2++);**

**} while(++j < n);**

**\*a1='\0'; return p;**

**}**

1. Determine a representação em complemento para 2, com 16 bits, das seguintes quantidades decimais: 5, -3, -128, -32768, 31, -8, 256, -32
2. Determine o valor em decimal representado por cada uma das quantidades seguintes, supondo que estão codificadas em complemento para 2 com 8 bits:

0b00101011, 0xA5, 0b10101101, 0x6B, 0xFA, 0x80

1. Determine a representação das quantidades do exercício anterior em hexadecimal com 16 bits (também codificadas em complemento para 2).
2. Como é realizada a deteção de *overflow* em operações de adição com quantidades sem sinal?
3. Como é realizada a deteção de *overflow* em operações de adição com quantidades com sinal (codificadas em complemento para 2)?
4. Considere os seguintes pares de valores em **$s0** e **$s1**:

**i. $s0 = 0x70000000 $s1 = 0x0FFFFFFF ii. $s0 = 0x40000000 $s1 = 0x40000000**

* 1. Qual o resultado produzido pela instrução **add $t0, $s0, $s1**?
  2. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?
  3. Qual o resultado produzido pela instrução **sub $t0, $s0, $s1**?
  4. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?
  5. Qual o resultado produzido pelas instruções:

**add $t0, $s0,$s1 add $t0, $t0,$s1**

* 1. Para a alínea anterior os resultados são os esperados ou ocorreu *overflow*?

1. Para a multiplicação de dois operandos de "**m**" e "**n**" bits, respetivamente, qual o número de bits necessário para o armazenamento do resultado qualquer que este seja?
2. Apresente a decomposição em instruções nativas das seguintes instruções virtuais:

a. **mul $5,$6,$7**

b. **la $t0,label c/ label = 0x00400058**

c. **div $2,$1,$2**

d. **rem $5,$6,$7**

1. **ble $8,0x16,target**
2. **bgt $4,0x3F,target**
3. Determine o resultado da instrução **mul $5,$6,$7**, quando

**$6=0xFFFFFFFE e $7=0x00000005.**

1. Determine o resultado da execução das instruções virtuais **div $5,$6,$7 e rem $5,$6,$7**

quando **$6=0xFFFFFFF0 e $7=0x00000003**

1. Admita que pretendemos executar, em *Assembly* do MIPS, as operações:

**$t0 = $t2/$t3** e **$t1 = $t2 % $t3**.

Escreva a sequência de instruções em *Assembly* que permitem realizar estas duas operações. Use apenas instruções nativas

1. Descreva as regras que são usadas, na ALU do MIPS, para realizar uma divisão inteira entre duas quantidades com sinal.
2. Considerando que **$t0=-7** e **$t1=2**, determine o resultado da instrução **div $t0,$t1** e o valor armazenado respetivamente nos registos **HI** e **LO**.
3. Repita o exercício anterior admitindo agora que **$t0=0xFFFFFFF9** e **$t1=0x00000002.**
4. Considerando que **$5=-9** e **$10=2**, determine o valor que ficará armazenado no registo destino pela instrução virtual **rem $6,$5,$10**.
5. As duas sub-rotinas seguintes permitem detetar *overflow* nas operações de adição com e sem sinal, no MIPS. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas seguintes situações:

**a. $a0=0x7FFFFFF1, $a1=0x0000000E;**

**b. $a0=0x7FFFFFF1, $a1=0x0000000F;**

**c. $a0=0xFFFFFFF1, $a1=0xFFFFFFFF;**

**d. $a0=0x80000000, $a1=0x80000000;**

**# Overflow detection, signed**

**# int isovf\_signed(int a, int b); isovf\_signed: or $v0,$0,$0**

**xor $1,$a0,$a1 slt $1,$1,$0**

**bne $1,$0,notovf\_s addu $1,$a0,$a1 xor $1,$1,$a0**

**slt $1,$1,$0**

**beq $1,$0,notovf\_s ori $v0,$0,1**

**notovf\_s: jr $ra**

**# Overflow detection, unsigned**

**# int isovf\_unsigned(unsigned int a, unsigned int b); isovf\_unsig: ori $v0,$0,0**

**nor $1,$a1,$0 sltu $1,$1,$a0**

**beq $1,$0,notovf\_u ori $v0,$0,1**

**notovf\_u: jr $ra**

1. As duas sub-rotinas anteriores podem ser também escritas alternativamente com o código abaixo. A abordagem é ligeiramente diferente. No caso de operações sem sinal, o *overflow* pode ser detetado para as operações de soma e subtração. Analise o código apresentado e determine o resultado produzido, pelas duas sub-rotinas, nas condições indicadas nas alíneas da questão anterior:

**# Overflow detection in addition, unsigned**

**# int isovf\_unsigned\_plus(unsigned int a, unsigned int b); isovf\_unsig\_plus:**

**ori $v0, $0, 0**

**addu $t2, $a0, $a1 # temp = A + B; bge $t2, $a0, notovf\_uadd**

**bge $t2, $a1, notovf\_uadd ori $v0, $0, 1**

**notovf\_uadd: jr $ra**

**# Overflow detection in subtraction, unsigned**

**# int isovf\_unsigned\_sub(unsigned int a, unsigned int b); isovf\_unsig\_sub:**

**ori $v0,$0,0 slt $1, $a0, $a1**

**beq $1, $0, notovf\_usub ori $v0, $0,1**

**notovf\_usub: jr $ra**

**# Overflow detection, signed**

**# int isovf\_signed(int a, int b); isovf\_signed:**

**ori $v0,$0,0**

**add $1, $a0, $a1 # res = a + b; xor $a1, $a0, $a1 # tmp = a ^ b;**

**bltz $a1, notovf\_s # if (tmp < 0) no\_ovf(); xor $a1, $1, $a0 # tmp = res ^ a;**

**bgez $a1, notovf\_s # if (tmp >= 0) no\_ovf(); ori $v0,$0,1**

**notovf\_s: jr $ra**

1. Ainda no código das sub-rotinas das questões anteriores, qual a razão para não haver salvaguarda de qualquer registo na stack?
2. Na conversão de uma quantidade codificada em formato IEEE 754, precisão simples, para decimal, qual o número máximo de casas decimais com que o resultado deve ser apresentado?
3. Responda à questão anterior admitindo que o valor original se encontra agora representado com precisão dupla no formato IEEE 754.
4. Determine a representação em formato IEEE 754, precisão simples, da quantidade real 19,187510. Determine a representação da mesma quantidade em precisão dupla.
5. Determine, em decimal (vírgula fixa), o valor das quantidades seguintes representadas em formato IEEE 754, precisão simples. Na alínea b) apresente apenas o valor em notação científica usando base 2.

