Arquitectura de Computadores

Licenciatura em Engenharia Informática

Teste 2 (A) – 2010/04/30 – Duração: 1h45m

Nome:_____ Número:_____

Teste sem consulta.
A interpretação do enunciado faz parte da avaliação. Explicite nas suas respostas todas as hipóteses assumidas.
Por favor, tente ser conciso nas sua respostas para que estas caibam na zona delimitada.

Teórica - Escolha Múltipla

Indique o número da resposta que achar correcta no quadrado definido para o efeito. Cada resposta errada desconta 25% da cotação da pergunta. A cotação mínima nesta parte é 0.0 valores.

Q-1 [0.5 val.] No processo de geração de de um ficheiro executável a partir de vários ficheiros fonte assembly

1. o Assembler transforma em código máquina o código de todos os ficheiros fonte assembly, gerando o ficheiro executável
2. o Assembler gera, para cada ficheiro assembly, um ficheiro com o código máquina correspondente e o Ligador (*Linker*) liga estes ficheiros, de forma a obter o ficheiro executável
3. o Ligador (*Linker*) transforma em código máquina o código de todos os ficheiros fonte assembly, gerando o ficheiro executável
4. o Assembler resolve as etiquetas dos ficheiros fonte em endereços de memória e o Ligador (*Linker*) liga estes ficheiros assembly modificados, de forma a obter o ficheiro executável

Opção correcta ☐

Q-2 [0.5 val.] No assembly NASM/IA-32 os operandos das instruções podem estar apenas

1. num registo ou em memória
2. num registo, em memória ou fazer parte da própria instrução
3. em memória
4. em memória ou num dispositivo de entrada/saída

Opção correcta ☐

Q-3 [0.5 val.] Considere *v* um vector com 20 inteiros; cada inteiro ocupa 4 bytes. Qual das seguintes opções corresponde a somar 5 a cada elemento do vector?

Opção 1

```
        mov ebx, v
        mov esi, 0
ciclo:  add dword [ebx+esi], 5
        inc esi
        cmp esi, 20
        jl ciclo
```

Opção 2

```
        mov ebx, [v]
        mov esi, 0
ciclo:  add dword [ebx+esi*4], 5
        inc esi
        cmp esi, 20
        jl ciclo
```

Opção 3

```
        mov ebx, v
        mov esi, 0
ciclo:  add dword [ebx+esi*4], 5
        inc esi
        cmp esi, 20
        jl ciclo
```

Opção 4

```
        mov ebx, [v]
        mov esi, 0
ciclo:  add dword [ebx+esi], 5
        inc esi
        cmp esi, 20
        jl ciclo
```

Opção correcta ☐

Q-4 [0.5 val.] Considere a seguinte declaração de variável `int y;` e chamada a uma função C `x = myfunc(y, 4);`. Qual das seguintes traduções para código *assembly* respeita as convenções da transformação de código C para *assembly*?

Opção 1

```
push dword 4
push dword [y]
call myfunc
add esp, 8
mov [x], eax
```

Opção 2

```
mov eax, 4
mov ebx, [y]
call myfunc
mov [x], eax
```

Opção 3

```
push dword 4
push dword [y]
call myfunc
pop dword [x]
```

Opção 4

```
push dword [y]
push dword 4
call myfunc
add esp, 2
mov [x], eax
```

Opção correcta ☐

Q-5 [0.5 val.] Na unidade de vírgula flutuante Intel

1. Os valores em vírgula flutuante podem ser guardados nos registos EAX, EBX, ECX, EDX
2. Os registos ST0 a ST7 tanto podem guardar valores em vírgula flutuante como inteiros
3. Carregar um valor inteiro para um registo ST0 a ST7 requer uma conversão de inteiro para vírgula flutuante
4. Não se podem efectuar comparações entre dois valores

Opção correcta ☐

Q-6 [0.5 val.] Numa cache, existem dois níveis:

Nível L1 (RAM estática): Caracterizado por um tempo de acesso de t_1 nano-segundos e uma capacidade de C_1 bytes

