AC1 RESPOSTAS

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

  CPU (processa a execução de uma sequência de informações), **Memória** (responsável pelo armazenamento), **Unidades I/O** (responsáveis com a comunicação com o exterior)

- **Quais são os 3 principais blocos funcionais que integram um CPU?**

  ALU, Registos, Unidade de Controlo

- **Qual a função do registo Program Counter?**

  Registo que guarda o endereço de memória da próxima instrução a ser executada

- **Quais os passos mais importantes em que se decompõe a execução de uma instrução no CPU?**

  Instruction fetch (leitura do código da máquina da instrução), **Instruction decode** (decodificação da instrução pela Unidade de Controlo), **Operand fetch** (leitura dos operandos), **Execute** (execução da operação especifica da instrução), **Store Result** (guardar o resultado no registo de destino específico da operação)

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

  Usado para programas que traduzem código fonte high-level, para código machine-level de forma a criar um programa executável. Considera um programa inteiro como código e traduz. Verifica erros

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

  Aceita como input o código assembly e executa a sua tradução para machine code. Tb verifica cada instrução e verifica se está corretamente bem escrita. (check dos operadores)

- **Quantos registos internos de uso geral tem o MIPS?**

  32 registos

- **Qual a dimensão, em bits, que cada um dos registos internos do MIPS pode armazenar?**

  32 bits

- **Qual a sintaxe, em Assembly, de uma instrução aritmética no MIPS?**

  'instrução' reg1, reg2, reg3 => add $t1,$t2,$t3

- **O que distingue a instrução SRL da instrução SRA do MIPS?**

  Shift Right Logical (SRL) é destinado a operações do tipo **unsigned**, **entram 0's**

  Shift Right Arithmetic (SRA) é destinado a operações do tipo **signed,** entram **0's caso positivo**, entram **1's caso negativo**

- **Se $5=0x81354AB3, qual o resultado, expresso em hexadecimal, das instruções:**
  - **srl $3,$5,1**

    $5 = 1000 0001 0011 0101 0100 1010 1011 0011

    logical => unsigned => entram 0's

    0100 0000 1001 1010 1010 0101 0101 1001

    $3 = 0x409AA559
  - **sra $4,$5,1**

    $5 = 1000 0001 0011 0101 0100 1010 1011 0011

    arithmetic => signed => como é negativo => entram 1's

    1100 0000 1001 1010 1010 0101 0101 1001

    $4 = 0xC09AA559

- **System calls:**
  - **O que é uma system call?**

    A chamada de uma função

  - **No MIPS, qual o registo usado para identificar a system call a executar? $v0**

  - **Qual o registo ou registos usados para passar argumentos para as system calls?**

    $a0 e $a1

  - **Qual o registo usado para obter o resultado devolvido por uma system call nos casos em que isso se aplica? $v0**

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

  É o número hexadecimal que identifica um registo

- **O que é o espaço de endereçamento de um processador?**

  Gama total possível de endereços que o CPU consegue referenciar (depende do address bus). Na arquitetura MIPS temos de **0 a 2^32 - 1**

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

  Secção de dados (datapath): elementos que encaminham, processam e armazenam informação. **Blocos lógicos (multiplexers), ALU, Registos internos.**
  Unidade de controlo (control path/control unit): responsável pela coordenação dos elementos do datapath durante a execução do programa
  Control Path tanto pode ser **sequencial (máquina de estados)** como **combinatório**, dependendo da arquitetura
  Mesmo alterando o control path, o **CPU** é sempre uma **máquina de estados síncrona**

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

  Na memória reside, ao mesmo tempo, informação de natureza variada
  Para isso as instruções são representadas da mesma forma que os números

- **Como se codifica uma instrução? Que informação fundamental deverá ter o código de uma instrução?**

- **Descreva pelas suas próprias palavras o conceito de ISA.**

  Arquitetura do conjunto de instruções com os requisitos básicos: Implementação simples e eficiente em hardware, fácil de entender e programar, desenvolvimento de compiladores eficientes

- **Quantas e quais são as classes de instruções que agrupam as diferentes instruções de uma dada arquitetura?**

  3 classes: **Processamento, transferência de informação, controlo de fluxo de execução**

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

  MIPS é uma arquitetura load-store.
  Arquiteturas Register-Memory: Operandos residem em registos internos do CPU ou em memória
  Arquiteturas Load-Store: Operandos das instruções residem em registos internos do CPU (nunca em memória)

