

Licenciatura em Engenharia Informática – 23 de Março de 2011

***Arquitectura de Computadores* 1º teste – versão B**

- Teste sem consulta.
- A interpretação do enunciado faz parte da avaliação. Em caso de dúvida sobre o enunciado, explicita nas suas respostas todas as hipóteses assumidas.
- A detecção de fraude pode levar à reprovação na cadeira de **todos** os envolvidos.

Número:

Nome:

Teóricas – Escolha múltipla: Para cada uma das perguntas seguintes indique a resposta correcta. *As respostas erradas descontam 20% da cotação.*

01- (0.7 val.) A representação de um número real em precisão simples de acordo com a norma IEEE FPS 754 em que o sinal é 1, o expoente (em excesso 127) é 10000010 e a mantissa é 110000000000000000000000, representa:

1. -0.75
2. -7.5
3. -1.4
4. -14.0

Resposta correcta:

02- (0.7 val.) Suponha que numa palavra de 32 bits o bit 31 é o mais significativo e o bit 0 o menos significativo. Para colocar o bit 5 (e só esse) do valor contido na variável x a 0, a instrução C a usar é:

1. $x = x \mid (1 \ll 5);$
2. $x = x \& (\sim 1 \gg 5);$
3. $x = x \mid (\sim 1 \ll 5);$
4. $x = x \& (\sim 1 \ll 5);$

Resposta correcta:

03- (0.7 val.) Suponha que a unidade aritmética de um CPU de 4 bits somou os dois números 0111 e 0100, produzindo um resultado com 4 bits e afectando as *flags* de *carry* (transporte) e *overflow*. Quais os valores que ficaram nessas *flags*?

1. $\text{carry} = 0$ e $\text{overflow} = 1$
2. $\text{carry} = 0$ e $\text{overflow} = 0$
3. $\text{carry} = 1$ e $\text{overflow} = 1$
4. $\text{carry} = 1$ e $\text{overflow} = 0$

Resposta correcta:

04- (0.7 val.) No *assembler* da máquina SASM

1. Para a mesma operação (por exemplo soma) a mnemónica é a mesma, quer para operandos de 8 bits quer para de 32; o assembler decide qual é a instrução máquina de acordo com a declaração das variáveis envolvidas (isto é se foram declaradas com *dd* ou *db*)
2. Para a mesma operação (por exemplo soma) a mnemónica é a mesma, quer para operandos de 8 bits quer para de 32.
3. Só existem instruções que operam a 32 bits
4. Para a mesma operação (por exemplo soma) há duas mnemónicas diferentes quando se manipulam operandos de 32 bits ou de 8 bits

Resposta correcta:

05- (0.7 val.) Para a instrução em linguagem C $a = b + c + 1$; (que se supõe que compilaria sem erros), um compilador que gerasse código para a máquina SASM, geraria

1. 3 instruções máquina
2. 1 instrução máquina
3. 4 instruções máquina
4. 2 instruções máquina

Resposta correcta:

06- (0.7 val.) Quando se pretendem somar números com e sem sinal,

1. Usa-se a mesma instrução máquina e testa-se sempre a flag *carry* (transporte) para verificar se houve *overflow*
2. Usa-se a mesma instrução máquina e testam-se flags diferentes para verificar se houve *overflow*
3. Usam-se instruções máquinas diferentes e testam-se flags diferentes para verificar se houve *overflow*
4. Usam-se instruções máquina diferentes

Resposta correcta:

07- (0.7 val.) Num CPU que usa 12 bits para representar inteiros com sinal em complemento para 2,

1. o maior valor que se consegue representar é 2047 e o menor -2048
2. o maior valor que se consegue representar é 1024 e o menor -1023
3. o maior valor que se consegue representar é 1023 e o menor -1024
4. o maior valor que se consegue representar é 4095 e o menor -4096

Resposta correcta:

08- (0.7 val.) Para o o fragmento de código C

do{

...