Nível L2 (RAM dinâmica) : Caracterizado por um tempo de acesso de t_2 nano-segundos e uma capacidade de C_2 bytes

t_1 é muito menor que t_2 e C_1 é muito menor do que C_2. Se um programa P for caracterizado por boas localidades temporal e espacial o seu tempo médio de acesso à memória

1. aproxima-se de t_1, mas só se a memória necessária for de dimensão próxima de C_1
2. aproxima-se de t_1, mesmo que a memória necessária seja próxima de C_2
3. aproxima-se de t_2 se a memória necessária for próxima de C_2
4. aproxima-se da média aritmética de de t_1 e t_2

Opção correcta ☐

Q-7 [0.5 val.] As caches organizadas de forma associativa pura são pouco utilizadas porque:

1. Descobrir se o conteúdo de um dado endereço está ou não na cache é demasiado demorado
2. Podem introduzir muitas falhas (*misses*) por conflito
3. A dimensão de cada linha tem de ser inferior a 64 bytes
4. Têm pouca capacidade, porque cada linha necessita de muito hardware para gestão

Opção correcta ☐

Q-8 [0.5 val.] As caches organizadas por mapa directo são pouco utilizadas porque:

1. Descobrir se o conteúdo um dado endereço está ou não na cache é demasiado demorado
2. Podem introduzir muitas falhas (*misses*) por conflito
3. O número de linhas na cache é limitado
4. Têm pouca capacidade, porque cada linha necessita de muito hardware para gestão

Opção correcta ☐

Teórica - Não de Escolha Múltipla

Q-9 [1.5 val.] Considere que o registo AL contém o valor 1. Complete o preenchimento da tabela abaixo, indicando os valores do registo AL e das flags Carry (CF), Overflow (OF), Zero (ZF) e Sign (SF) após a execução de cada uma das instruções. Relativamente ao conteúdo do registo AL deve apresentar a representação em binário e a sua interpretação como inteiro (em decimal) com e sem sinal.

	AL (binário)	AL (decimal com sinal)	Valor AL (decimal sem signal)	CF	OF	ZF	SF
add al, 3	0000 0100	4	4	0	0	0	0
mov al, 1							
add al, −1							
add al, −1							
shl al, 1							
xor al, **al**							

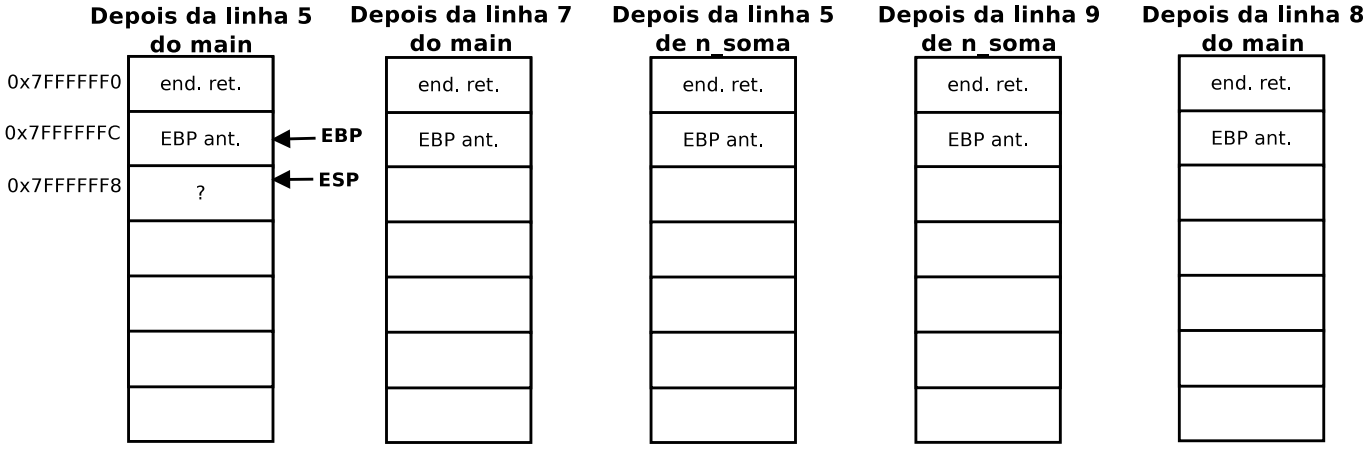
Q-10 [2.0 val.] Dada a subrotina assembly n_soma à esquerda, o programa principal C à direita e o estado da pilha de execução antes da instrução na linha 5 do programa principal, preencha o conteúdo da pilha de execução nos restantes instantes indicados. **Nota 1**: Apesar do instante preenchido apresentar os endereços dos elementos da pilha, não precisa de fornecer esta informação nos instantes seguintes. **Nota 2**: Pode referir-se ao endereço de retorno da função n_soma como *endret_n_soma* e ao conteúdo do EBP como *ebp_n_soma*.