Exemplo: Operandos são copiados da memória para os registos internos, para ser possível efetuar a operação (load) e depois o resultado é enviado dos registos internos para a memória externa (store)

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

  Instruction fetch (leitura do código da máquina da instrução)**, Instruction decode** (decodificação da instrução pela Unidade de Controlo)**, Operand fetch** (leitura dos operandos)**, Execute** (execução da operação especifica da instrução)**, Store Result** (guardar o resultado no registo de destino específico da operação)

- **Como se designa o barramento que permite identificar, na memória, a origem/destino da informação transferida?**

  Address Bus

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

  Ligação de **transferência de informação** entre CPU<->Memória, CPU<->I/O, ou seja liga o CPU aos restantes

- **Os processadores da arquitetura hipotética ZWYZ possuem 4 registos internos e todas as instruções são codificadas em 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?**

  24 bits: 5 bits reg1 reg2 reg3 imm
  Como existem apenas 4 registos internos é possível representá-los com 2 bits, 00, 01, 10, 11. Logo cada campo reg vai ter 2 bits. Sobrando assim 13 bits para o imm. Pelo que a gama é (complemento para 2) é -2^12 a 2^12 - 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:**
  - **Qual a dimensão do espaço de endereçamento desta arquitetura?**

    2^32 endereços possíveis

- **Qual a dimensão máxima da memória suportada por esta arquitetura expressa em bytes?**

  Como é bit addressable, o total da memória é 2^32 bits. Sendo que queremos o total em bytes. 2^32/8

- **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 expresso em bytes?**

  2^24 de espaço de endereçamento, logo 2^24 words de 32 bits. 2^24*32/8 = total de bytes da memória

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

    32 bits
  - **O que diferencia o registo $0 dos restantes registos de uso geral?**

    Contém o valor 0 e não pode ser alterado
  - **Qual o endereço do registo interno do MIPS a que corresponde a designação lógica $ra?**

    $31

- **No MIPS, um dos formatos de codificação de instruções é designado por R:**
  - **Quais os campos em que se divide este formato de codificação?**
  - **Qual o significado de cada um desses campos?**
  - **Qual o valor do campo opCode nesse formato?**

    Formato R:
    - **op - opcode** (6 bits, sempre 0 no tipo R)
    - **rs - register source** (5 bits, endereço do registo do primeiro operando fonte)
    - **rt - register operand** (5 bits, endereço do registo do segundo operando fonte)
    - **rd - register destination** (5 bits, endereço do registo de destino)
    - **shamt - shift amount** (5 bits, usado em instruções de shift, neste é 0)
    - **funct - function code** (6 bits, indica o tipo de operação)

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

    opcode = 0, logo é tipo R
    rs = $0
    rt = $0
    rd = $0

shamt = 00000
funct = 000000 (indica que é a intrução: sll)
Sendo que o shamt é 0, entao não faz nenhum shift.

- **O símbolo ” >> “ da linguagem C significa deslocamento à direita e é traduzido por SRL ou SRA (no caso do MIPS). Em que casos é que o compilador gera um SRL e quando é que gera um SRA?**

  Shift Right Logical (SRL) é destinado a operações do tipo **unsigned**, **entram 0’s**

  Shift Right Arithmetic (SRA) é destinado a operações do tipo **signed,** entram **0’s caso positivo**, entram **1’s caso negativo**

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

  or $4, $0, $15

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

    Tipo R
    op = 000000 (6 bits) -> 0 nas tipo R
    rs = 01101 (5 bits)
    rt = 11000 (5 bits)
    rd = 00101 (5 bits)
    shamt = 00000 (5 bits)
    funct = 100110 (6 bits)
    0000 0001 1011 1000 0010 1000 0010 0110 = 0x01B82826
  - **`sub $25,$14,$8`**

    Tipo R
    op = 000000 (6 bits) -> 0 nas tipo R
    rs = 01110 (5 bits)
    rt = 01000 (5 bits)
    rd = 11001 (5 bits)
    shamt = 00000 (5 bits)
    funct = 100010 (6 bits)
    0000 0001 1100 1000 1100 1000 0010 0010 = 0x01C8C822
  - **`sll $3,$9,7`**

