Teste 1A de Arquitetura de Computadores 08/05/2021
Duração 2h

- *Teste sem consulta e sem esclarecimento de dúvidas*
- *A detecção de fraude conduz à reprovação de todos os envolvidos*

Nº: ***Nome:***

Q1 – 1,5 valores Considere um sistema computacional em que cada posição de memória tem 12 bits. Responda às perguntas seguintes sobre o conteúdo que pode ser guardado nessa posição de memória **M**. As suas respostas não precisam de indicar um valor, podendo ser uma expressão numérica que inclui uma potência de 2.

a) Se a sequência de bits guardada na posição de memória **M** for interpretada como um número inteiro sem sinal, qual o menor e o maior número que pode ser guardado em **M**?
Menor 0
Maior $2^{12} - 1$

b) Suponha que para representar números inteiros com sinal é usada a representação em complemento para 2. Indique o valor dos 12 bits de M quando nela é armazenado -3 (base 10)? Apresente os passos que deu até chegar à resposta.
0000 0000 0011. 3
1111 1111 1100 complemento para 1
\+ 1
1111 1111 1101 complemento para 2

c) Suponha que para representar números inteiros com sinal é usada a representação em complemento para 2. Qual é o maior e o menor inteiro com sinal que pode ser armazenado em M?
Menor -2^{11}
Maior $2^{11} - 1$

Q2- 2.0 valores Considere um CPU que tem uma unidade aritmética e lógica (ALU) em que as duas entradas têm 32 bits e a saída tem 32 bits; os valores dos operandos e resultado estão em complemento para 2. A UAL tem associadas duas *flags* CF (*Carry Flag*) e OF (*Overflow Flag*).

a) A ALU acabou de fazer uma soma de dois valores U1 e U2 que são o conteúdo de duas variáveis em C declaradas como *unsigned*. A CF está a 1. O que significa significa em termos do resultado produzido pela ALU?

Se CF está a 1, isso significa que o resultado da soma não consegue ser representado em 32 bits, sendo maior do $2^{32} - 1$. Assim sendo, se considerarmos os 32 bits que saem da ALU o resultado não é correto

b) A ALU acabou de fazer uma soma de dois valores V1 e V2 que são o conteúdo de duas variáveis em C declaradas como *signed*. A OF está a 1. O que é que isto significa em termos do resultado produzido pela ALU ?

Se OF flag está a 1 é porque se verificou um de dois seguintes casos

- a soma de dois números positivos dá um resultado negativo

01.....
+01 .....
10 .....
No bit mais significativo
Carry in é 1 e Carry out é 0

- a soma de dois números negativos dá um resultado positivo

10.....
+10.....
0
No bit mais significativo
Carry in é 0 e Carry out é 1

Q3- 2.0 valores Na representação de números reais em precisão simples IEEE Floating Point Standard um número real é representado em 32 bits (o bit 31 é o mais significativo e o bit 0 é o menos significativo) com a seguinte interpretação:

- Bit 31: sinal – 0 maior ou igual a zero, 1 menor do que zero
- Bits 30 a 23 (8 bits): expoente E. Sendo E um inteiro sem sinal codificado nestes bits, o valor real do expoente é E - 127
- Bits 22 a 0 (23 bits): mantissa. Sendo a configuração dos bits da mantissa xxxxx...xxx_2, o valor efetivo da mantissa é 1.xxxx...xx_2

Considere o número real representado por *1 10000001 10000000000000000000000*. Preencha a seguinte tabela. Note que a 4ª coluna da tabela só será considerada se a 1ª, 2ª e 3ª colunas estiverem preenchidas

Sinal(+ ou -)	Expoente (base 10)	Mantissa (base 2)	Valor representado (base 10)
-	$(2^7+2^0)-127=129-127=2$	1.1_2	$1.1_2*2^2 = 110.0_2 = 6.0_{10}$

Q4- 1.5 valores Considere a seguinte sequência de instruções

```
        movl  $5, %eax
        cmpl  $6, %eax.     5 – 6 (resultado < 0 )
        jle  l1         se o resultado é <= 0 ir para l1
        movl $5, %ebx
        jmp l2
l1:     movl $6, %ebx       ebx ← 6
l2:     movl %ebx, %ecx.    ecx ← 6
```