**a. 0xC19A8000.**

**b. 0x80580000.**

1. Considere que o conteúdo dos dois registos seguintes da FPU representam a codificação de duas quantidades reais no formato IEEE 754 precisão simples:

**$f0 = 0x416A0000**

**$f2 = 0xC0C00000**

Calcule o resultado das instruções seguintes, apresentando o seu resultado em hexadecimal:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **a. abs.s** | **$f4,$f2** | **#** | **$f4** | **=** | **abs($f2)** | | |
| **b. neg.s** | **$f6,$f0** | **#** | **$f6** | **=** | **neg($f0)** | | |
| **c. sub.s** | **$f8, $f0,$f2** | **#** | **$f8** | **=** | **$f0 – $f2** | | |
| **d. sub.s** | **$f10,$f2,$f0** | **#** | **$f10** | **=** | **$f2** | **-** | **$f0** |
| **e. add.s** | **$f12,$f0,$f2** | **#** | **$f12** | **=** | **$f0** | **+** | **$f2** |
| **f. mul.s** | **$f14,$f0,$f2** | **#** | **$f14** | **=** | **$f0** | **\*** | **$f2** |
| **g. div.s** | **$f16,$f0,$f2** | **#** | **$f16** | **=** | **$f0** | **/** | **$f2** |
| **h. div.s** | **$f18,$f2,$f0** | **#** | **$f18** | **=** | **$f2** | **/** | **$f0** |

**i. cvt.d.s $f20,$f2 # Convert single to double**

**j. cvt.w.s $f22,$f0 # Convert single to integer**

1. Considere a sequência de duas instruções Assembly:

**lui $t0,0xC0A8 mtc1 $t0,$f8**

qual o valor que ficará armazenado no registo **$f8,** expresso em base dez e vírgula fixa, admitindo uma interpretação em IEEE 754 precisão simples?

1. Considerando que **$f2=0x3A600000** e **$f4=0xBA600000**, determine o resultado armazenado em

**$f0** pela instrução **sub.s $f0,$f2,$f4**.

1. Repita o exercício anterior admitindo agora as seguintes condições:

**$f4=0x3F100000** e **$f6=0x408C0000** e a instrução **add.s $f8,$f4,$f6.**

**$f2=0x3F900000** e **$f4=0xBEA00000** e a instrução **mul.s $f0,$f2,$f4**

**$f2=0x258c0000** e **$f4=0x41600000** e a instrução **div.s $f0,$f2,$f4**

1. Numa norma hipotética KPT de codificação em vírgula flutuante, a mantissa normalizada após a realização de uma operação aritmética tem o valor **1.1111 1111 1111 1110 1000 0000**. Qual será o valor final da mantissa (com 16 bits na parte fracionária) após arredondamento para o ímpar mais próximo?
2. Assuma que **x** é uma variável do tipo **float** residente em **$f8** e que o *label* **endWhile** corresponde ao endereço da primeira instrução imediatamente após um ciclo *while()*. Se a avaliação da condição para executar o *loop* for *while (x > 1.5){..}* escreva, em Assembly do MIPS, a sequência de instruções necessárias para determinar esta condição.
3. Determine, de acordo com o formato IEEE 754 precisão simples, a representação normalizada, e arredondada para o par mais próximo, do número **100,110110000000000000101102**.
4. **Numa implementação *single cycle* da arquitetura MIPS, a frequência máxima de operação é de 2GHz (para os atrasos de propagação a seguir indicados). Determine o atraso máximo que pode ocorrer nas operações da ALU. Considere que, para o *File Register* e para as memórias, os tempos de escrita indicados são os tempos de preparação da operação antes de uma transição ativa do sinal de relógio.**

**Memórias externas: leitura – 175ps, escrita – 120ps; *File register*: leitura – 45ps, escrita – 15ps; Unidade de Controlo: 10ps; Somadores: 40ps; Outros: 0ns; Setup time do *Program Counter*: 5ps**

Instrução com maior atraso: LW

T lw « T rm + max(T rrf , T cntl, T se) + T alu + T rm + T wrf

1. ***Determine, numa implementação single-cycle da arquitetura MIPS, a frequência máxima de operação imposta pela instrução “sw”, assumindo os atrasos a seguir indicados:***

***Memórias externas: leitura – 12ns, escrita – 4ns; File register: leitura – 4ns, escrita – 1ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 5ns; Somadores: 2ns; Outros: 0ns.***

***Setup time do Program Counter: 1ns***

T sw = T rm + max( T rrf , T cntrl , T se) + T alu + T wm

= 12 + max(4,1,0) + 5 + 4

= 12 + 4 + 5 + 4

= 25 ns

1. ***Determine, numa implementação single-cycle da arquitetura MIPS, a frequência máxima de operação imposta pela instrução "beq", assumindo os atrasos a seguir indicados:***

***Memórias externas: leitura – 11ns, escrita – 3ns; File register: leitura – 3ns, escrita – 1ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 5ns; Somadores: 2ns; Outros: 0ns.***

***Setup time do Program Counter: 1ns***

T beq = T rm + max( max(T rrf , T cntrl) + T alu , T se + T add + T sl2) + T stPC