    Tipo R
    op = 000000 (6 bits) -> 0 nas tipo R
    rs = 00000 (5 bits)
    rt = 01001 (5 bits)
    rd = 00011 (5 bits)

shamt = 00111 (5 bits)
funct = 000000 (6 bits)
0000 0000 0000 1001 0001 1001 1100 0000 = 0x000919C0

- `sra $18,$9,8`

  Tipo R
  op = 000000 (6 bits) -> 0 nas tipo R
  rs = 00000 (5 bits)
  rt = 01001 (5 bits)
  rd = 10010 (5 bits)
  shamt = 01000 (5 bits)
  funct = 000011 (6 bits)
  0000 0000 0000 1001 1001 0010 0000 0011 = 0x00099203

- **Traduza para instruções Assembly do MIPS a seguinte expressão aritmética, supondo x e y são inteiros e residentes em $t2 e $t5, respectivamente (apenas pode usar instruções nativas e não deverá usar a instrução de multiplicação):**
  y = -3 * x + 5;

  $t5 = -3 * $t2 + 5
  $t5 = -($t2 + $t2 + $t2) + 5

  add $t5, $t2, $t2
  add $t5, $t5, $t2
  sub $t5, $0, $t5
  addi $t5, $t5, 5

- **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;

  sra $a2, $a0, 2
  add $a2, $a2, $a1
  sra $t2, $t0, 5
  mult $t1, $t1, 2
  sub $t2, $t2, $t1

- **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:**
  - **add h, i, j #**

    h = i + j

- **addi j, j, 1** **#**
  add h, g, j **#**
  h = j + 1 + g

- **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?**
  h = 3 + 4
  h = 7

  h = 4 + 1 + 1
  h = 6

- **Qual a operação realizada pela instrução "slt" e quais os resultados possíveis?**
  slt = set if less than
  Fica 1 se for menor, 0 se for maior. O valor é armazenado no registo **$t1**

- **Qual o valor armazenado no registo $1 na execução da instrução "slt $1, $3, $7", admitindo que:**
  - **$3=5 e $7=23**
    $1 = 1
  - **$3=0xFE e $7=0x913D45FC**
    $3 = 0xFFFFFFFE > $7
    $1 = 0
- **Com que registo implícito comparam as instruções "bltz", "blez", "bgtz" e "bgez"?**
  Comparam $1 com $0

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

    (se $15 < $3 -> exit)
    slt $1, $15, $3 (se $15 < $3 então $1 = 1)
    bne $1, $0, exit (se $1 != 0 -> exit)
  - **ble $6,$9,exit**

    (se $6 <= $9 -> exit)
    slt $1, $9, $6 (se $9 < $6 então $1 = 1)
    beq $1, $0, exit (se $1 = 0 -> exit)
  - **bgt $5,0xA3,exit**
    addi $1, $0, 0xA3 (se $5 > 0xA3 -> exit)
    slt $1, $1, $5 (se 0xA3 < $5 então $1 = 1)
    bne $1, $0, exit (se $1 != 0 -> exit)

- **bge $10,0x57,exit**

```
                                (se $10 >= 0x57 -> exit)
    slti $1, $10, 0x57          (se $10 < 0x57 então $1 = 1)
    beq $1, $0, exit            (se $1 = 0 -> exit)
```

- **blt $19,0x39,exit**

```
                                (se $19 < 0x39 -> exit)
    slti $1, $19, 0x39          (se $19 < 0x39 então $1 = 1)
    bne $1, $0, exit            (se $1 != 0 -> exit)
```

- **ble $23,0x16,exit**

```
    addi $1, $0, 0x16           (se $23 <= 0x16 -> exit)
    slt $1, $1, $23             (se 0x16 < $23 então $1 = 1)
    beq $1, $0, exit            (se $1 = 0 -> exit)
```

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

  No caso do **while()** o teste condicional é executado no **início** do ciclo
  No caso do **do...while()** o teste condicional é efetuado no **fim** do ciclo

- **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, respectivamente):**

  - **if(a > b && b != 0)**
    c = b << 2;
    else
    c = (a & b) ^ (a | b);

  - **if(a > 3 || b <= c)**
    c = c – (a + b);
    else
    c = c + (a – 5);

- **Qual o modo de endereçamento usado pelo MIPS para ter acesso a palavras residentes na memória externa?**

  Endereçamento indirecto por registo com deslocamento

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

  $3 = registo de destino onde é armazenada a palavra que se encontra no registo calculado (conteúdo do registo $5 + 0x24)

