Sistemas Operacionais – Prof. Rafael R. Obelheiro

Exercícios — Fundamentos de SO — Respostas

Processador hipotético

1. Sem resposta

2. O programa exibe a sequência de Fibonacci:

$$1, 1, 2, 3, 5, 8, 13, 21, \ldots,$$

onde cada termo é a soma de seus dois antecessores:

$$F_n = F_{n-1} + F_{n-2}$$
$$F_1 = 1 \qquad F_2 = 1$$

(Estritamente falando, a sequência de Fibonacci inicia com $F_0 = 0$, mas o programa começa com F_1. Fica como exercício modificá-lo para iniciar com F_0.)

3. O algoritmo implementado pelo programa é o seguinte:

```
read(n);
do {
   print(n);
   if (n != 0)
      n--;
} while (n != 0);
```

A posição de memória 600 armazena a constante 1, para poder decrementar o acumulador, e a posição de memória 601 armazena a variável `n`.

Programa:

```
100: 0001    ; AC = 1
101: 2600    ; [600] = AC (1)
102: 7000    ; AC = teclado
103: 2601    ; [601] = AC (N)
104: 8001    ; video = AC
105: 6109    ; JZ 109
106: 4600    ; AC -= [600]
107: 2601    ; [601] = AC
108: 5104    ; JMP 104
109: 9999    ; HALT
```

4. O algoritmo implementado pelo programa é o seguinte:

```
read(n1);
n2 = 0;
do {
   print(n2);
   n2++;
   n1--;
} while (n1 != 0);
print(n2);
```

A memória é usada de acordo com a seguinte tabela:

Endereço	Variável
700	1 (const)
701	n1
702	n2

Programa:

```
100: 0001    ; AC = 1
101: 2700    ; [700] = AC (1)
102: 7000    ; AC = teclado
103: 2701    ; [701] = AC (N1)
104: 0000    ; AC = 0
105: 2702    ; [702] = AC (N2)
106: 8001    ; video = AC
107: 3700    ; AC += [700]
108: 2702    ; [702] = AC
109: 1701    ; AC = [701]
110: 4700    ; AC -= [700]
111: 2701    ; [701] = AC
112: 6115    ; JZ 115
113: 1702    ; AC = [702]
114: 5106    ; JMP 106
115: 1702    ; AC = [702]
116: 8001    ; video = AC
117: 9999    ; HALT
```

5. O algoritmo implementado pelo programa é o seguinte:

```
1 read(n1);
2 read(n2);
3 n3 = 0;
4 while (n2 != 0) {
5     n3 += n1;
6     n2--;
7 }
8 print(n3);
```

A memória é usada de acordo com a seguinte tabela:

Endereço	Variável
500	1 (const)
501	n1
502	n2
503	n3

Programa:

```
100: 0001    ; AC = 1
101: 2500    ; [500] = AC (1)
102: 7000    ; AC = teclado
103: 2501    ; [501] = AC (N1)
104: 7000    ; AC = teclado
105: 2502    ; [502] = AC (N2)
106: 0000    ; AC = 0
107: 2503    ; [503] = AC (N1*N2)
108: 1502    ; AC = [502]
109: 6117    ; JZ 117
110: 1503    ; AC = [503]
111: 3501    ; AC += [501]
112: 2503    ; [503] = AC
113: 1502    ; AC = [502]
114: 4500    ; AC -= [500]
115: 2502    ; [502] = AC
116: 5109    ; JMP 109
117: 1503    ; AC = [503]
118: 8001    ; video = AC
119: 9999    ; HALT
```

Programação Assembly x86

6. (a) EDX ← 40
   (b) EAX ← 20
   (c) EAX ← 1020 (conteúdo de *mem[EBX]*)
   (d) EBX ← 50
7. AX ← 256 (1 << 8, ou 1 deslocado 8 bits para a esquerda)
8. A instrução `shl` é executada ECX=10 vezes. Portanto, o resultado é que EAX fica com o valor 1024 (1 << 10).
9. O trecho é equivalente a `x--` em C.
10. O efeito de cada linha do código Assembly é o seguinte:

```
EAX <- 250
EAX <- 260
x <- 4 (260%256)
y <- 1 (260/256)
```

11. O efeito de cada linha do código Assembly é o seguinte:

```
AX <- 532 (AH=532/256=2; AL=532%256=20)
AH <- 12
x <- 3092 (AX=256*AH+AL=256*12+20=3092)
```

12. O efeito de cada linha do código Assembly é o seguinte:

```
AX <- 255 (AH=0; AL=255)
AL <- 0 (AH não é alterado)
x <- 0
y <- 0
```

13. `x <- max(x, 10)`
14. `x <- min(x, 10)`
15. (a)

```
 mov x, %eax
 imul 3, %eax, %eax
 add %eax, 1        # ou inc %eax
 mov %eax, y
```

    (b)

```
 mov b, %ebx        # A) calcula b*b e armazena em EBX
 mul %ebx, %ebx     # A)
 mov a, %eax        # B) calcula a*c e armazena em EAX
 mov c, %ecx        # B)
 mul %ecx, %eax     # B)
 shl $2, %eax       # C) multiplica EAX por 4 (EAX=4*a*c)
 sub %eax, %ebx     # D) EBX = b*b - 4*a*c
 mov %ebx, delta    # E) delta = EBX
```

    (c)

```
 mov x, %eax
 cmp y, %eax
 jg  atribui        # usar JA se 'x' e 'y' fossem inteiros sem sinal
 mov y, %eax
atribui:
 mov %eax, max
 ...                # sequência do código
```

(d)

```
  mov a, %eax
  cmp b, %eax
  jge maiorigual     # usar JAE se 'a' e 'b' fossem inteiros sem sinal
  mov $b, ptr
  jmp fimse
maiorigual:
  mov $a, ptr
fimse:
  ...                # sequência do código
```

(e)

```
  mov x, %ecx
  mov y, %eax
inicio:
  shl $1, %eax       # multiplica EAX por 2
  loop inicio        # decrementa ECX e volta para 'inicio' se ECX != 0
  mov %eax, y
```

16. Sem resposta

17. Ver arquivo `ex17-cmp51.s`.

18. Ver arquivo `ex18-cmp-2num.s`.

19. Ver arquivo `ex19-media-2num.s`.

20. Ver arquivo `ex20-dif-2num.s`.

21. Ver arquivos `ex21a-pot-mul.s` e `ex21b-pot-soma.s`.

22. (a) A pilha fica com a seguinte configuração:

EBP+16	c	
EBP+12	b	
EBP+8	a	
EBP+4	end. retorno	
EBP	EBP salvo	
EBP−4	i	
EBP−8	j	ESP

(b) A pilha fica com a configuração abaixo, e o registrador ESI contém 250 (último valor empilhado antes de `pop %esi`):

EBP+16	c	
EBP+12	b	
EBP+8	a	
EBP+4	end. retorno	
EBP	EBP salvo	
EBP−4	i	
EBP−8	j	
EBP−12	200	
EBP−16	300	ESP

23. Ver arquivo `ex23-print3.s`.
24. Ver arquivo `ex24-strlen.s`.
25. Ver arquivo `ex25-media-4.s`.
26. Ver arquivo `ex26-media-N.s`.