Universidade de Aveiro
Dep. de Eletrónica, Telecomunicações e Informática
Arquitetura de Computadores I

Nome _____

Notas importantes!

1. Verifique, para todas as questões, qual a resposta correta e assinale com uma cruz a sua escolha na tabela no fim. Por cada resposta incorreta será descontado, à cotação global, um número igual ao da cotação da respetiva pergunta.
2. Pode usar até um máximo de 4 respostas duplas (por cada dupla: 0 certas descontam até 2/3, 1 certa conta até 1/3). Se usar mais de 4 duplas, serão aceites as 4 primeiras e as restantes serão consideradas respostas erradas.
3. Durante a realização do teste não é permitida a permanência junto do aluno, mesmo que desligado, de qualquer dispositivo eletrónico não expressamente autorizado (neste lote incluem-se calculadoras, telemóveis e *smartwatches*). A sua deteção durante a realização do exame implica a imediata anulação do mesmo.

Grupo I

1. Considere que no endereço de memória `0x00400020` se encontra armazenada a instrução `beq $6,$2,label`. Se `label` corresponder ao endereço `0x00400000`, o valor dos 16 bits menos significativos do código máquina dessa instrução será:
   a. `0xFFFB`
   b. `0xFFF8`
   c. `0xFFF4`
   d. `0xFFF7`

2. Suponha que `$3=0xFC4756E8` e `$2=0x00FFFF00`. A instrução "`xor $6,$3,$2`" produz o seguinte resultado armazenado em `$6`:
   a. `0x3047A9E8`
   b. `0xFC0000E8`
   c. `0xFCB8A9E8`
   d. `0x00B80017`

3. Na arquitetura básica de um sistema computacional o *Control Bus* permite:
   a. transferir dados entre os registos do CPU.
   b. especificar a natureza de uma operação efetuada sobre a memória.
   c. transferir o código máquina das instruções para o *instruction register*.
   d. identificar, na memória, a origem/destino dos dados transferidos.

4. A arquitetura MIPS não permite realizar operações lógicas e/ou aritméticas sobre o conteúdo de posições da memória externa, ou, simultaneamente, sobre o conteúdo de registos internos e posições da memória externa. Por essa razão:
   a. permite, para a mesma instrução e num mesmo ciclo de relógio, operar sobre a ALU e efetuar uma operação de leitura da memória.
   b. todas as variáveis de um dado programa devem residir em registos internos.
   c. é menos eficiente do que arquiteturas que permitem esse tipo de operações.
   d. é considerada como sendo uma arquitetura do tipo *Load-Store*.

5. Um endereço de memória externa num sistema computacional é:
   a. a informação armazenada em cada posição de memória.
   b. um número único que identifica cada posição de memória.
   c. o valor transferido de um registo do CPU para uma posição de memória.
   d. o meio de identificar o tipo de operação realizada sobre essa memória.

6. Os processadores da arquitetura hipotética ZXIoT implementam um total de 130 instruções. Todas as instruções são codificadas com 32 bits, e um dos formatos tem 5 campos: *opcode*, três campos para identificar registos internos e um campo para codificar constantes no intervalo `[-16384,+16383]`. Considerando que o número de bits do campo *opcode* é apenas o necessário para codificar as 130 instruções, conclui-se que o número de registos é:
   a. 32 b. 16 c. 8 d. 4

Exame Arquitetura de Computadores I

7. Considere que o segmento de dados de um programa contém as seguintes diretivas:

```
L1: .word   0, 0    4,4
L2: .asciiz "ExameJan23"
    .align  3
L3: .space  16
L4:
```

Se o endereço correspondente ao *label* `L1` for `0x10010A10`, o endereço a que corresponde o *label*

a. `0x10010A40` b. `0x10010A34` c. `0x10010A30` d. `0x10010A`

8. Considere um sistema computacional em que existe uma memória do tipo ROM (*Read Only* armazena programas, e uma memória do tipo RAM que armazena dados. Sabendo que as duas me em espaços de endereçamento distintos, pode afirmar-se que estamos perante uma arquitetura:

a. do tipo *Harvard*.
b. mista *Harvard/von Neumann*.
c. do tipo *von Neumann*.
d. do tipo *von Neumann* com armazenamento não volátil de programas.