  $5 = o conteúdo deste registo é um endereço da memória
  0x24 = offset

- **Qual é o formato de codificação das instruções de acesso à memória no MIPS e qual o significado de cada um dos seus campos?**

  Formato I:

  - **op - opcode** (6 bits, sempre 0 no tipo R)
  - **rs - register source** (5 bits, endereço do registo do primeiro operando fonte)
  - **rt - register operand** (5 bits, endereço do registo do segundo operando fonte)
  - **offset** (16 bits, deslocamento em complementos para 2)

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

  sw = store word, transfere 4 bytes de um registo interno para a memória
  sb = store byte, transfere um byte de um registo interno para a memória

- **O que distingue as instruções "lb" e "lbu"?**

  lb = load byte signed, transfere um byte da memória para o registo interno - **24 bits mais significativos são colocados a 1 se o byte tem 1 no bit mais significativo e a 0 se for 0**
  lbu = load byte unsigned, load byte signed, transfere um byte da memória para o registo interno - **24 bits mais significativos são colocados a 0**

- **O que acontece quando uma instrução lw/sw acede a um endereço que não é múltiplo de 4?**

  O MIPS gera uma excepção

- **Traduza para assembly do MIPS os seguintes trechos de código de linguagem C (atribua registos internos para o armazenamento das variáveis i e k) :**
  - **int i, k;**
    for(i=5, k=0; i < 20; i++, k+=5);

  - **int i=100, k=0;**
    for(; i >= 0;)
    {
    i--;
    k -= 2;
    }
  - **unsigned int k=0;**
    for(; ;)
    {

```
        k += 10;
    }
```

- ```
  int k=0, i=100;
  do
  {
        k += 5;
  } while(--i >= 0);
  ```

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

  Tipo I:
  op (6 bits) = 0x23 = 10 0011
  rs (5 bits) = 0x5 = 0 0101
  rt (5 bits) = 0x3 = 0 0011
  offset (16 bits) = 0x24 = 0000 0000 0010 0100
  1000 1100 1010 0011 0000 0000 0010 0100 = 0x8CA30024

- **Suponha que a memória externa foi inicializada, a partir do endereço 0x10010000, com os valores 0x01=0x10010000,0x02=0x10010001,0x03=0x10010002,0x04=0x10010003, 0x05=0x10010004.. 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?**

  0x10010000 + 0x24 = 0x10010024 (múltiplo de 4 pelo que o byte começa aqui)
  Neste endereço está armazenado o valor 0x23
  $3 = 0x26 0x25 0x24 0x23

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

    0x10010000 + 0xA3 = 0x100100A3 (não é múltiplo de 4)
    O número de endereço múltiplo de 4 anterior a este é 0x100100A0.
    No endereço está armazenado o valor 0xA2
    $3 = 0x00 0x00 0x00 0xA2

  - **lb $4,0xA3($5)**

    0x10010000 + 0xA3 = 0x100100A3 (não é múltiplo de 4)
    O número de endereço múltiplo de 4 anterior a este é 0x100100A0.
  ```

No endereço está armazenado o valor 0xA2
$3 = 0xFF 0xFF 0xFF 0xA2 (como 0xA2 é 1010 0010 fica com o sinal do bit mais significativo)

- **Quantos bytes são reservados em memória por cada uma das diretivas:**
  - **L1: .asciiz "Aulas5&6T"**
    1 char = 1 byte
    9 bytes + 1 byte do ‘\0’
  - **L2: .byte 5,8,23**
    3 bytes
  - **L3: .word 5,8,23**
    3*4 = 12 bytes (+ alinhamento)
  - **L4: .space 5**
    5 bytes

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