= 11 + max(max(3,1) + 5 , 2 + 0 + 0 ) + 1

= 11 + max ( 3 + 5 = 7 , 2 ) + 1

= 11 + 7 + 1

= 19 ns

1. ***Determine, numa implementação single cycle da arquitetura MIPS, o período mínimo do sinal de relógio imposto pelas instruções tipo R, assumindo os atrasos a seguir indicados:***

***Memórias externas: leitura – 12ns, escrita – 4ns; File register: leitura – 3ns, escrita – 1ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 6ns; Somadores: 2ns; Outros: 0ns.***

***Setup time do Program Counter: 1ns***

T type-r = T rm + max(T rrf, T cntrl ) + T alu + T wrf

= 12 + max(3,1) + 6 + 1

= 12 + 3 + 6 + 1

= 22 ns

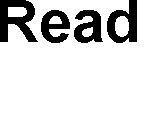
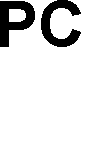
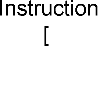
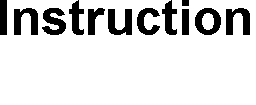
1. Identifique os principais aspetos que caracterizem uma arquitetura *single cycle*, quer do ponto de vista do modelo da arquitetura, como das características da sua unidade de controlo.
2. ***Numa implementação single cycle da arquitetura MIPS, no decurso da execução de uma qualquer instrução, a que corresponde o valor presente na saída do registo PC?***

Resposta: PC+4

1. ***Preencha a tabela seguinte, para as instruções indicadas, com os valores presentes à saída da unidade de controlo principal da arquitetura single cycle dada nas aulas.***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrução** | **Opcode** | **ALUOp[1..0]** | **Branch** | **RegDst** | **ALUSrc** | **Memto**  **Reg** | **Reg**  **Write** | **Mem**  **Read** | **Mem**  **WRite** |
| lw | 100011 | 00 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| sw | 101011 | 00 | 0 | X | 1 | x | 0 | 0 | 1 |
| addi | 001000 | 00 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| slti | 001010 | 11 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| beq | '000100 | 01 | 1 | X | 0 | x | 0 | 0 | 0 |
| R - Format | 000000 | 10 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

1. Admita que na versão *single cycle* do CPU MIPS dado nas aulas, pretendíamos acrescentar o suporte das instruções **jal address** e **jr $reg**. Esquematize as alterações que teria de introduzir no *datapath* para permitir a execução destas instruções (use como base o esquema da próxi



## Fig. 1 - Datapath single-cycle

1. Admita que na versão *single cycle* do CPU MIPS, pretendíamos executar a instrução **slt $3,$5,$9**. Descreva por palavras suas como é esta instrução realizada ao nível da ALU, e qual o conteúdo final no registo **$3**, admitindo que **$5=0xFF120008** e **$9=0x00C00FFF**.
2. Suponha que os tempos de atraso introduzidos pelos vários elementos funcionais de um *datapath single-cycle* são os seguintes:

|  |  |  |  |
| --- | --- | --- | --- |
| Acesso à memória para leitura (tRM): | 12ns | Acesso à memória para preparar escrita (tWM): | 4ns |
| Acesso ao register file para leitura (tRRF): | 5ns | Acesso ao register file para preparar escrita (tWRF): 2ns | |
| Operação da ALU (tALU): | 7ns | Operação de um somador (tADD): | 2ns |
| Multiplexers e restantes elementos funcionais: | 0ns | Unidade de controlo (tCNTL): | 2ns |
| Tempo de setup do PC (tstPC): | 1ns |  | |

* 1. Determine o tempo mínimo para execução das instruções tipo **R**, **LW**, **SW**, **BEQ** e **J**.
  2. Calcule a máxima frequência do relógio que garanta uma correta execução de todas as instruções.

1. Suponha agora que dispunha de uma tecnologia que que o período de relógio podia ser adaptado instrução a instrução, em função da instrução em curso. Determine qual o ganho de eficiência que poderia obter com esta tecnologia face a uma tecnologia em que a frequência do relógio é a que obteve na questão anterior (admita os mesmos atrasos de propagação). Para isso, assuma que o programa de *benchmarking* tem a seguinte distribuição de ocorrência de instruções:

15% de **lw**, 15% de **sw**, 40% de tipo **R**, 20% de **branches** e 10% de **jumps**

1. Ainda para os tempos utilizados nas duas questões anteriores, determine qual a máxima frequência de trabalho no caso de o *datapath* ser do tipo *multi-cycle*.



**PCWrite PCWriteCond**

**IorD**

**PCSource**

**ALUOp**

**MemRead MemWrite IRWrite**

**MemtoReg**

**Control (FSM)**

OP [5-0]

**ALUSelA ALUSelB**

**RegWrite**

**RegDst**

32

2 M

32

1 U

X

0 6

Inst[25-0]

**Shift Left 2**

32

PC[31-28]

**Read Write**

Inst [25-21]

**PC**

0M

**Write**

**Inst [31-26]**

**Write**

0M

**R Reg.#1** 32

U

U X

1X

**Address**

Inst [20-16]

**R Data #1**

A

14

**R Reg.#2**

1

**Read Data**

Zero

**ALU**

**Result**

**Write Data**

**Inst [25-0]**

**Instruction Register**

Inst [15-11]

0M

32

12

U X

**W Reg.** **R Data #2**

B

4

**W Data**

**Registos**

**Memory**

0 M

1. U
2. X

3 5

0

M

**Alu Out**

U

**Data Register**

32

1 X

32

3

32

16

32

Inst [15-0]