Subrotina n_soma implementada em assembly:

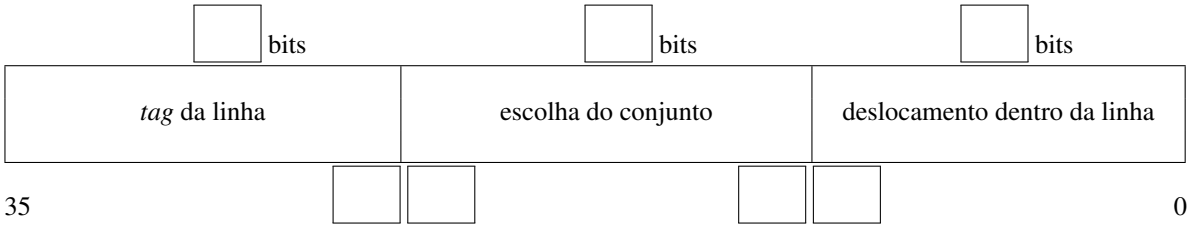
```
1  global n_soma
2  section .text
3  n_soma:
4   push    ebp
5   mov     ebp, esp
6   mov     edx, [ebp+8]
7   mov     eax, [edx]
8   add     eax, [ebp+12]
9   mov [edx], eax
10  pop     ebp
11  ret
```

Programa principal implementado em C

```
1  #include <stdio.h>
2  extern void n_soma(int *var, int val);
3
4  int main() {
5   int x;
6
7   x = 4;
8   n_soma(&x, 2);
9   printf ("O valor de x e' %d", x);
10  return 0;
11 }
```



Q-11 [1.5 val.] Considere um CPU que gera endereços com 36 bits. Entre o CPU e a RAM está uma cache com um só nível e com capacidade para 4 Mbytes. A cache tem uma organização associativa por grupos (ou conjuntos); as linhas da cache e da memória central têm 128 bytes e cada conjunto tem 8 linhas.
Preencha os quadrados em branco na figura abaixo. Na parte superior indique o número de bits usado em cada parte; na parte inferior, complete a indicação da posição dos bits inicial e final de cada uma das partes.



Prática

Nesta parte assume-se o *assembly* do Pentium com as directivas e mnemónicas do NASM.

Q-12 [3.0 val.] A seguinte subrotina pretende calcular o número de caracteres numa string cujo endereço inicial é passado como parâmetro de entrada. Algumas linhas estão erradas; para essas linhas escreva ao lado a respectiva correcção.

Etiqueta	Instrução	Eventual correcção
	global mystrlen	
section .text		
mystrlen:		
	push ebp	
	mov ebp, esp	
	mov ebx, [ebp+12]	
	mov eax, 0	
more:		
	cmp byte ebx, 0	
	je more	
	inc eax	
	inc [ebx]	
	jmp more	
done:		
	pop ebp	
	ret	

Q-13 [3.0 val.] Considere a seguinte linha de um programa em C

```
int res = (a * (-b)) + (a + b) / 2;
```

onde "/" é a divisão inteira.
Apresente o código em *assembly* correspondente. Declare a e b como palavras de 32 bits inicializadas e res como não inicializado.

```
section .data
```

```
section .bss
```

```
section .text
```

Q-14 [3.0 val.] Considere o seguinte fragmento de código escrito em C:

```
int a[5] = {1, 2, 3, 4, 5};
int b[5];
int *c = a;
int d;

b[2] = a[1];
d = *c;
```

Apresenta-se a seguir o código equivalente em *assembly*. Pretende-se que preencha as casas em que há directivas e instruções incompletas (ou seja todas as casas da tabela que não estão sombreadas).