```
0x10010020      ??
0x1001001F      ??
0x1001001E      ??
0x1001001D      ??
0x1001001C      ??
-----------------------------
0x1001001B      0x00
0x1001001A      0x00
0x10010019      0x00
0x10010018      0x17    (23)
0x10010017      0x00
0x10010016      0x00
0x10010015      0x00
0x10010014      0x08
0x10010013      0x00
0x10010012      0x00
0x10010011      0x00
0x10010010      0x05
0x1001000F      ??
0x1001000E      ??
0x1001000D      ??
-----------------------------
0x1001000C      0x17    (23)
0x1001000B      0x08
0x1001000A      0x05
-----------------------------
0x10010009      '\0'
0x10010008      'T'
0x10010007      '6'
0x10010006      '&'
0x10010005      '5'
0x10010004      's'
0x10010003      'a'
0x10010002      'l'
0x10010001      'u'
0x10010000      'A'
```

- **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:".**

  L1: 0x10010000
  L2: 0x1001000A
  L3: 0x10010010
  L4: 0x1001001C

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

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

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

  (offset = PC_destino - PC_atual) (corresponde ao número de instruções até à desejada)
  (16 bits mas os 2 bits menos significativos não são representados, são 00)

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

  Como o offset tem apenas 16 bits e precisamos de 32, vai ser estendido (complemento para 2) e somado/subtraído ao PC (program counter) para dar a instrução para onde saltar.
  PC_destino = PC_atual + offset*4

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

  beq/bne = Formato I
  j = Formato J
  jr = Formato R

- **A partir do código máquina de uma instrução "j", como é formado o endereço-alvo (Jump Target Address)?**

  PC_destino = label*4 concatenado com os 4 bits mais significativos do PC_atual

- **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);`

  Queremos saber b[5]. Sendo que o ponteiro p aponta para o primeiro endereço do array (primeira posição). p+20 (5*4) vai apontar para a 5ª.
  Como queremos o valor guardado usamos *.
  Logo, a = *(p + 20)

- **Assuma que as variáveis `f, g, h, i` e `j` correspondem aos registos `$t0`, `$t1`, `$t2`, `$t3` e `$t4` respectivamente. Considere que o endereço**

base dos arrays 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]]`

- **Qual a tradução para assembly de cada uma das instruções C indicadas?**

- **Quantas instruções assembly são necessárias para cada uma das instruções C indicadas? E quantos registos auxiliares são necessários?**

- **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]=

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

- **Pretende-se escrever uma função para a troca do conteúdo de duas variáveis (troca(a, 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

- **Na instrução "`jr $ra`", como é obtido o endereço-alvo?**

  O rs (jr é instrução do tipo R), registo do CPU, armazena o endereço alvo da memória (32bits). Logo, é um **endereçamento indirecto por registo**.

- **Qual é o menor e o maior endereço para onde uma instrução "j", residente no endereço de memória 0x5A18F34C, pode saltar?**

  range: +- 256 MBytes (2^28) dentro dos 4 significativos

  0x5 = 4 bits mais significativos do PC atual
  Como pede o endereço queremos a label (26 bits + 2 bits que são ignorados na instrução = 28)
  0x50000000 a 0x5FFFFFFC

- **Qual é o menor e o maior endereço para onde uma instrução "beq", residente no endereço de memória 0x5A18F34C, pode saltar?**

  range: +- 128 kBytes (32*4 kBytes ou 2^17)

  0x5A180000 a 0x5A19FFFC

- **Qual é o menor e o maior endereço para onde uma instrução "jr", residente no endereço de memória 0x5A18F34C pode saltar?**

  range: +- 4 GBytes (tamanho da memória)

  Pode ir de **0x00000000 a 0xFFFFFFFC**

- **Qual a gama de representação da constante nas instruções aritméticas imediatas?**

  imediato = 16bits
  No caso de instruções aritméticas **funcionam com signed**, logo a gama de representação é **2^15 a 2^15 - 1**

- **Qual a gama de representação da constante nas instruções lógicas imediatas?**
  imediato = 16bits
  No caso das instruções lógicas funcionam com **unsigned**, logo a gama de representação é **0 a 2^16 - 1**

- **Por que razão não existe, no ISA do MIPS, uma instrução que permita manipular diretamente uma constante de 32 bits?**

- **Como é que, no assembly do MIPS, se podem manipular constantes de 32 bits?**
  Usando lui, colocando os 16 mais significativos no registo de destino

- **Apresente a decomposição em instruções nativas das seguintes instruções virtuais:**
  - **li $6,0x8B47BE0F**
  - **xori $3,$4,0x12345678**
  - **addi $5,$2,0xF345AB17**
  - **beq $7,100,L1**
  - **blt $3,0x123456,L2**

- **O que é uma sub-rotina?**
  Quando uma função (caller) chama outra função (calle)

- **Qual a instrução do MIPS usada para saltar para uma sub-rotina?**
  jal subrotina

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

- **Quais as operações que são sequencialmente realizadas na execução de uma instrução "jal"?**

- **Qual o nome virtual e o número do registo associado à execução dessa instrução?**

- **No caso de uma sub-rotina ser simultaneamente chamada e chamadora (sub-rotina intermédia) que operações é obrigatório realizar nessa sub-rotina?**
  É necessário fazer a salvaguarda na stack dos registos $s.. e registos $t.. caso sejam usados antes da função e retirados da stack após a chamada.
- **Qual a instrução usada para retornar de uma sub-rotina?**
  jr $ra
- **Que operação fundamental é realizada na execução dessa instrução?**

- **O que é uma stack e qual a finalidade do stack pointer?**

- **Como funcionam as operações de push e pop?**

  salvaguarda do $t0 por exemplo:

  push:

  ```
  addu  $sp, $sp, -4
  sw    $ra, 0($sp)
  sw    $t0, 4($sp)
  ```

  pop:

  ```
  lw    $t0, 4($sp)
  lw    $ra, 0($sp)
  addu  $sp, $sp, 4
  ```

- **Por que razão as stacks crescem normalmente no sentido dos endereços mais baixos?**

  Como a memória cresce de baixo para cima, como não sabemos qual é o último endereço que está cheio para poder começar a crescer a stack, então a stack cresce de cima para baixo para não haver espaço vazio desnecessário.

- **Quais as regras para a implementação em software de uma stack no MIPS?**

- **Qual o registo usado, no MIPS, como stack pointer?**

  $sp

- **De acordo com a convenção de utilização de registos no MIPS:**
  - **Que registos são usados para passar parâmetros e para devolver resultados de uma sub-rotina?**
  - **Quais os registos que uma sub-rotina pode livremente usar e alterar sem necessidade de prévia salvaguarda?**

    $a.., $v..(registos de retorno e de parâmetros)
  - **Quais os registos que uma sub-rotina não pode alterar?**

    $t.. (na sub-rotina faz-se a salvaguarda de registos $s..)
  - **Quais os registos que uma sub-rotina chamadora tem a garantia que a sub-rotina chamada não altera?**

    $s.. (na sub-rotina chamadora faz-se salvaguarda de registo $t..)
  - **Em que situação devem ser usados registos "$sn"? f.Em que situação devem ser usados os restantes registos: $tn, $an e $vn?**

- **De acordo com a convenção de utilização de registos do MIPS:**
  - **Que registos podem ter que ser copiados para a stack numa sub-rotina intermédia?**
  - **Que registos podem ter que ser copiados para a stack numa sub-rotina terminal?**

- **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);

  $a0 = a
  $a1 = b
  $a2 = c
  $a3 = d
  $v0 = retorno

- **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).**

- **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;
}
```

- **Determine a representação em complemento para 2 com 16 bits das seguintes quantidades:**
  - **5** 0000 0000 0000 0101
  - **-3** 1111 1111 1111 1101
  - **-128** 1111 1111 1000 0000
  - **-32768 31** não dá para representar

- **-8** 1111 1111 1111 1000
- **256** 0000 0001 0000 0000
- **-32** 1111 1111 1110 0000

- **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**

- **Determine a representação das quantidades do exercício anterior em hexadecimal com 16 bits (também codificadas em complemento para 2).**

- **Como é realizada a deteção de overflow em operações de adição com quantidades sem sinal?**

  Se o carry = 1, sinaliza overflow

- **Como é realizada a deteção de overflow em operações de adição com quantidades com sinal (codificadas em complemento para 2)?**

  Se o carry in for diferente do carry out (ou exclusivo dos dois)

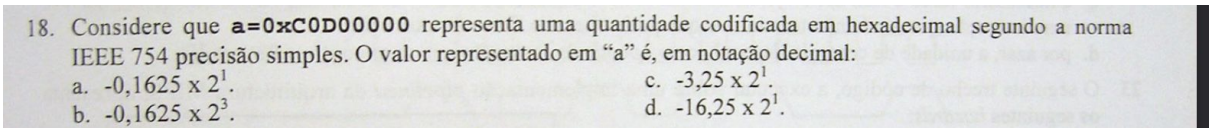
- **Considere os seguintes pares de valores em $s0 e $s1:**
  $s0 = 0x70000000 $s1 = 0x0FFFFFFF
  $s0 = 0x40000000 $s1 = 0x40000000
  - **Qual o resultado produzido pela instrução add $t0, $s0, $s1?**
  - **Para a alínea anterior os resultados são os esperados ou ocorreu overflow?**
  - **Qual o resultado produzido pela instrução sub $t0, $s0, $s1?**
  - **Para a alínea anterior os resultados são os esperados ou ocorreu overflow?**
  - **Qual o resultado produzido pelas instruções: add $t0, $s0,$s1 add $t0, $t0,$s1**
  - **Para a alínea anterior os resultados são os esperados ou ocorreu overflow?**

