Arquitectura de Computadores

Licenciatura em Engenharia Informática

1º teste (A) – 2006/05/07 – Duração: 2h00m

Nome:_____

Total de páginas: 5+__ *páginas* Classificação:_____

Q-1 Considere uma máquina máquina Intel que utiliza o ISA que indicou na questão anterior.

a) [0.75 val.] Apesar desta máquina operar a 32 bits, pode-se aceder a registos de tamanho inferior. Estes são na realidade subconjuntos dos registos de carácter geral, como o EAX. Diga o porquê da necessidade desta arquitectura ter registos de tamanhos inferiores a 32 bits?

Compatibilidade com modelos anteriores que operavam a 8 ou 16 bits

b) [0.25 val.] No caso particular do registo EAX que nomes têm esses registos e qual é o seu respectivo tamanho (em bits)?

AX (16 bits), AL (8 bits) e AH (8 bits)

c) [0.75 val.] Nesta máquina a memória é endereçada ao byte, a 16 bits ou a 32 bits? O que é que isso significa?

Endereçada ao byte. Isto significa que o tamanho de um célula de memória é de 1 byte. Por outras palavras, a cada byte em memória tem o seu endereço e logo pode ser referenciado.

d) [0.25 val.] Que convenção é utilizada para ordenar os bytes dentro de uma palavra (*word*) de memória?

Little-endian

e) [0.25 val.] Coloque a seguinte palavra de 32 bits (FF FE C3 0F) no endereço 1001 da memória abaixo. Escreva o novo valor na coluna *Conteúdo Actualizado*.

Endereço	Conteúdo	Conteúdo Actualizado
00000FFC:	2F	_____
00000FFD:	10	_____
00000FFE:	04	_____
00000FFF:	FF	_____
00001000:	FC	_____
00001001:	00	0F
00001002:	01	C3
00001003:	A3	FE
00001004:	01	FF

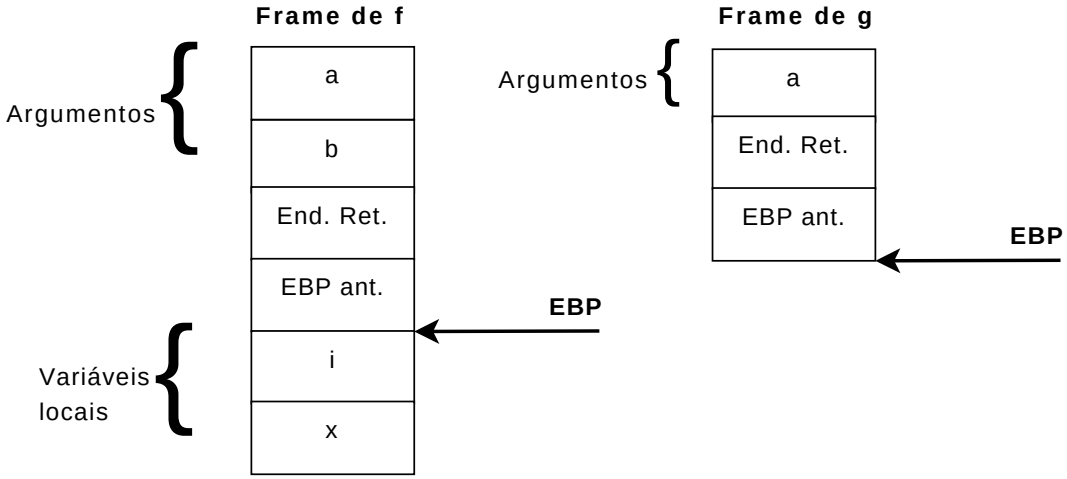
f) [0.75 val.] Considerando a mesma zona de memória, diga qual é o resultado das seguintes operações. *Considere que todas as posições de memória não indicadas na tabela acima contêm o valor zero.*

`mov eax, [1000H]`	eax = FE C3 0F FC	Lembre-se que o H denota que o número está em hexadecimal.
`mov eax, 1000`	eax = 1000	
`mov eax, ebx`	eax = 0FFCH	O conteúdo de ebx é OFFCH.
`mov eax, [ebx]`	eax = FF 04 10 2F	O conteúdo de ebx é OFFCH.
`mov eax, {1000H}`	eax = 0	Suponha que a notação { } denota endereçamento indirecto por memória. Algo que não existe neste assembly.