**Sign Extend**

**Shift Left 2**

ALU

Control

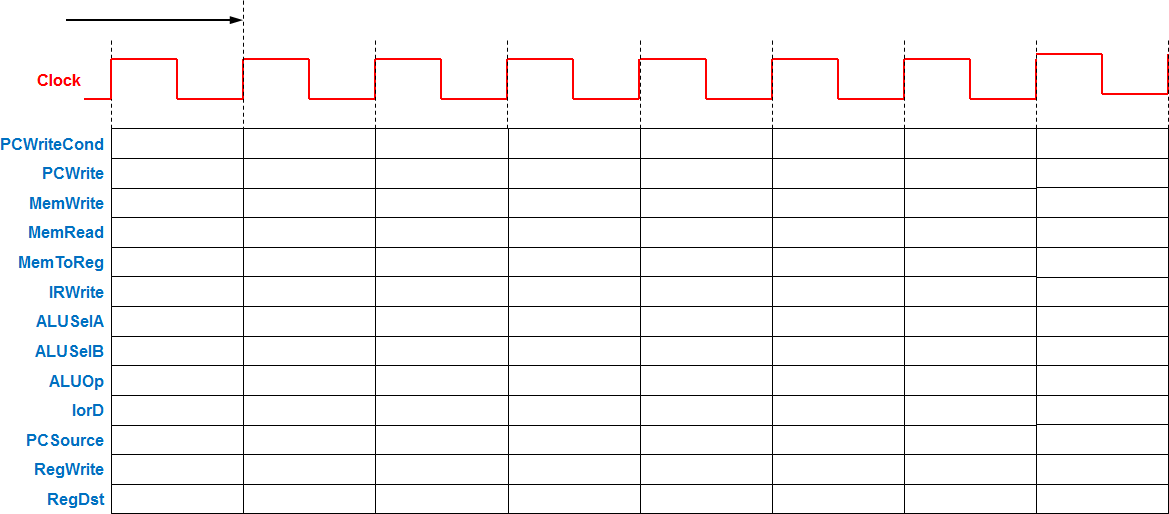
Inst [5-0]

**Clock**

## Fig. 2 - Datapath multi-cycle

1. Considere o *datapath multi-cycle* presente na figura 2 e a respetiva unidade de controlo. Preencha a tabela abaixo considerando que a coluna da esquerda corresponde ao último ciclo de execução da primeira instrução do código seguinte, e que a sequencia de instruções em causa é:

**add $t0, $t2, $t1 sw $t0, 0($t3) beq $t0, $t1, next**



PCWriteCond – 0 // 0 / 0 / 0 / 0 // 0 / 0 / 0

PCWrite – 0 // 1 / 0 / 0 / 0 // 1 / 0 / 0

MemWrite- 0 // 0 / 0 / 0 / 1 // 0 / 0 / 0

MemRead- 0 // 0 / 0 / 0 / 0 // 0 / 0 / 0

MemToReg- 0 // 0 / 0 / 0 / 0 // 0 / 0 / 0

IRWrite- 0 // 1 / 0 / 0 / 0 // 1 / 0 / 0

ALUSelA- x // 0 / 0 / 1 / x // 0 / 0 / 1

ALUSelB- x //01/11/10/x //01/11/00

AluOP- x //00/00/00/x //00/00/01

IorD- x // 0 / x / x / 1 // 0 / x / x

PCSource- x //00/ x / x / x //00/ x / 01

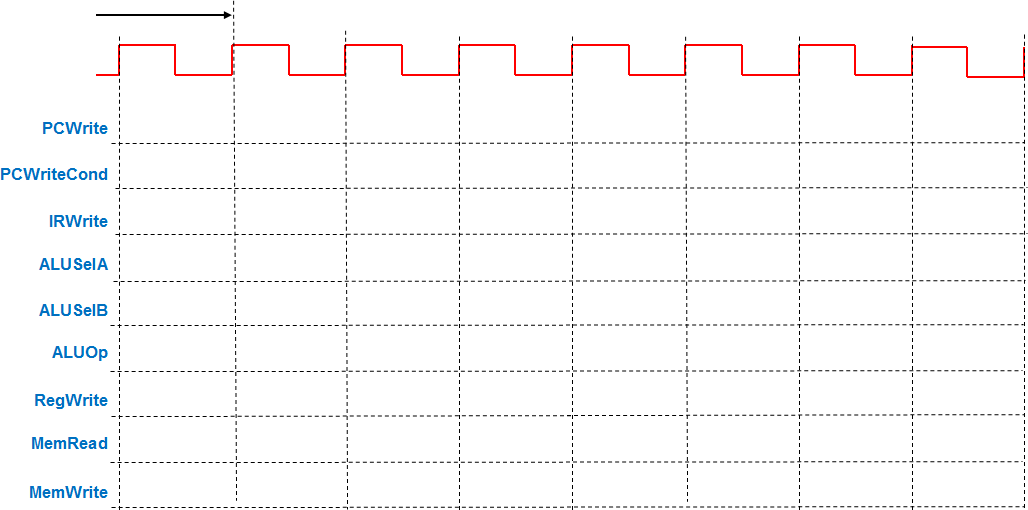
RegWrite- 1 // 0 / 0 / 0 / 1// 0 / 0 / 0

RegDst- 1 // x / x / x / x // x / x / x

1. Repita o exercício anterior para as seguintes sequências de instrução:

|  |  |  |
| --- | --- | --- |
| **a. or $t0, $0, $t1 addi $t0, $t1, 0x20**  **j label** | **b. lw $s0, 0($t1) lw $s1, 4($t1)**  **add $t2, $s1, $s2** | **c. sw $t0, 0($t1) sub $t0, $t3, $t2**  **slt $t1, $t0, $t2** |