- **Para a multiplicação de dois operandos de "m" e "n" bits, respetivamente, qual o número de bits necessário para o armazenamento do resultado?**

  m+n bits

Mini-resumo vírgula flutuante:

Exemplo:



18. Considere que **a=0xC0D00000** representa uma quantidade codificada em hexadecimal segundo a norma IEEE 754 precisão simples. O valor representado em "a" é, em notação decimal:
a. $-0{,}1625 \times 2^{1}$.
b. $-0{,}1625 \times 2^{3}$.
c. $-3{,}25 \times 2^{1}$.
d. $-16{,}25 \times 2^{1}$.

(no caso de 64bits a bias é 1023 e são 11 exponential bits)

Neste caso temos 1 float de 32 bits

a = 0xC0D00000

a = 1100 0000 1101 0000 0000 0000 0000 0000

- Como o bit mais significativo é 1, então o número é **negativo**
- Como a precisão é simples então temos **8 bits de exponential bits** e **127 (0x7F) de bias:**

  100 0000 1 = 0x81

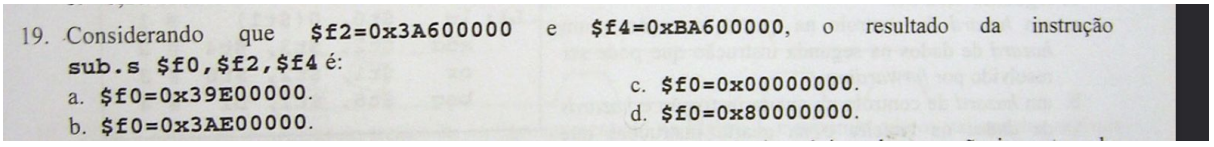
  0x81 - 0x7F = 0x02 = 2 em decimal
- Os restantes bits são **mantissa:**

  101 0000 0000 0000 0000 0000

  1x2^-1 + 1x2^-3 = 1/2 + 1/8 = ⅝ = 0.625

-**1.625 x 2² = -162.5**, sendo assim a resposta poderá ser tanto a b) como a d)

Exemplo:



19. Considerando que **$f2=0x3A600000** e **$f4=0xBA600000**, o resultado da instrução **sub.s $f0,$f2,$f4** é:
a. **$f0=0x39E00000**.
b. **$f0=0x3AE00000**.
c. **$f0=0x00000000**.
d. **$f0=0x80000000**.

$f2 = 0**011 1010 0**110 0000 0000 0000 0000 0000 (positivo)

$f4 = 1**011 1010 0**110 0000 0000 0000 0000 0000 (negativo)

exponential bits = 0x74 (igual nas duas)

0x7F - 0x74 = 0x0B = 11

mantissa = 110 0000 0000 0000 0000 0000 (igual nas duas)

1.1100x2^11 + (-1.1100x2^11)

subtraindo dá 0

VOU SKIPAR PARA SINGLE E MULTI CYCLE E DEPOIS CONTINUO O RESTO

skipei do ex 96 a ex 120

Modelo de von Neumann (multi cycle):

- Tem **uma memória** para dados e código.
- **Dois ciclos de relógio**, um para dados e outros para código
- **Frequência máxima:** maior dos tempos de atraso dos elementos operativos (ALU, Memória, File Register)
- **Unidade de controlo: máquina de estados (moore)** em que o primeiro e segundo ciclo é comum na execução todas as instruções
- **Unidade aritmética e lógica:** ALU (apenas)

Modelo de Harvard (single cycle):

- Tem **duas memórias** independentes para dados e código
- **Um ciclo de relógio** para aceder a dados e código
- **Frequência máxima:** maior dos atrasos cumulativos da instrução mais lenta (lw)
- **Unidade de controlo: elemento combinatório que gera sinais** conforme o campo opcode das instruções
- **Unidade aritmética e lógica:** ALU e dois somadores

Modelo de Harvard (pipeline):