Escreva no espaço ao lado o conteúdo dos registos eax, ebx e ecx após a execução da última instrução. Se não for possível determinar o conteúdo de um dos registos escreva ? como conteúdo do registo.

eax 5

ebx. 6

ecx. 6

Q5- 1.5 valores Considere a seguinte sequência de instruções

```
        xorl %eax, %eax     eax ← 0
        incl %eax.          eax ← 1
        movl $0x80000000, %ebx. ebx ← 0x80000000
        leal 2(%ebx, %eax, 4), %ecx
                ecx ← 2+(0x8000000 + 1 * 4)
```

Escreva no espaço ao lado o conteúdo dos registos eax, ebx, e ecx após a execução da última instrução.

eax 1

ebx. 0x80000000

ecx 0x80000006

Q6- 2.0 valores Considere a seguinte sequência de instruções

```
        movl $0xffff8000, %ebp
        movl $1, %eax.      eax ← 1
        movl $2 %ebx        ebx ← 2
        pushl %eax
        call sub1           sub1(1)
cont:   add $4, %esp
        . . .
sub1:   push %ebp
        movl %esp, %ebp
        push %ebx
         movl 8(%ebp), %eax.        eax ← 1
        movl $4, %ebx
        addl %ebx, %eax             eax ← 5
        movl -4(%ebp), %ebx         ebx ← antigo ebx (2)
        mov %ebp, %esp
        popl %ebp
        ret
```

Escreva no espaço ao lado o conteúdo dos registos eax, ebx e ebp depois da execução da instrução máquina que está na etiqueta `cont`.

eax 5

ebx. 2

ebp. 0xffff8000

Q7- 2.0 valores Suponha que se pretende construir o programa `prog` que é construído a partir de dois ficheiros fonte, um escrito em C (`prog1.c`) e outro em assembler do Pentium IA-32 (`prog2.s`). Para construir o programa `prog` deram-se os seguintes comandos no interpretador de comandos (`shell`):

```
as -o prog2.o prog2.s
gcc -o prog prog1.c prog2.o
```

Explique o que é feito por cada um dos 2 comandos anteriores.

as -o prog2.o prog2.s. traduz as mnemónicas do ficheiro fonte prog2.s para um ficheiro objeto prog2.o

gcc -o prog prog1.c prog2.o traduz o ficheiro fonte prog1.c para um ficheiro objeto prog2.o e invoca o ligador para fundir os 2 ficheiros .o, atribuindo endereços aos símbolos que estão definidos num dos ficheiros e referenciados no outro

Q8- 3.5 valores Pretende-se que construa uma função chamada *myStrlen* que poderia ser usada num programa em C com a inclusão de uma linha

```
extern int myStrlen( char str[])
```

`myStrlen` está escrita em Assembly do Intel Pentium 32 bits; recebe como parâmetro de entrada o endereço inicial *str* de uma sequência de caracteres que termina por um byte a 0. A função retorna o comprimento da cadeia ou seja o número de bytes que existem entre str e o byte a 0 que termina a cadeia. Suponha que este código está guardado num ficheiro `myStrlen.s`. Por outro lado, no ficheiro `main.c` está o seguinte código C:

```
#include <stdlib.h>
char s[] = "hello, world\n";
extern int myStrlen( char str[] );
int main() {
       int l = myStrlen( s );
       printf("%d\n", l );
       return 0;
}
```

Foi criado um ficheiro executável main com a seguinte sequência de comandos

```
as -o myStrlen.o myStrlen.s
gcc -o main main.c myStrlen.o
```

Executando o programa main este escreveu no terminal o número 13.