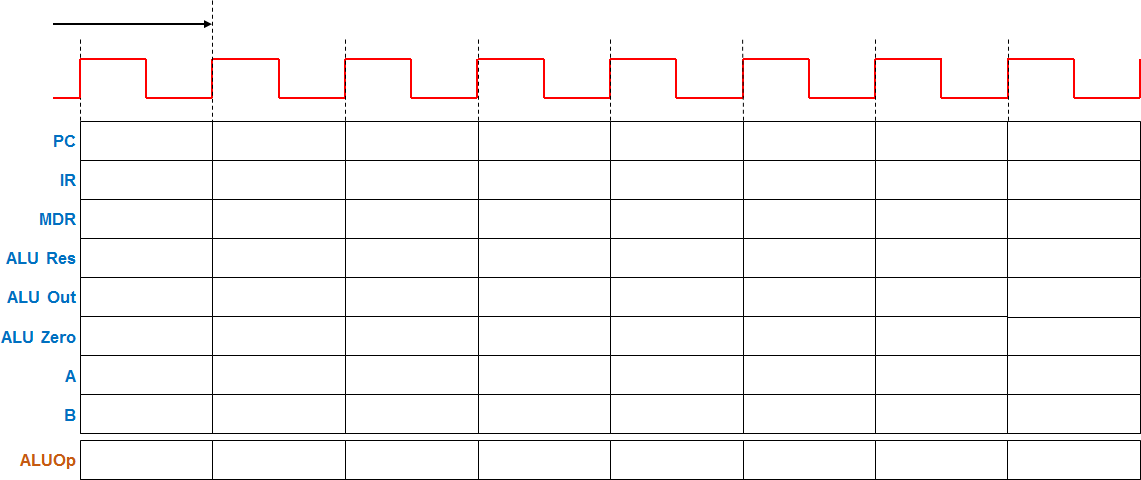
1. Para as mesmas sequências de instruções apresentadas nos dois exercícios anteriores, preencha, na forma de um diagrama temporal, a tabela seguinte.



1. Ainda para as mesmas sequências de instruções apresentadas nos três exercícios anteriores, preencha a tabela abaixo com os valores presentes à saída da ALU e dos elementos de estado indicados. Consulte a tabela da última página se necessário. Admita que, no início de cada sequência, o conteúdo dos registos relevantes é o seguinte:

**[PC=0x00400000], [$t0=0x000013FC], [$t1=0x10010000], [$t2=0x90FFFF64], [$t3=0x00000028]**

**e que na memória [\*0x10010000 = 0x00000020] e [\*0x10010004 = 0x00000038]**



1. Calcule o número de ciclos de relógio que o programa seguinte demora a executar, desde o *Instruction fetch* da 1ª instrução até à conclusão da última instrução:



* 1. num *datapath single-cycle*
  2. num *datapath multi-cycle*

**Memória de dados Address Value 0x0000000 0x10**

**0x0000004 0x20**

**main: # p0 = 0;**

**lw $1,0($0) # p1 = \*p0 = 0x10; add $4,$0,$0 # v = 0;**

**lw $2,4($0) # p2=\*(p0+1)=0x20;**

**loop: # do {**

**lw $3,0($1) # aux1 = \*p1; add $4,$4,$3 # v = v + \*p1; sw $4,36($1) # \*(p1 + 9) = v; addiu $1,$1,4 # p1++; sltu $5,$1,$2 #**

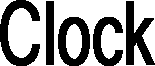
**bne $5,$0,loop # } while(p1 < p2); sw $4,8($0) # \*(p0 + 2) = v; lw $1,12($0) # aux2 = \*(p0 + 3);**

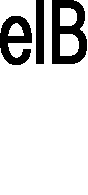
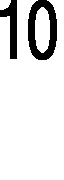
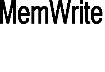
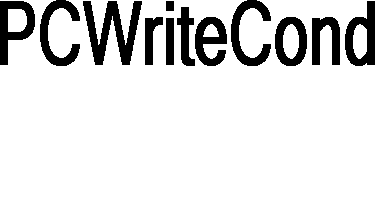
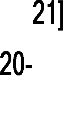
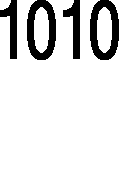
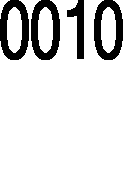
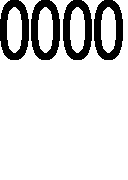
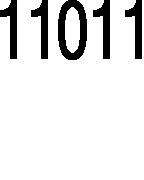
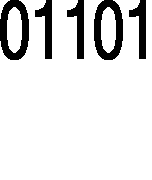
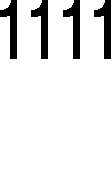
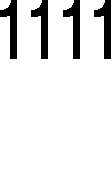
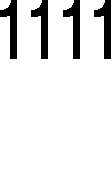
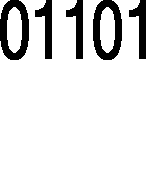
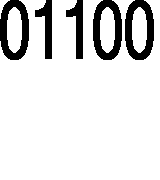
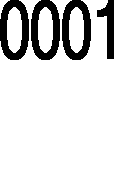
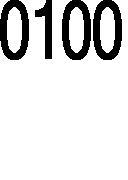
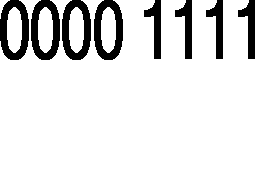
1. Repita o exercício anterior assumindo que o valor armazenado no endereço de memória **0x00000004**

é **0x2C**.

1. Descreva, sucintamente, as principais diferenças, ao nível estrutural, entre os *datapath single-cycle* e

*multi-cycle*.

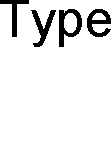
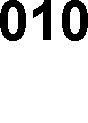
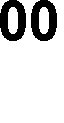
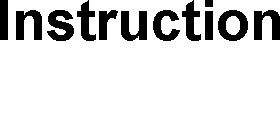
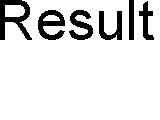
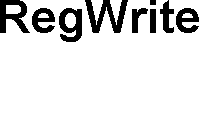
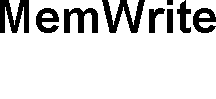
1. Indique, para o caso de um *datapath multi-cycle,* quais as operações realizadas pela ALU no decurso dos dois primeiros ciclos de relógio de qualquer instrução.
2. Considere o diagrama temporal seguinte relativo à execução de uma sequência de três instruções no *datapath* da Fig. 2, das quais apenas a segunda está completamente representada. Obtenha o código *assembly* desta sequência de três instruções.