Considere o seguinte segmento de código *assembly*:

```
lui   $3, 0xC614
ori   $4, $3, 0x19A5
sw    $4, 0x40($6)
lbu   $8, 0x41($6)
```

inal da execução do código anterior, o valor de `$8`, num MIPS *little endian*, é:

a. `0x000000C6` b. `0x00000019` c. `0x00000014` d. `0x00`

pal modo de endereçamento utilizado na instrução `lb $t0,4($ra)` é:

to por registo com deslocamento.
o por registo.
ao PC.
direto.

nstrução `jal sqblz`, armazenada no endereço de memória `0x004004CC`. O lab
`x00400518`. Na conclusão da execução da instrução, o registo `$ra`:

nado o valor `0x00400518`.
ado o valor `0x004004D0`.
do o valor `0x0040051C`.
lo o valor `0x004004CC`.

7. Considere que o segmento de dados de um programa contém as seguintes diretivas:

```
L1: .word 0, 0
L2: .asciiz "Exame-Jan23"
    .align 3
L3: .space 16
L4:
```

Se o endereço correspondente ao *label* **L1** for **0x10010A10**, o endereço a que corresponde o label **L4** é:

a. `0x10010A40` b. `0x10010A34` c. `0x10010A30` d. `0x10010A38`

8. *Considere um sistema computacional em que existe uma memória do tipo* ROM (*Read Only Memory*), q[...] armazena programas, e uma memória do tipo RAM que armazena dados. Sabendo que as duas memórias resid[...] em espaços de endereçamento distintos, pode afirmar-se que estamos perante uma arquitetura:

a. do tipo *Harvard*.
b. mista *Harvard/von Neumann*.
c. do tipo *von Neumann*.
d. do tipo *von Neumann* com armazenamento não volátil de programas.

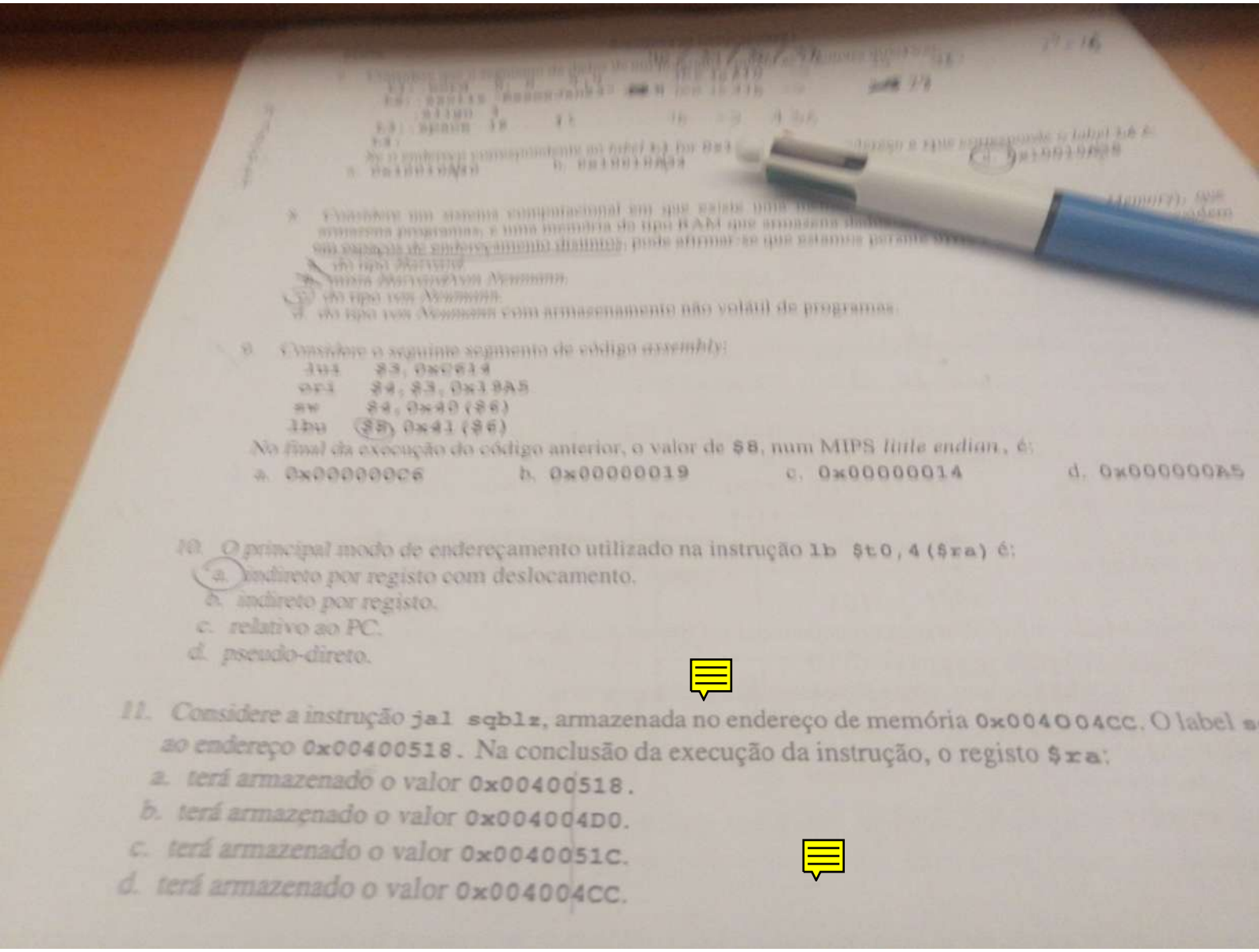
9. Considere o seguinte segmento de código *assembly*:

```
lui  $3,0xC6...
ori  $4,$3,0x19...
sw   $4,0x40($6)
lbu  $8,0x41($6)
```