a) Implemente em C a função `myStrlen` sem usar funções da biblioteca.

```
int myStrlen( char str[] ) {
     int i = 0;
     while( str[i] != 0x00 ) i++;
     return i;
}
```

b) Assuma que a função `myStrlen` está implementada. Escreva o código assembly Intel Pentium IA-32 correspondente à invocação da função. Utilize as mnemónicas e convenções do *as* (**as**sembler).

```
.data
   str:  .asciiz   "hello, world\n"  # reserva de para a cadeia
                   # .asciiz significa que a cadeia é terminada por um byte a 0
.text
     ...
     pushl $str
     call myStrlen
     add $4,%esp
```

c) Escreva, em *assembly* do Pentium (versão 32 bits) e as mnemónicas e convenções do *as* (**as**sembler) o código da função *myStrlen*.

```
.text
.globl myStrlen
myStrlen:
       push %ebp
       movl %esp, %ebp
        movl 8(%ebp), %edx.          # edx ← &str[0]
       movl $0x00, %eax              # eax ← 0 ; eax guarda a variável i
ciclo: cmpb $0x00, (%edx,%eax,1).    #  str[i] – 0x00
       jz fim                        # se deu 0 acabou a cadeia
       incl %eax                     # i++
       jmp  ciclo
fim:   mov %ebp, %esp
       popl %ebp
       ret
```

As duas perguntas seguintes destinam-se a ajudar à avaliação dos trabalhos 1 e 2. Se não está a realizar a avaliação laboratorial este ano, por já ter obtido uma nota laboratorial em ano anterior, não deve responder a estas duas perguntas, mas escrever neste espaço "Não estou a realizar a parte laboratorial em 2020/21" e rubricar.

Neste caso, nota final será ajustada de acordo com a fórmula: *notaObtidaNasQuestões_1_a_7* 20/16*

Q9- 2.0 valores
Para o TPC1 foi-lhe proposto que completasse o código de um simulador (escrito na linguagem C) de uma arquitetura composta por um CPU muito simples e uma memória; esta questão é uma extensão do TPC1 e o CPU a simular inclui um novo registo, SP (um *stack pointer*) e duas novas instruções.

O código a completar está no ficheiro `dorun.c` (cuja listagem segue abaixo com as "caixas" para preencher) e deve implementar o ciclo de *fetch*, *decode* e *execute* para simular a execução das novas instruções; apenas para referência (e para ajuda, e para tornar o enunciado mais curto 😊) são apresentadas duas instruções que já existiam no TPC1, `HALT` e `LOAD`, e uma terceira, `DEC`, que simplesmente decrementa o valor em AC.

Instruction Set Architecture:

O processador usa palavras de 16 bits e tem um único registo geral (AC), um *stack pointer* (SP), e um *Program Counter* (PC), além do registo que guarda a instrução a executar (IR). A memória está também organizada em palavras de 16 bits por endereço, sendo cada endereço de 12 bits.

As instruções têm tamanho fixo de 16 bits, sendo os 4 mais significativos para o código de operação e os restantes 12 para endereço, quando necessário. As instruções suportadas e o respetivo código máquina em representação hexadecimal, são descritas a seguir:

Código	*Assembly*	*Descrição*
0x0XXX	HALT	Pára (halt) o CPU
0x1EEE	LOAD EEE	AC <- Mem[EEE]
0x2EEE	DEC	AC <- AC - 1
0x3EEE	CALL EEE	Guarda o endereço da instrução seguinte na pilha e salta para a subrotina em EEE
0x4XXX	RET	Retoma a execução no endereço guardado CALL

XXX significa que os bits são ignorados
EEE representa o endereço a indicar na instrução
Mem[EEE] representa a célula de memória de endereço EEE

Quando o CPU é ligado (*power-on*), o PC é inicializado com zero (`0x000`) e o SP com `0xFFF`, para marcar o topo da pilha (*top-of-stack*). A pilha cresce para endereços decrescentes. Em seguida apresenta-se um exemplo de um programa que decrementa um inteiro deixando o resultado no AC:

```
end.    code      assembly
0x000:   0x1010    LOAD 0x010  (inteiro a decrementar)
0x001:   0x3013    CALL 0x003
0x002:   0x0000    HALT
0x003:   0x2000    DEC
0x004:   0x4000    RET
...
0x010: 2           Inteiro (2 como exemplo)
...
0x0FF: ?           Topo do stack (valor não inicializado)
```

```
extern unsigned short int Mem[];     // A memória está definida num outro módulo C

void dorun(){
    unsigned short int pc, ir, ac, sp;
    unsigned short int opcode;
    unsigned short int address;
    pc = 0;
    while( 1 ) {
        ir =   Mem[pc];
                              ; // FETCH uma palavra (16 bits)
        opcode = (ir  & 0xf00) >> 8;       ; // opcode, 4 bits mais significativos
        address = ir &  0x0ff;              ; // endereço, 12 bits menos significativos

        switch( opcode ){              // EXECUTE
            case 0x00:   /* HALT */
                return;

            case 0x01:  /* LOAD */
                ac = Mem[address];
                pc = pc + 1;
                break;

            case 0x02: /* DEC */
                ac = ac - 1;
                pc = pc + 1;
                break;

            case 0x03: /* CALL */
                Mem[sp] = pc;
                sp = sp - 1;
                pc = address;
                break;

            case 0x04: /* RET */
                sp = sp + 1;
                pc = Mem[sp];

                break;

            default:
                printf("Invalid instruction!\n");
                return;
        }
    }
}
```