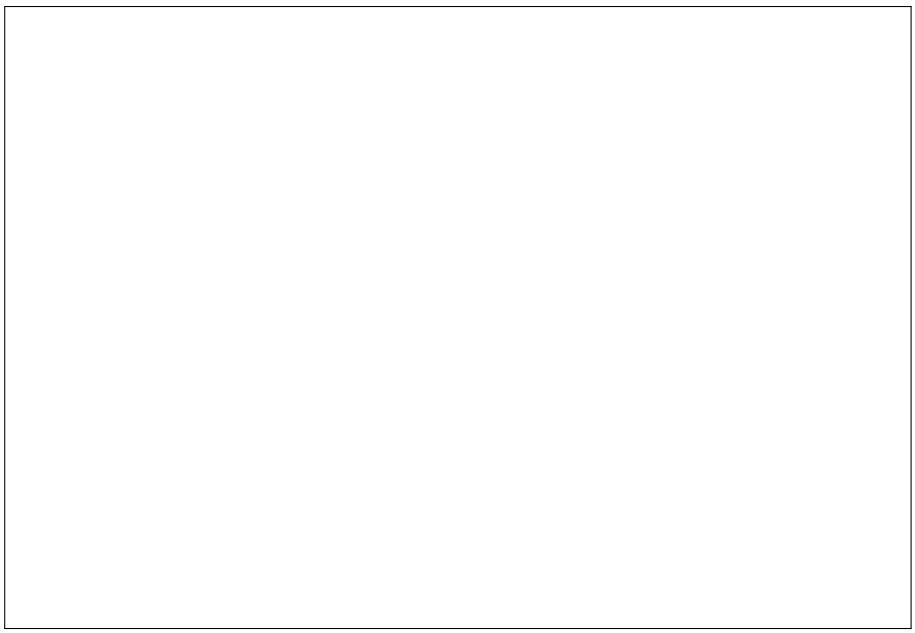
Etiqueta ou directiva	Instrução ou directiva	Comentário
section .data		
a:	dd	; int a[5] = {1, 2, 3, 4, 5};
c:	dd	; int *c = a;
section		
b:		; int b[5];
d: section .text		; int d;
	mov eax, [ ] mov [ ], eax	; b[2] = a[1]
	mov eax, [ ] mov , [ ] mov [ ], ebx	; d = *c

Q-15 [1.0 val.] Pretende-se que escreva uma subrotina em *assembly* com funcionalidade equivalente à função *fib* escrita em C.

```
int fib (int a)
{
    int f;
    int n1, n2;

    if (a == 0)
        f = 1;
    else if (a == 1)
        f = 1;
    else {
        n1 = fib (a-1);
        n2 = fib (a-2);
        f = n1 + n2;
    }
    return f;
}
```

```
section .text
fib:
```



```
ret
```

Principais instruções da IA-32 e directivas do NASM

Movimento de dados

mov *op1, op2*

Aritméticas

inc *op*
dec *op*
add *op1, op2*
sub *op1, op2*
cmp *op1, op2*
neg *op*

mul *op*
imul *op*
div *op*
idiv *op*

Lógicas e de bits

or *op1, op2*
and *op1, op2*
test *op1, op2*
xor *op1, op2*
not *op*

shl *op, n*
shr *op, n*
sal *op, n*
sar *op, n*
rol *op, n*
ror *op, n*

Saltos

jmp *label*

jz *label*
jnz *label*
jc *label*
jnc *label*
jo *label*
jno *label*
js *label*
jns *label*

je *label*
jne *label*

ja *label*
jae *label*
jb *label*
jbe *label*

jg *label*
jge *label*
jl *label*
jle *label*

Pilha

push *op*
pop *op*

Subrotinas

call *label*
ret

Várias

int *n*
nop

Directivas para reserva de espaço sem e com inicialização

resb
resw
resd
resq
db
dw
dd
dq

Qualificadores de memória

byte
word
dword
qword
tword