1. Considere a seguinte sequência de três instruções a serem executadas num *datapath muti-cycle*:

|  |  |  |
| --- | --- | --- |
| **lw** | **$6,** | **0($7)** |
| **and** | **$8,** | **$6, $5** |
| **beq** | **$8,** | **$0, L1** |

No diagrama temporal seguinte, relativo à execução desta sequência, identifique o nome dos sinais de controlo representados. (Note: o IorD não faz parte destes sinais)



1. Considere o *datapath multi-cycle* e a unidade de controlo fornecidos na figura acima. Admita que os valores indicados no *datapath* fornecido correspondem à “fotografia” tirada no decurso da execução de uma instrução armazenada no endereço **0x8040000C**. Tendo em conta todos os sinais, identifique, em *assembly*, a instrução que está em execução e a respetiva fase.
2. Considere a instrução **beq $5,$6,L2** armazenada no endereço **0x0040002C**. Admita que

**$5=0x1001009C** e **$6=0x100100B0**. Identifique os registos representados na figura seguinte e obtenha o código máquina, em hexadecimal, da instrução indicada.

clock

Início da execução do branch

?

0x0040002C

0x100100B0

0x00400030

? 0x00400030 0x00400044

1. Considere o *datapath* e a unidade de controlo fornecidos na figura acima (com ligeiras alterações relativamente à versão das aulas teórico-práticas). Analise cuidadosamente as alterações introduzidas e identifique quais são as novas instruções que este *datapath* permite executar quando comparado com a versão fornecida nas aulas TP.



**0**

**0**

**0**

**0**

**0**

**0**

**10**

**PCWrite** **PCWriteCond**

**IorD MemRead MemWrite**

**IRWrite**

**MemtoReg**

**CPInv** **1**

**Control**

**PCSource 11**

**ALUOp** **00**

**ALUSelA 1**

**ALUSelB 00**

**RegWrite 1**

OP [5-0]

**DatSrc**

**1**

**RegDst 00**

3 M

2 U

1 X

06

Inst[25-0]

**Shift Left 2**

**0x00400048**

PC[31-28]

0 M

0 U

11

M

Inst[25-21]

**Inst [31-26]**

**R Reg.#1**

U

1 X

X

**Address**

Inst[20-16]

**R Data #1**

4

**R Reg.#2**

Inst

[15-11] 0M

**MemData**

**Inst [25-0]**

1 U

**W Reg.**

**R Data #2**

0

1

Zero **0 ALU**

Result

M

**Alu Out**

**Write Data**

**Memory**

**Instruction Register**

0x1F

X

22

4

**W Data**

U

**Registos**

2 X

3 5

0 M

**Data Register**

X

23

Inst [15-0]

1 U

**ALU**

**Control**

16

**Sign** 32

**Extend**

**Shift Left 2**

Data [7-0]

Data [15-8]

Data [23-16]

Data [31-24]

0

1M

0 M

U 8

U

32

2 X

37

1 X

8

Add [1,0]

B

A

**P C**

**Sign Extend**

1. Descreva, justificando, as principais características da unidade de controlo numa implementação *pipelined* da arquitetura MIPS, incluindo a sua natureza (combinatória ou síncrona) os sinais que constituem as variáveis independentes de entrada e as suas saídas.
2. Indique o que determina a máxima frequência de relógio de uma implementação *pipelined* da arquitetura MIPS com base nos principais elementos operativos que a constituem.
3. Calcule, numa implementação *pipelined* da arquitetura MIPS, em que a operação de *Write Back* é executada a meio do ciclo de relógio, a frequência máxima de operação assumindo que os elementos operativos apresentam os seguintes atrasos de propagação:

a.

Memórias externas: Leitura: 10 ns, Escrita: – 8ns; File register: Leitura – 2ns, Escrita – 2ns; Unidade de Controlo: 2ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns.

b.

Memórias externas: Leitura: 5 ns, Escrita: – 7ns; File register: Leitura – 1ns, Escrita – 1ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 8ns; Somadores: 1ns; Outros: 0ns.

c.

Memórias externas: Leitura: 8 ns, Escrita: – 10ns; File register: Leitura – 2ns, Escrita – 4ns; Unidade de Controlo: 2ns; ALU (qualquer operação): 6ns; Somadores: 2ns; Outros: 0ns.

1. Identifique os principais tipos de *hazard* que podem existir numa implementação *pipelined* de um processador.
2. Numa arquitetura *pipelined*, como se designa a técnica que permite utilizar como operando de uma instrução um resultado produzido por outra instrução que se encontra numa etapa mais avançada do mesmo.
3. Explique por palavras suas em que circunstâncias pode ocorrer um *hazard* de dados numa implementação *pipelined* de um processador
4. A existência de *hazards* de controlo pode ser resolvida por diferentes técnicas dependendo da arquitetura em causa. Identifique a técnica usada para o efeito numa arquitetura MIPS com *datapath pipelined*, referindo como se designa essa técnica e em que consiste.
5. Em certas circunstâncias relacionadas com *hazards* de dados, não é possível resolver o problema sem recorrer a uma paragem parcial do *pipeline,* através do atraso de um ou mais ciclos de relógio no início da execução de uma instrução. Indique como se designa essa técnica e em que consiste ao nível do controlo do *pipeline*
6. Determine o número de ciclos de relógio que o trecho de código seguinte demora a executar num pipeline de 5 fases, desde o instante em que é feito o *Instruction Fetch* da 1ª instrução, até à conclusão da última instrução.

|  |  |  |
| --- | --- | --- |
| **add** | **$1,$2,$3** |  |
| **lw** | **$2,0($4)** |
| **sub** | **$3,$4,$3** |
| **addi** | **$4,$4,4** |
| **and** | **$5,$1,$5** | **#"and" em ID, "add" já terminou** |
| **sw** | **$2,0($1)** | **#"sw" em ID, "add" e "lw" já terminaram** |

1. Num *datapath single-cycle* o código da pergunta anterior demoraria 6 ciclos de relógio a executar. Por que razão é a execução no *datapath pipelined* mais rápida?
2. Quantos ciclos de relógio demora a execução do mesmo código num *datapath multi-cycle*?
3. Admita uma implementação *pipelined* da arquitetura MIPS com unidade de *forwarding* para EX e ID. Identifique, para as seguintes sequências de instruções, de onde e para onde deve ser executado o *forwarding* para que não seja necessário realizar qualquer *stall* ao pipeline:

a.