No final da execução do código anterior, o valor de **$8**, num MIPS *little endian* [...]

a. `0x000000C6` b. `0x00000019` c. `0x00000014` d. `0x00`[...]

[...] modo de endereçamento utilizado na instrução `lb $t0,4($ra)` é:



9. Considere o seguinte segmento de código *assembly*:

```
lui   $3, 0xC614
ori   $4, $3, 0x19A5
sw    $4, 0x40($6)
lbu   $8, 0x41($6)
```

No final da execução do código anterior, o valor de `$8`, num MIPS *little endian*, é:

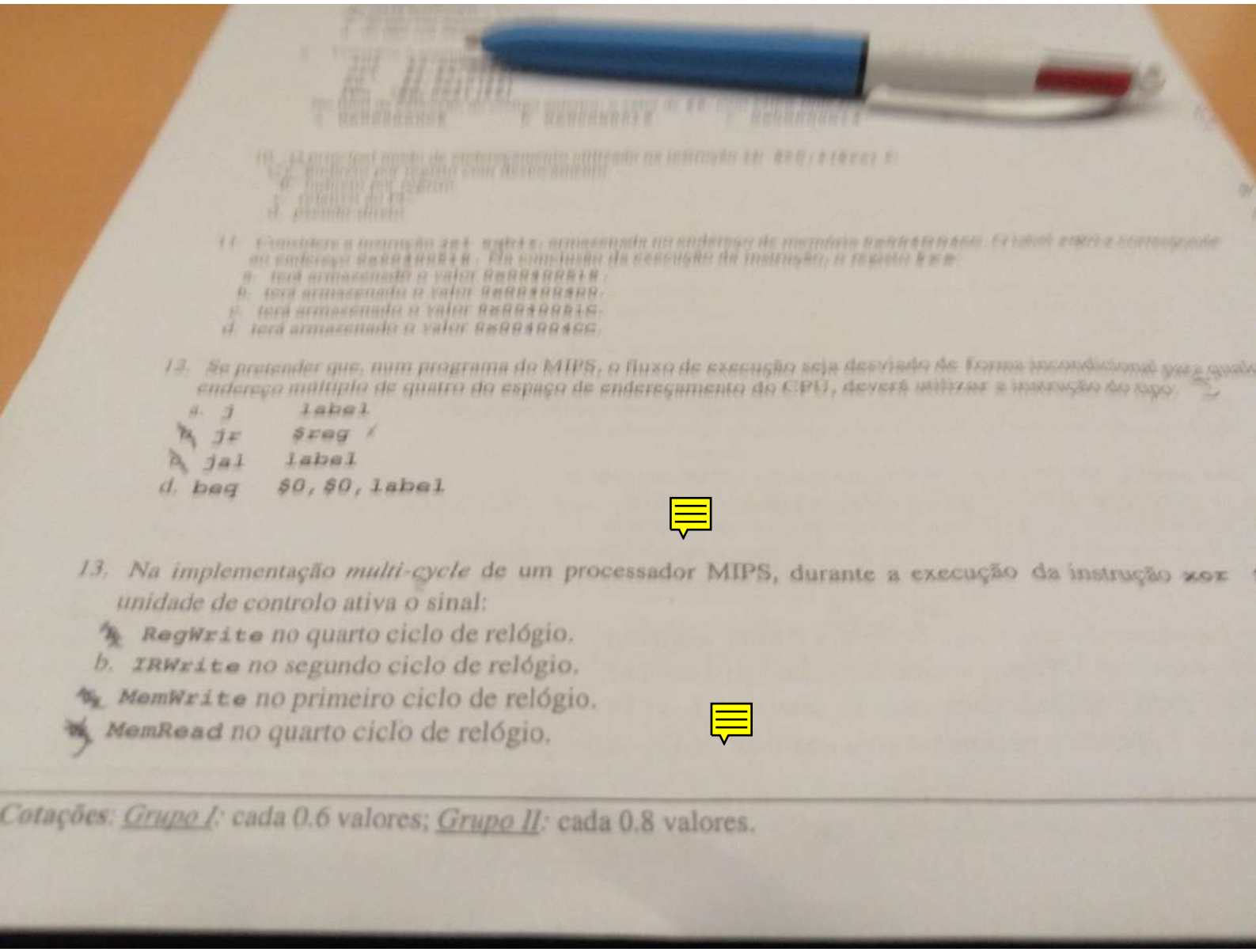
a. `0x000000C6` b. `0x00000019` c. `0x00000014` d. `0x000000A5`

