

Arquitectura de Computadores - 2º teste B – 4 de Maio de 2011

- Teste sem consulta; informação sobre o Pentium e o NASM está na última folha
- Sem esclarecimento de dúvidas; interpretações sobre as perguntas devem ser incluídas na resposta
- A detecção de fraude pode levar à reprovação de **todos** os envolvidos
- Em todas as perguntas assume-se que o CPU envolvido é o Pentium (versão 32 bits) e que as mnemónicas e as convenções usadas nas pseudo-instruções são as do NASM

Número:

Nome:

Parte 1: Perguntas de escolha múltipla. Respostas erradas descontam 20% da cotação; a cotação mínima nesta parte é 0.0 valores

1- 0.9 val Considere que antes da execução da instrução `add eax, ebx` `eax` contém 0x00112233 e `ebx` contém 0x11223344. Qual o conteúdo dos registos `eax` e `ebx` após a execução da instrução?

1. `eax` fica com 0x00112233 e `ebx` com 0x11335577
2. `eax` fica com 0x11335588 e `ebx` com 0x11223355
3. `eax` fica com 0x11335577 e `ebx` com 0x11223344
4. `eax` fica com 0x11225566 e `ebx` com 0x11335577

Resposta correcta:

2- 0.9 val Na instrução `add op1, op2`

1. `op1` pode ser um registo ou uma posição de memória, `op2` pode ser um registo, uma posição de memória ou uma constante; `op1` e `op2` podem ser em simultâneo posições de memória
2. `op1` pode ser um registo ou uma posição de memória, `op2` pode ser um registo, uma posição de memória ou uma constante, mas `op1` e `op2` não podem ser posições de memória em simultâneo
3. `op1` pode ser um registo ou uma posição de memória, `op2` pode ser um registo, uma posição de memória; `op1` e `op2` podem ser posições de memória em simultâneo
4. `op1` pode ser um registo, uma posição de memória ou uma constante, `op2` pode ser um registo ou uma posição de memória, mas `op1` e `op2` não podem ser posições de memória em simultâneo

Resposta correcta:

3-1.0 val Suponha que é executada a instrução `xor eax, eax`. Depois da execução desta instrução

1. ZF fica igual a 1 e SF fica igual a 0
2. ZF fica igual a 0 e SF fica igual a 1
3. Não é possível saber os valores de ZF e SF
4. ZF fica igual a 0 e SF fica igual a 0

Resposta correcta:

4- 0.9 val Suponha a seguinte linha no NASM na secção `.data`

```
n:      dd      0x11223344
```

Quando o programa está em execução a variável `n` está armazenada nas posições de memória 1000 a 1003 e não é alterada. Qual é o conteúdo da posição de memória com endereço 1000?

1. 0x44
2. 0x33
3. 0x22
4. 0x11

Resposta correcta:

5- 1.0 val Foi escrito um programa para fazer a conversão de C graus Celsius para F graus Fahrenheit de acordo com a fórmula $F = (9.0/5.0) * C + 32.0$. O programa é o seguinte

```
section .data
N5:  dd    5.0
N9:  dd    9.0
N32: dd    32.0
C:   dd    24.0
F:   dd    0
section .text
...
    fld    dword[N5]
    fld    dword[N9]
    fdiv    st1
    fld    dword[C]
    XXX    YY
    fld    dword[N32]
    fadd    st1
    fst     dword[F]
```

Onde está XXX YY deveria estar

1. `fmul st0`
2. `fadd st0`
3. `fadd st2`
4. `fmul st1`

Resposta correcta:

6- 0.9 val O registo `ebx` contém `0x1000` e o registo `esi` contém `0x0010`. Qual é o endereço de memória referenciado quando se executa a instrução `mov dword [ebx+esi*2], 0x100` ?

1. `0x1020`
2. `0x100`
3. `0x1002`
4. nenhum dos anteriores

Resposta correcta:

7- 1.0 val No Pentium o registo `esp` contém `1004` e executou-se a instrução `push dword 0x11223344`. Supondo que `mem[X]` representa o conteúdo de um byte de memória com endereço X, qual o conteúdo da memória após executar a instrução?

1. `mem[1000] = 0x11, mem[1001] = 0x22, mem[1002] = 0x33, mem[1003] = 0x44`
2. `mem[1000] = 0x44, mem[1001] = 0x33, mem[1002] = 0x22, mem[1003] = 0x11`
3. `mem[1004] = 0x11, mem[1005] = 0x22, mem[1006] = 0x33, mem[1007] = 0x44`
4. `mem[1004] = 0x44, mem[1005] = 0x33, mem[1006] = 0x22, mem[1007] = 0x11`

Resposta correcta:

8- 0.9 val O registo `ebp` serve para fazer acesso aos valores dos parâmetros de entrada de um procedimento e às variáveis locais ao procedimento, como por exemplo em `mov eax, [ebp+4]`. O valor que se soma a `ebp` chama-se deslocamento.