**add $t0,$t1,$t2 lw $t1,0($t3) beq $t3,$t0,LABEL**

b.

**sub $t0,$t1,$t2 addi $t3,$t0,0x20**

c.

**lw $t0,0($t2) sll $t2,$t2,2 sw $t3,0($t0)**

d.

**lw $t3,0($t6) xori $t0,$t4,0x20 sw $t3,0($t0)**

1. Descreva, por palavras suas, a função da unidade de *forwarding* de uma implementação *pipelined* da arquitetura MIPS.
2. Admita o seguinte trecho de código, a executar sobre uma implementação *pipelined* da arquitetura MIPS com *delayed branches*, e unidade de *forwarding* de EX/MEM e MEM/WB para o estágio EX.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LABEL: lw** | **$t3,** | **0($t4)** | **#** | **1** |
| **sub** | **$t7,** | **$t5, $t6** | **#** | **2** |
| **ori** | **$t2,** | **$0, 0** | **#** | **3** |
| **beq** | **$t2,** | **$0, LABEL** | **#** | **4** |
| **add** | **$t4,** | **$t7, $t7** | **#** | **5** |

* 1. Identifique os vários *hazards* neste código e determine se os *hazards* de dados podem ser resolvidos por *forwarding*.
  2. Identifique as situações em que é necessário executar *stalling* do pipeline e o respetivo número de

*stalls*

* 1. Resolva o problema supondo que a arquitetura suporta também *forwarding* de EX/MEM para ID.

1. Para o trecho de código seguinte identifique todas as situações de *hazard* de dados e de controlo que ocorrem na execução num pipeline de 5 fases, com *branches* resolvidos em ID.

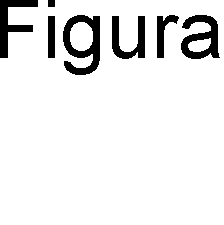
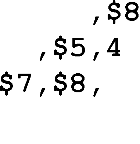
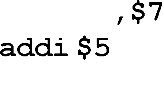
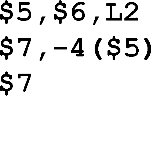
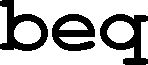
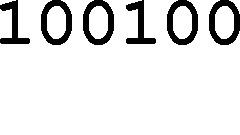
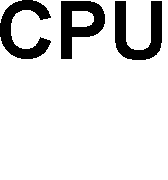
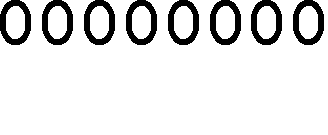
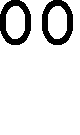
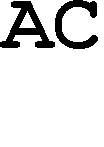
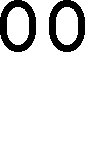
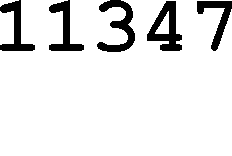
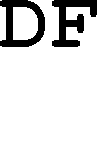
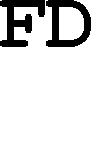
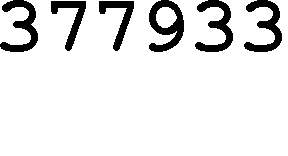
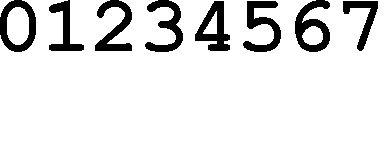
|  |  |  |
| --- | --- | --- |
| **main:** | **lw** | **$1,0($0)** |
|  | **add** | **$4,$0,$0** |
|  | **lw** | **$2,4($0)** |
| **loop:** | **lw** | **$3,0($1)** |
|  | **add** | **$4,$4,$3** |
|  | **sw** | **$4,36($1)** |
|  | **addiu** | **$1,$1,4** |
|  | **sltu** | **$5,$1,$2** |
|  | **bne** | **$5,$0,loop** |
|  | **sw** | **$4,8($0)** |
|  | **lw** | **$1,12($0)** |

**Memória de dados Addr Value**

**0x0000000 0x10**

**0x0000004 0x20**

1. Apresente o modo de resolução das situações de *hazard* de dados do código da questão anterior, admitindo que o pipeline não implementa *forwarding*.
2. Calcule o número de ciclos de relógio que o programa anterior demora a executar num pipeline de 5 fases, sem *forwarding*, com *branches* resolvidos em ID e *delayed branch*, desde o *Instruction Fetch* da 1ª instrução até à conclusão da última instrução.
3. Resolva o problema anterior, considerando agora que o pipeline implementa *forwarding* de MEM e WB para EX.
4. Calcule finalmente o número de ciclos de relógio que o programa do problema 157 demora a executar num pipeline de 5 fases, com *forwarding* para EX e para ID, com *branches* resolvidos em ID e *delayed branch*, desde o IF da 1ª instrução até à conclusão da última instrução.
5. Considere o trecho de código apresentado na figura seguinte, bem como as tabelas e os valores dos registos que aí se apresentam. Admita que o valor presente no registo **$PC** corresponde ao endereço da primeira instrução, que nesse instante o conteúdo dos registos é o indicado, e que vai iniciar-se o *instruction fetch* dessa instrução. Considere, para já, o *datapath* e a unidade de controlo fornecidos na pergunta 130 (Fig. 2), correspondentes a uma implementação *multi-cycle* simplificada da arquitetura MIPS.