} while(i > 5)

o compilador gerou o seguinte código SASM:

do:

...

compare i, 5

seguido de

1. bgz do
2. bez do
3. blz do
4. br do

Resposta correcta:

Teóricas – Não de escolha múltipla: Responda à questões seguintes, procurando não ultrapassar o espaço reservado. Se não o conseguir, use a última folha

09- (1.5 val.) Considere um sistema computacional cujo organização e instruções máquina são as que correspondem às do *assembler* SASM. Descreva as acções elementares que são realizadas durante cada fase de execução da instrução *iand a, b*, e porque ordem. Para cada fase, indique os componentes que intervêm e que dados são transferidos.

10- (1.5 val) Considere o seguinte fragmento de código na linguagem SASM

```
.data
n      dd  17.0
temp   dd  ?
exp     dd  ?
```

O valor está armazenado como um real de precisão simples, na forma

$$(-1)^{\text{ sinal }} \times \text{ mantissa } \times 2^{\text{ expoente }}$$

e é usada a representação segundo a norma IEEE FPS 754, em que supondo que o bit mais significativo é o bit 31 e o bit menos significativo é o 0, se tem

Bit 31: sinal

Bits 30-23: expoente representado em excesso 127

Bits 22-0: mantissa

Escreva um fragmento de código SASM que escreve na variável **exp** o valor efectivo do expoente da variável *n*; pretende-se o valor efectivo do expoente e não a sua codificação em *excesso 127*.

```
.code
```

11- (1.5 val.) Considere os seguintes fragmentos de código C

```
int a, b, c;
```

```
...
```

```
if( (a < c) && (b == 4)){
    a = c;
    b = a;
```

```
}
```

Complete o código SASM seguinte que deve efectuar as mesmas acções que o código C apresentado.

```
.data
a dd ?
b dd ?
c dd ?
...
.code
```

```
...
```

```
end
```

Perguntas práticas (linguagem C)

12 – 2.5 val Complete o programa seguinte que interpreta o vector **v8** com tamanho **SIZE** como uma sequência de dígitos em base 8; **v8[0]** é o dígito menos significativo e **v8[SIZE-1]** é o dígito mais significativo.

O programa imprime o valor em decimal representado pela cadeia de caracteres. Isto é, se **SIZE** for 3, e **v8[0]** contiver 3, **v8[1]** 2 e **v8[2]** 4 o programa deverá imprimir $3*1 + 2*8 + 4*8*8$

```
#include <stdio.h>
#define SIZE 8
```

```
char v8[SIZE];
```

```
int main()
{
    ...
```

```
    return 0;
}
```

13 – 2.5 val As linhas marcadas com ******* do seguinte programa estão erradas. Para cada linha diga qual é o erro e escreva à frente da linha a correcção sugerida.

```
#include <stdio.h>
#include <string.h>
```

```
int main()
```

```
{
    char *s1[] = "ola"; // *** (1)
    char s2[100];
    char *s3;
```

```
    s2 = s1; // *** (2)
    s3 = s1;
```

```
    s2[0] = "a" ; // *** (3)
```

```
    if( s2 == "ala" && s3 == "ola" ) // *** (4)
        printf( "Compilou e funcionou!\n" );
```

```
    return ; // *** (5)
}
```

14- 2.5 val Diga o que é que o seguinte programa imprime para o ecrã:

```
#include <stdio.h>

int main()
{
    int v[3];

    int *p = v;

    *p = 2; p++;
    *p = 1; p++;
    *p = 0; p++;

    *(p-3) = *(v+2);
    p = v+1;

    printf("%d\n%d\n", *v, *p );

    return 0;
}
```

15- 2.5 val Pretende-se implementar a função `my_strncpy` cujo protótipo é o seguinte.

```
void my_strncpy ( char dest[], char src[], unsigned int num );
```