10. O principal modo de endereçamento utilizado na instrução `lb $t0,4($ra)` é:

a. indireto por registo com deslocamento.
b. indireto por registo.
c. relativo ao PC.
d. pseudo-direto.

11. Considere a instrução `jal sqblz`, armazenada no endereço de memória `0x004004CC`. O label `sqblz` corresponde ao endereço `0x00400518`. Na conclusão da execução da instrução, o registo `$ra`:

a. terá armazenado o valor `0x00400518`.
b. terá armazenado o valor `0x004004D0`.
c. terá armazenado o valor `0x0040051C`.
d. terá armazenado o valor `0x004004CC`.



12. Se pretender que, num programa do MIPS, o fluxo de execução seja desviado de forma incondicional para qualquer endereço múltiplo de quatro do espaço de endereçamento do CPU, deverá utilizar a instrução do tipo:

a. `j label`
b. `jr $reg`
c. `jal label`
d. `beq $0, $0, label`

13. Na implementação *multi-cycle* de um processador MIPS, durante a execução da instrução `xor` a unidade de controlo ativa o sinal:

a. `RegWrite` no quarto ciclo de relógio.
b. `IRWrite` no segundo ciclo de relógio.
c. `MemWrite` no primeiro ciclo de relógio.
d. `MemRead` no quarto ciclo de relógio.

Cotações: Grupo I: cada 0.6 valores; Grupo II: cada 0.8 valores.

14. A deteção de *overflow* numa operação de adição de números inteiros **com sinal** faz-se através:
   a. da avaliação do **bit** mais significativo do resultado.
   b. do *xor* entre o *carry in* e o *carry out* da célula de 1 **bit** mais significativa do resultado.
   c. do *xor* entre os 2 **bits** mais significativos do resultado.
   d. da avaliação do *carry out* do **bit** mais significativo do resultado.

15. Considerando que `$1=0xFFFFFFFA` e `$4=0xFFFFFFFD`, o conteúdo dos registos `HI` e `LO` após a execução da instrução `mult $1, $4` (multiplicação *signed*) é:
   a. `HI=0x00000000, LO=0x00000012`
   b. `HI= ????????, LO=0x00000018`
   c. `HI=0xFFFFFFFF, LO=0xFFFFFFEE`
   d. `HI=0x00000000, LO=0xFFFFFFEE`