1. Para fazer acesso aos valores dos parâmetros de entrada usam-se deslocamentos positivos e para fazer acesso às variáveis locais usam-se deslocamentos positivos.
2. Para fazer acesso aos valores dos parâmetros de entrada usam-se deslocamentos negativos e para fazer acesso às variáveis locais usam-se deslocamentos positivos.
3. Para fazer acesso aos valores dos parâmetros de entrada usam-se deslocamento positivos e para fazer acesso às variáveis locais usam-se deslocamentos negativos
4. Nenhuma das respostas anteriores é verdadeira

Resposta correcta:

Parte 2

9- 3.0 val Considere os seguintes fragmentos de código C

```
int  x, y, z;  
...  
while(( x >= 7) && (y == 0)){  
    x = 5 * y + 3 * z;  
    y = x - 6;  
}
```

Escreva o código *assembly* correspondente; admita que o seu programa inclui as seguintes linhas

```
section .bss  
x:    resd 1  
y:    resd 1  
z:    resd 1  
...  
section .text  
...  
while:
```

```
after_while:  
...
```

10- 2.0 val Pretende-se escrever um programa em NASM equivalente ao seguinte código C

```
int vec[20], i;
int size = 20;
...
for( i = 0; i < size; i++)  vec[i] = i;
```

Assinale as linhas que estão erradas no troço de código apresentado a seguir; indique à frente à sua proposta de correcção

```
section .data
SIZE: dd    20
section .bss
vec:  resd 20
section .text
...
    mov     ebx, [vec]
    mov     ecx, SIZE
    mov     eax, 0      ; eax vai armazenar o valor de i
l1:  mov     [ebx], eax
    inc     ebx
    inc     eax
    loop    l1
...
```

11- 2.5 val O produto interno de dois vectores X e Y com 100 elementos pode ser calculado pelo seguinte código C

```
int i, p, X[100], Y[100];
...
p = 0;
for( i = 0; i < 100; i++ )
    p = p + Y[i]*X[i];
```

Admitindo que o seu programa inclui as seguintes linhas

```
section .bss
X:      resd 100
Y:      resd 100
P:      resd 1
```

Supondo que nenhum dos valores intermédios excede $2^{31}-1$, indique os operandos das duas instruções mov que estão incompletas.

```
section .text
...
    mov     esi, X
    mov     edi, Y
    mov     edx, 0; edx vai armazenar o valor de p
    mov     ecx, 0; ecx vai armazenar o valor de i

again:  mov

    mov

    push    edx
    imul    ebx
    pop     edx
    add     edx, eax
    inc     ecx
    cmp     ecx, 100
    jb      again
    mov     [p], edx
```

12- 2.5 val Considere que pretende transformar o código da pergunta 11 numa subrotina que pode ser invocada de um programa em C:

```
int m1[100], m2[100], z
int innerProd100( int *x, int *y); /* protótipo da função
                                   em assembler */

...
z = innerProd100( m1, m2);
```

Para fazer essa alteração, é preciso modificar o código da pergunta 11.

```
global innerProd100
section .text
innerProd100:
    push ebp
    mov  ebp, esp

    _____ ; instrução que substitui mov esi,X

    _____ ; instrução que substitui mov edi,Y
    mov  edx, 0
    mov  ecx, 0
again:
    ...          ; código igual ao da pergunta 11
    jnb  again

    _____ ; instrução que substitui mov [p], edx
    mov  ebp, esp
    pop  ebp
    ret
```

Preencha as instruções em falta.

13- 2.5 val Considere que já existe o código da rotina em assembler `innerProd100` e que se escreveu um programa principal em C que chama a rotina `innerProd100`

```
int m1[100], m2[100];
int innerProd100( int *x, int *y); /* protótipo da função
                                   em assembler */

...
int main(){
    printf("z = %d\n", innerProd100( m2, m1));
    return 0;
}
```

Supondo que os vectores `m1` e `m2` correspondem a

```
section .bss
m1:  resd 100
m2:  resd 100
```

Complete o código abaixo que corresponde à invocação de `innerProd100`

```
push  _____

push  _____
call  innerProd100

_____
```

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`

Unidade de Reais

`fld op`

`fild op`

`fldz`

`fldl`

`fldpi`

`fst op`

`fxch op`

`fadd op`

`fsub op`

`fsub op`

`fmul op`

`fdiv op`

`fchs`

`fabs`

`fsin`

`fcos`

`fptan`

`fsqrt`

Directivas nasm

`db`

`dw`

`dd`

`dq`

`dt`

Qualificadores de memória

`byte`

`word`

`dword`

`qword`

Convenções de chamada de funções do C

Invocador

- **Empilha parâmetros:** da direita para a esquerda
- **call** invocado
- **recupera espaço** ocupado pelos parâmetros

Invocado

- **Começa por** `push ebp`
`mov ebp, esp`
- **Acaba com** `mov esp, ebp`
`pop ebp`
`ret`
- **Retorna valores** `eax` se inteiro ou `st0` se real