Q-2 Considere o seguinte fragmento de um programa em C.

```
int f(int a, int b) {
  int i, x = 0;
  for (i = 0; i < 10; i++) {
    x = x + g(a-b)
    b--;
  }
  return x;
}
int g(int a) {
  return a*a - a;
}
int main() {
 printf("O valor e' %d", f(20,10));
  return 0;
}
```

a) [1.25 val.] Faça um esquema das frames de activação das funções f e g.



b) [0.50 val.] Indique nos esquemas acima para onde deve apontar o registo EBP. Qual é a utilidade deste registo?

A utilidade do EBP é servir como base para os argumentos e para as variáveis locais de uma função. A motivação de se usar o EBP em vez do ESP é que é o evitar que as operações sobre a pilha alterem essa base. Portanto, a separação das funcionalidades permite o uso arbitrário da pilha sem que tal influa no endereçamento dos valores da frame de activação da função.

Q-3 Considere o seguinte programa em NASM/Intel (IA-32):

```
mov eax, 1000
add eax,  1000
mul dword [1000]
```

e o seguinte estado: `Overflow Flag (OF)`: 1 `Carry Flag (CF)`: 0 `Zero Flag (ZF)`: 1

a) [0.5 val.] Qual é o valor das flags após a primeira instrução. `OF`: 1 `CF`: 0 `ZF`: 1 (não muda nada)

b) [0.5 val.] Qual é o valor das flags após a segunda instrução. `OF`: 0 `CF`: 0 `ZF`: 0

c) [0.5 val.] Explique o porquê de, na terceira instrução, se ter de anotar a quantidade de informação a ler de memória, quando não necessário fazê-lo nas duas primeiras.

A terceira instrução lê dados da memória, usando endereçamento directo. Em instruções como `add eax, [1000]` o registo (neste caso o eax) serve como medida implicita para a quantidade de informação a ler. Quando não temos usa medida, temos de explicitamente anotar essa quantidade de informação.

Q-4 [0.75 val.] Escreva o código NASM/Intel (IA-32) que efectua a chamada à função `raiz` ilustrada no seguinte código C: `x = raiz(5.0)`. O protótipo da função é: `float raiz(float x)`.

```
arg dd 5.0 ; este declaração era necessária porque não existe endereçamento imediato para números em FP
...
push dword [arg]
call raiz
fstp [x]
add esp, 4
```

Q-5 Implemente um programa em assembly NASM/Intel (IA-32) que calcula a função de Fibonacci do valor colocado na posição de memória com etiqueta *N*. O resultado da função deve ser colocado na posição com etiqueta *fibN*. **As soluções das alíneas devem ser escritas no espaço reservado da página seguinte.**

a) [1.0 val.] Declarar a variável *N*, com tamanho de 32bits e com valor inicial 10 e a variável *fibN*, com tamanho de 32 bits e sem valor inicial.

b) [de 1.50 a 3.00 val.] Implementação do cálculo da função. **Implemente apenas uma das opções, indique a sua escolha com uma cruz.**

- **[1.50 val.]** Implementação iterativa no programa principal, sem recorrer a uma subrotina.
- **[2.00 val.]** Implementação iterativa usando uma subrotina. Deve implementar o programa principal que chama a rotina, passando o argumento *N* por pilha, e guardar o resultado, retornado no registo EAX, em *fibN*.

  O algoritmo iterativo é o seguinte:

```
int fib(int n) {
  int pen = 0, ult = 1, res = 1, i;
  for (i = 1; i < n; i++) {
    res = pen + ult;
    pen = ult;
    ult = res;
  }
  return res;
}
```

- **[2.50 val.]** Implementação recursiva, recorrendo ao ESP como base para os argumentos. Deve implementar o programa principal que chama a rotina, passando o argumento *N* por pilha, e guardar o resultado, retornado no registo EAX, em *fibN*.
- **[3.00 val.]** Implementação recursiva, recorrendo ao EBP como base para os argumentos. Deve implementar o programa principal que chama a rotina, passando o argumento *N* por pilha, e guardar o resultado, retornado no registo EAX, em *fibN*.