16. Em linguagem C, o código que permite atribuir o valor `0x1F` ao elemento de índice 5 do *array* "`t`" é:

a.
```
int t[20];
int *mp;

mp=t[0];
*(mp+5)=0x1F;
```

b.
```
int t[20];
int *mp;

mp=t;
*(mp+5)=0x1F;
```

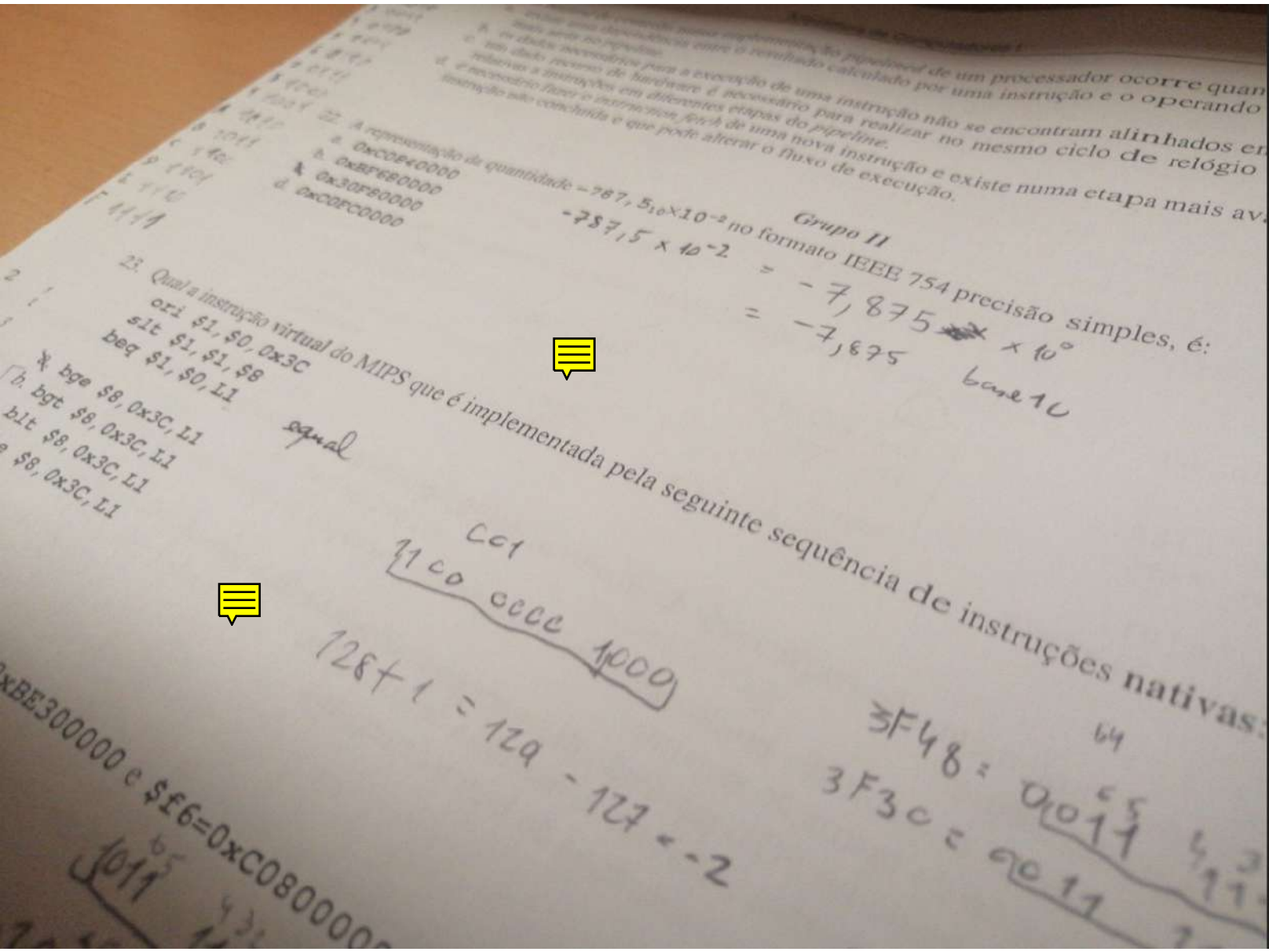
c.
```
int t[20];
int *mp;

mp=&t;
*(mp+20)=0x1F;
```

d.
```
int t[20];
int *mp;

mp=t;
*(mp)+5=0x1F;
```

17. Numa implementação ***single-cycle*** da arquitetura MIPS, semelhante à apresentada nas aulas, na transi
sinal de relógio:
   a. é escrito o resultado produzido pela instrução em execução no elemento de estado respetiv
   execução da instrução seguinte.
   b. é armazenado no PC o resultado da operação realizada na ALU para cálculo de PC+4.
   c. é realizada a leitura síncrona da memória de instruções.
   d. é realizada a leitura assíncrona do banco de registos.