1. Determine o valor presente à saída do registo **ALUOut** durante a terceira fase de execução da segunda instrução (**lw $7,-4($5)**).
2. Face aos valores presentes no segmento de dados (tabela da esquerda) e nos registos, calcule o número total de ciclos de relógio que demora a execução completa do trecho de código apresentado, numa implementação *multi-cycle* do MIPS (desde o instante inicial do *instruction fetch* da primeira instrução até ao momento em que vai iniciar-se o *instruction fetch* da instrução presente em “**L2:**”).
3. Suponha agora que o mesmo código é executado numa versão *pipelined* do *datapath* do MIPS semelhante à abordada nas aulas teórico-práticas de AC1. Admita que este *datapath* suporta apenas *forwarding* para EX. Determine o número total de ciclos de relógio que demora a execução completa do trecho de código apresentado, até ao instante inicial do *instruction fetch* da instrução imediatamente a seguir ao **nop**.
4. Continue a considerar a execução do código numa versão *pipelined* do *datapath* do MIPS com *forwarding* para EX e para ID. Admita que, no instante zero, correspondente a uma transição ativa do sinal de relógio, vai iniciar-se o *instruction fetch* da primeira instrução. Determine o valor à saída da ALU na conclusão do sexto ciclo de relógio contado a partir do instante zero.
5. Repita as questões 163 a 166 para os dados da figura seguinte:

**Endereço**

**...**

|  |  |
| --- | --- |
| **0x10010028** | **0x31434120** |
| **0x1001002C** | **0x31303220** |
| **0x10010030** | **0x00000032** |
| **0x10010034** | **0xE0DE0AC1** |
| **0x10010038** | **0x0000FFFF** |
| **0x1001003C** | **0x00000000** |

**Dados**

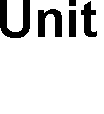
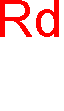
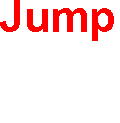
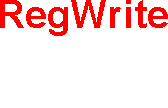
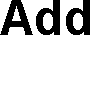
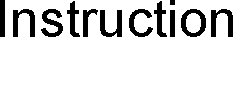
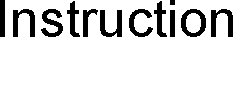
**...**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **...** **Figura** | | | | | |
|  | | | | |  |
| **$6** |  |  |  | **L1:** | **lb** | **$9,0($8)** |
| **$7** |  |  |  |  | **lw** | **$2,4($7)** |
| **$8** |  |  |  |  | **addi** | **$2,$2,1** |
| **$9** |  |  |  |  | **xor** | **$5,$6,$9** |
|  | **...** |  |  |  | **addi $8,$8,1**  **sw** **$2,4($7)** | |
| **$PC** | **0x004000C4** | **bne** | | | | **$5,$0,L1** |
| **CPU** | | **xor**  **L2: nop** | | | | **$0,$0,$0** |

|  |
| --- |
| **0x00000000** |
| **0x10010038** |
| **0x10010028** |
| **0xE0DE0AC1** |

**...** **...**

1. Considere a versão com *pipeline* do *datapath* apresentado na Fig. 3. Identifique todas as combinações de *forwarding* disponíveis neste *datapath* e, para cada uma delas, escreva uma curta sequência de instruções que desencadeie esse tipo específico de *forwarding*. Nos casos em que tal se aplique, identifique igualmente os casos em que é preciso gerar *stalling* e o número de ciclos de *stalling* necessários.



***Fig. 3 – Piplined Datapath***

**Tabela de códigos de função (funct) e códigos de operação (OpCode) das principais instruções do MIPS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Arithm / Logical Instructions** | |  | **Comparison Instructions** | |
| **Instruction** | **(funct)** |  | **Instruction** | **(OpCode)** |
| **add** | **100000 (0x20)** |  | **slti** | **001010 (0x0A)** |
| **addu** | **100001 (0x21)** |  | **sltiu** | **001001 (0x09)** |
| **and** | **100100 (0x24)** |  |  |  |
| **div** | **011010 (0x1A)** |  |  |  |
| **divu** | **011011 (0x1B)** |  |  |  |
| **mult** | **011000 (0x18)** |  | **Branch Instructions** | |
| **multu** | **011001 (0x19)** |  | **beq** | **000100 (0x04)** |
| **nor** | **100111 (0x27)** |  | **bne** | **000101 (0x05)** |
| **or** | **100101 (0x25)** |  | **bgtz** | **000111 (0x07)** |
| **sll** | **000000 (0x00)** |  | **bgez** | **000001 (0x01)1** |
| **sra** | **000011 (0x03)** |  | **bltz** | **000001 (0x01)** |
| **srl** | **000010 (0x02)** |  | **blez** | **000110 (0x06)** |
| **sub** | **100010 (0x22)** |  |  |  |
| **subu** | **100011 (0x23)** |  | **Jump Instructions** | |
| **xor** | **100110 (0x26)** |  | **j** | **000010 (0x02)** |
| **slt** | **101010 (0x2A)** |  | **jal** | **000011 (0x03)** |
| **sltu** | **101001 (0x29)** |  | **jalr** | **001001 (0x09)** |
|  |  |  | **jr** | **001000 (0x08)** |
| **Arithm / Logical Imm** | |  |  |  |
| **Instruction** | **(OpCode)** |  | **Load/Store Instructions** | |
| **addi** | **001000 (0x08)** |  | **lb** | **100000 (0x20)** |
| **addiu** | **001001 (0x09)** |  | **lbu** | **100100 (0x24)** |
| **andi** | **001100 (0x0C)** |  | **lw** | **100011 (0x23)** |
| **ori** | **001101 (0x0D)** |  | **sb** | **101000 (0x28)** |
| **xori** | **001110 (0x0E)** |  | **sw** | **101011 (0x2B)** |
|  |  |  |  |  |
|  |  |  | **Data Movement Instructions** | |
|  |  |  | **mfhi** | **010000 (0x10)** |
|  |  |  | **mflo** | **010010 (0x12)** |
|  |  |  | **mthi** | **010001 (0x11)** |
|  |  |  | **mtlo** | **010011 (0x13)** |

1 O OpCode é igual ao da instrução **bltz** mas o valor de **rt** é igual a 00001b