  O algoritmo recursivo é o seguinte:

```
int fib(int n) {
  if (n <= 1) return 1;
  else return fib(n-1) + fib(n-2);
}
```

```
SYS_EXIT equ 1
LINUX_SYSCALL equ 80H
global _start
section .data
N:      dd     10
section .bss
fibN:   resd   1
section .text
_start:
   push dword [N]
   call fib
   mov [fibN], eax
   add esp, 4
   mov ax, SYS_EXIT
   mov bx, 0
   int LINUX_SYSCALL

fib:
   push ebp
   mov ebp, esp

   mov   ebx, [ebp+8]
   cmp   ebx, 1        ; comparar com 1
   jle   one           ; saltar se <= 1
   dec   ebx
   push  ebx
   call  fib           ; fib(n-1)
   add   esp, 4        ; repor a pilha
   push  eax           ; guardar resultado na pilha
   mov   ebx, [ebp+8]
   sub   ebx, 2
   push  ebx
   call  fib           ; fib(n-2)
   add   esp, 4        ; repor a pilha
   pop   ebx           ; buscar o resultado previamente guardado
   add   eax, ebx      ; eax = fib(n-2) + fib(n-1)
   jmp   endfib        ; saltar para o fim

one:
   mov   eax, 1
endfib:
   pop   ebp
   ret                   ; podia se colocar aqui ret 4, mas nesse caso tiravamos os add esp, 4
```

Q-6 Suponha que tem à sua disposição as seguintes funções que operam sobre strings:
`int stringLength(char* src)`: Calcula o tamanho da string dada.
`char* stringInvert(char* src)`: Retorna uma string cujo conteúdo é a inversão da string dada.
`void stringInvert2(char* dest, char* src)`: Faz o mesmo processamento que a função anterior, mas a string a conter o resultado é passada por referência. **Nota:** a função pressupõe que a memória para a string destino foi reservada antes da chamada.

a) [1.00 val.] Implemente a função `main` de um programa em C que dada uma string em `argv[1]` calcula e imprime o seu tamanho e a sua string inversa. Deve usar as funções `stringLength` e `stringInvert` acima definidas. **Nota:** o programa só recebe o argumento que contem a string a ser processada.

```
int main(int argc, char* argv[]) {
  if (argc != 2)
    printf("Numero de argumentos errado");
  printf("Tamanho: %d String inversa: %s\n", stringLength(argv[1]), stringInvert(argv[1]);
  return 0;
}
```

b) [de 1.50 a 3.00 val.] Implementação da rotina para o cálculo da função. **Implemente apenas uma das opções, indique a sua escolha com uma cruz.**

- **[1.50 val.]** Implemente a função `stringLength`.
- **[2.00 val.]** Implemente a função `stringInvert2`. **Não se esqueça:** a função pressupõe que a memória para a string destino foi reservada antes da chamada.
- **[2.50 val.]** Implemente a função `stringInvert`. **Não pode usar variáveis globais.**
- **[3.00 val.]** Implemente a função `stringInvert` usando apontadores para percorrer as strings. **Não pode usar variáveis globais.**

```
int stringLength(char *str) {
  int i = 0;

  while (str[i] != '\0')
    i++;
  return i;
}

void stringInvert2(char *dst, char *src) {
  int len = stringLength(src);
  int i;

  dst[len] = '\0';
  for (i = 0; i < len; i++)
    dst[len-1-i] = src[i];
}

char *stringInvert(char *str) {
  int len = sizeof(char) * stringLength(str);
  char *result = (char *) malloc (len + 1);
  char *p = str, *q = result + len;

  *q = '\0';
  q--;
  for (; *p != '\0'; p++, q--)
    *q = *p;
  return result;
}

/* Versao condensada */
char *stringInvertCond(char *str) {
  int len = sizeof(char) * stringLength(str);
  char *result = (char *) malloc (len + 1);
  char *p = str, *q = result + len;

  *q = '\0';
  while (*p != '\0')
    *(--q) = *(p++);
  return result;
}
```