27. Na situação apresentada na questão anterior, admita que no instante zero, correspondente [illegible] sinal de relógio, vai iniciar-se o *instruction fetch* da primeira instrução. O valor à saída da ALU no oitavo ciclo de relógio, contado a partir do instante zero, é:

a. 0x00000005
b. 0x0000684C
c. 0x00000001
d. 0x093B8630

28. Numa implementação *pipelined* da arquitetura MIPS, o *hazard* de dados existente na sequência de instruções `lw $1,0($2)` seguida de `beq $1,$3,target`, pode ser resolvido:

a. com *stall* durante 2 ciclos de relógio.
b. sem recurso a *stalling*, com *forwarding* de **EX/MEM** para **EX**.
c. com *stall* durante 1 ciclo de relógio seguido de *forwarding* de **MEM/WB** para **EX**.
d. com *stall* durante 1 ciclo de relógio seguido de *forwarding* de **EX/MEM** para **ID**.

29. O código máquina da instrução `lw $15,-4($3)`, representado em hexadecimal, é (considere instrução, *opcode* = 0x23):

a. 0x8DE3FFFC
b. 0x8C83FFF8
c. 0x8C6FFFFC
d. 0x8DE3FFF8

0x23 | $15 | $3 | -4

Grupo I: cada 0.6 valores; Grupo II: cada 0.8 valores.

14. A deteção de *overflow* numa operação de adição de números inteiros **com sinal**. Faz-se através:
a. da avaliação do bit mais significativo do resultado.
b. do XOR entre o carry in e o carry out da célula de 1 bit mais significativa do resultado.
c. do XOR entre os 2 bits mais significativos do resultado.
d. da avaliação do carry out do bit mais significativo do resultado.

15. Considerando que $t1=0xFFFFFFFA e $t4=0xFFFFFFFD, o conteúdo dos registos HI e LO após a execução da instrução mult $t1, $t4 (multiplicação signed) é:
a. HI=0x00000000, LO=0x00000012
b. HI=0xFFFFFFFF, LO=0x00000018
c. HI=0xFFFFFFFF, LO=0xFFFFFFEE
d. HI=0x00000000, LO=0xFFFFFFEE

16. Em linguagem C, o código que permite atribuir o valor 0x1F ao elemento de índice 5 do array "t" é:

a.	b.	c.	d.
`int t[20];` `int *mp;` `mp=t[5];` `*(mp+5)=0x1F;`	`int t[20];` `int *mp;` `mp=t;` `*(mp+5)=0x1F;`	`int t[20];` `int *mp;` `mp=&t;` `*(mp+20)=0x1F;`	`int t[20];` `int *mp;` `mp=t;` `*(mp)+5=0x1F;`

17. Numa implementação *single-cycle* da arquitetura MIPS, semelhante à apresentada nas aulas, na transição ativa do sinal de relógio:
a. é escrito o resultado produzido pela instrução em execução no elemento de estado respetivo e inicia-se a execução da instrução seguinte.
b. é armazenado no PC o resultado da operação realizada na ALU para cálculo de PC+4.
c. é realizada a leitura síncrona da memória de instruções.
d. é realizada a leitura assíncrona do banco de registos.

A unidade de *forwarding* de uma implementação *pipelined* da arquitetura MIPS é um bloco:
combinatório responsável pelo avanço das instruções no pipeline.
combinatório responsável pela geração dos sinais de controlo que permitem o encaminhamento de um res
para estágios mais avançados do pipeline.
combinatório que deteta a dependência entre instruções que se encontram em execução no pipeline e q
sinais de controlo que permitem a utilização de resultados produzidos por instruções que se encont
estágios de execução mais avançados.
baseado numa máquina de estados que deteta situações em que uma dada instrução pode avançar, num
ciclo, mais do que um estágio no pipeline.

ciclo de relógio de execução da instrução **SW** numa arquitetura MIPS *multi-cycle* é calcula
conteúdo do:
dificado no campo **RT** da instrução com o valor do *offset sign extended* para 32 bits
ficado no campo **RS** da instrução com o valor do *offset* estendido com zeros para 3
ficado no campo **RS** da instrução com o valor do *offset sign extended* para 32 bits
nter com o valor multiplicado por 4 do *offset sign extended* para 32 bits.

vírgula flutuante IEEE 754, **precisão dupla**, um expoente positivo é codificado:
superior a 1023 e igual ou inferior a 2046.
superior a 0 e inferior a 1024.
superior a 128 e inferior a 255.
com o bit mais significativo a 0.

e − 1023

$-7,87,5 \times 10^{-2} = -7,875 = -111.111 \times 2^{0} = -1,1111 \times 2^{2}$

$E = 2 + 127 = 129$

0,875	
×2	
①,750	
×2	
①,500	
×2	
①,000	

Parte Decimal

SINAL	Expoente	Mantissa (23 bits)
1	10000001	11110000000000000000000
C	0	F C