Q10 – 2.0 valores - Sobre o TPC2

Pretende-se nesta pergunta implementar uma função escrita em *Assembly* da arquitetura *Pentium IA-32,* de acordo com as convenções dadas nas aulas, que distribui os utentes de um serviço de saúde por um conjunto de regiões do país às quais pertencem (e.g. Lisboa, Porto, etc.), de modo a avaliar a distribuição demográfica desses utentes. Por simplificação, cada zona é identificada com um número (0, 1, 2, 3, etc.), e o vetor que contém informação sobre os utentes é um vetor de estruturas. Cada estrutura tem dois campos, em concreto dois inteiros -- número de utente e número da região (0, 1, 2,...).

```
#define MAXU 300000        // Número máximo permitido de utentes

#define MAXR 50            // Número de regiões consideradas


typedef struct {
   int numero_utente;      // offset de 0 bytes dentro da estrutura
   unsigned int regiao;    // offset de 4 bytes dentro da estrutura
} utente;

utente utentesServico[MAXU];           // vetor com a descrição dos utentes

unsigned int utentesRegiao[MAXR];      // vetor com o número de utentes por região
```

A função irá ser invocada a partir de um programa em C onde é declarada da seguinte forma:

```
extern void classifica( int utentes[ ], int size, int regioes [ ] );
```

O vetor `utentes` tem `size` posições (e `size` é menor ou igual a MAXU) e que `utentes[k].regiao` (com k >= 0 e k < size) contém apenas valores inteiros entre `0` e `MAXR-1.` O segundo parâmetro da função, vetor `regioes[ ]` tem `MAXR` posições e `regioes[i]` contém o número de ocorrências do valor `i` no campo `regiao` do vetor `utentes`.

O código abaixo exemplifica o uso da função `classifica`.

```
extern void classifica( int utentes[ ], int size, int regioes [ ] );

int main( ) {
   // código que preenche o vetor utentesServiço e a variável size
   // omitido
   ...
   for( int i = 0; i < MAXR; i++ )
         utentesRegiao[i] = 0;
   classifica( utentesServico, size, utentesRegiao );
  ...
   // código que processa o vetor utentesRegiao
   // omitido
   return 0;
}
```

Complete o código da subrotina `classifica` de modo a implementar a funcionalidade descrita:

```
.text
# void classifica( int utentes[ ], int size, int regioes [ ] );

MAXR = 50    # numero de regiões, i.e. dimensão do vetor regioes
SIZEUT = 8   # número de bytes ocupado por cada registo utente
OFFRG = 4    # offset do campo regiao dentro do registo de cada utente
             # o offset do campo numero_utente dentro do registo utente é zero

.globl classifica
classifica:
      push %ebp
      movl %esp, %ebp
      pushl %ebx

      movl          8(%ebp) , %edx          # endereço do vetor de utentes
      movl $0, %ecx                         # contador
      movl          16(%ebp) , %ebx         # endereço do vetor regioes
l1:
      cmpl %ecx,   12(%ebp)                  # o vetor de utentes chegou ao fim ?
      jz fim                                     # se sim, termina subrotina

      mov                4 (%edx), %eax    # consulta o numero da região

      incl    (%ebx,%eax, 4)                # incrementa regioes[%eax]

      addl               $SIZEUT ,%edx     # avança para o próximo utente

       incl %eax                                    # incrementa o contador
      jmp l1
fim:  popl %ebx
      movl %ebp, %esp
      popl %ebp
      ret
```