A função copia os primeiros *num* caracteres que começam em *src* (cadeia origem) para o endereço *dest* (cadeia destino). Se o final da cadeia origem (assinalado por um carácter 0) é encontrado antes de se terem copiado *num* caracteres, o resto da cadeia destino é preenchida com o carácter 0 até o total de *num* caracteres tenham sido copiados para a cadeia destino. Note-se que não se coloca um carácter a 0 no final da cadeia de destino, pelo que esta só será terminada se o comprimento da cadeia origem for menor do que *num*. Recapitulando

dest Apontador para o início do vector de caracteres para o qual se vai fazer a cópia

src Apontador para o início da cadeia a ser copiada

num Máximo número de caracteres a ser copiado a partir de *src*

```
void strncpy ( char * dest, char * src, unsigned int num ) {
```

```
}
```

Assembler SASM

Pseudo-Instruções

Directiva	Explicação	Exemplos
dd	Reserva de espaço para variável com 32 bits	avg dd ? i1 dd 20 f1 dd 3.14
db	Reserva de espaço para uma sequência de bytes	b1 db 100 b2 db 'A' b3 db ? str db 'hello,world', 0
.code	Seguem-se linhas com instruções	
.data	Seguem-se linhas com pseudo-instruções de reserva de espaço	
end	Fim do programa em SASM	

Algumas Instruções

O processador para o qual o SASM está definido tem duas flags (ou códigos de condição)

ZF (Zero Flag): ZF é 1 se a última operação aritmética e lógica deu resultado 0

SF (Sign Flag): SF é 1 se a última operação aritmética e lógica deu um resultado menor do que 0

Nas explicações do quadro seguinte, assuma as seguintes declarações:

int ix, iy;

unsigned char cx, cy;

float fx, fy;

Quando uma instrução tem dois operandos, o segundo pode ser o nome de uma variável ou uma constante.

Instrução SASM	Explicação (Equivalente C quando existe)
move ix,iy	ix = iy
moveb cx,cy	cx = cy
ineg ix	ix = - ix /* ix recebe o seu complemento para 2 */
iadd ix,iy	ix = ix + iy
isub ix,iy	ix = ix - iy
imult ix,iy	ix = ix * iy
idivi ix,iy	ix = ix / iy /* quociente da divisão inteira */
irem ix,iy	ix = ix % iy /* resto da divisão inteira */
fpadd fx,fy	fx = fx + fy
fpsub fx,fy	fx = fx - fy
fpmul fx,fy	fx = fx * fy
fpdiv fx,fy	fx = fx / fy
br etiqueta	PC recebe o valor representado pela etiqueta
bgz etiqueta	Salta se a última operação efectuada deu um resultado maior do que 0
bgez etiqueta	Salta se a última operação efectuada deu um resultado maior ou igual a 0
blz etiqueta	Salta se a última operação efectuada deu um resultado menor do que 0
blez etiqueta	Salta se a última operação efectuada deu um resultado menor ou igual a 0
bez etiqueta	Salta se a última operação efectuada deu um resultado igual a 0
bnz etiqueta	Salta se a última operação efectuada deu um resultado diferente de 0
compare ix,iy	Calcula ix - iy afectando as flags ZF e SF
compareb cx,cy	Calcula cx - cy afectando as flags ZF e SF
lnot ix	ix = ~ix /* negação bit a bit */
land ix,iy	ix = ix & iy /* AND bit a bit */
lor ix,iy	ix = ix iy /* OR bit a bit */
lxor ix,iy	ix = ix ^ iy /* XOR bit a bit */
llsh ix	ix << 1 /* Deslocamento lógico de ix um bit à esquerda; bit 0 recebe 0 */
rllsh ix	Deslocamento lógico de ix um bit à direita; bit 31 recebe 0
rash ix	Deslocamento aritmético de ix um bit à direita; o bit 31 mantém-se igual
rrot ix	Rotação de ix um bit à direita; o bit 31 recebe o valor que estava no bit 0
lrot ix	Rotação de ix um bit à esquerda; o bit 0 recebe o valor que estava no bit 31