- **Provém do single cycle**
- **Hazards:**
  - **Structural:** ocorre quando mais do que uma instrução necessita de aceder ao mesmo **hardware**
  - **Solução:**
  - **Data:** resulta da **dependência** existente entre o resultado calculado por uma instrução e o operando usado por outra que segue mais atrás no pipeline (i.e., mais recente)
  - **Solução:**
  - **Control:** ocorre quando é necessário fazer o **instruction fetch** de uma nova instrução e existe numa etapa mais avançada do pipeline uma instrução que pode alterar o **fluxo de execução** e que ainda não terminou

- **Solução:**

Datapath (ambas):

- Leitura da memória/registos do CPU é assíncrona
- Escrita da memória/registos do CPU é feita síncrona no rising edge do clock

- **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.**
  Memórias externas: leitura – 175ps, escrita – 150ps; File register: leitura – 25ps, escrita – 35ps; Unidade de Controlo: 10ps; Somadores: 50ps; Outros: 0ns; Escrita noutros elementos de estado: 20ps. Setup time dos elementos de estado: 5ps

  fmax = 2GHz
  Tmax = 1/2 = 0.5 ns (tempo total do lw)
  É util saber qual o caminho das instruções no single cycle
  tEXEC = tRM+ max(tRFR, tCNTL, tSE) + tALU+ tRM+ tWFR
  0.5ns = 175ps + 25ps + max(1ps, 10ps, 5ps) + tALU + 175 + 35ps
  500ps = 420ps + tALU
  tALU = 80ps

- **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 – 8ns, escrita – 10ns; File register: leitura – 2ns, escrita – 3ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns. Escrita noutros elementos de estado: 2ns; Setup time dos elementos de estado: 1ns

  tEXEC = tRM+ max(tRFR, tCNTL, tSE) + tALU+ tWM
  tEXEC = 8ns + max(2ns, 1ns, 1ns) + 6ns + 10ns
  tEXEC = 8 + 2 + 6 + 10
  tEXEC = 26ns
  fmax = 1/26ns = 0.038 GHz = 38MHz

- **Determine, numa implementação single-cycle da arquitetura MIPS, a frequência máxima de operação imposta pela instrução "beq", assumindo os atrasos a seguir indicados, é: Memórias externas: leitura – 6ns, escrita – 8ns; File register: leitura – 2ns, escrita – 3ns; Unidade de Controlo: 1ns; ALU (qualquer operação): 6ns; Somadores: 4ns; Outros: 0ns. Escrita noutros elementos de estado: 1ns; Setup time dos elementos de estado: 0.5ns**

  tEXEC = tRM+ max(max(tRFR , tCNTL) + tALU, tSE+ tSL2+ tADD) + tstPC
  tEXEC = 6ns + max(max(2ns, 1ns) + 6ns, 0.5ns + 0 + 0)
  tEXEC= 6 + max(2 + 6, 0.5)
  tEXEC = 6 + 8

tEXEC = 14 ns
fmax = 1/14ns = 0.07 GHz = 70MHz

- **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, é: Memórias externas: leitura – 9ns, escrita – 11ns; File register: leitura – 3ns, escrita – 4ns; Unidade de Controlo: 2ns; ALU (qualquer operação): 7ns; Somadores: 4ns; Outros: 0ns. Escrita noutros elementos de estado: 2ns; Setup time dos elementos de estado: 1ns**

  tEXEC = tRM+ max(tRFR, tCNTL) + tALU+ tWFR
  tEXEC = 9ns + max(3ns, 2ns) + 7ns + 4ns
  tEXEC = 9 + 3 + 7 + 4
  tEXEC = 23 ns

- **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.**

  Modelo de Harvard (single cycle):
  - Tem duas memórias independentes para dados e código
  - Um ciclo de relógio para aceder a dados e código
  - Frequência máxima: maior dos atrasos cumulativos da instrução mais lenta (lw)
  - Unidade de controlo: elemento combinatório que gera sinais conforme o campo opcode das instruções
  - Unidade aritmética e lógica: ALU e dois somadores

- **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?**

  O endereço da instrução atual (???)

- **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	ALU Src	MemTo Reg	Reg Write	Mem Read	MemWrite
lw	100011	00	0	0	1?	1	0	1	0
sw	101011	00	0	0	1	0	0	0	1
addi	001000	00	0	0					
slti	001010	11	0	0					

beq	**000100**	01	1	0	0	0	0	0	0
R-Format	**000000**	10	0	1	0	0	1	0	0

-