sop-exerc-fundam.pdf

<https://moodle.joinville.udesc.br/pluginfile.php/2717/mod_resource/content/12/sop-exerc-fundam.pdf>

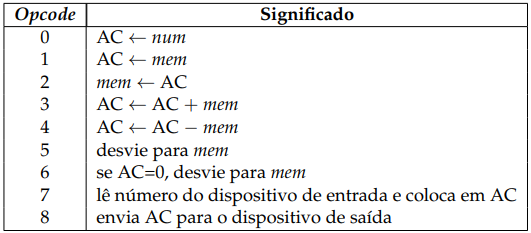
Sistemas Operacionais - Prof. Rafael R. Obelheiro

Exercícios - Fundamentos de SO

<http://www.jegerlehner.ch/intel/IntelCodeTable.pdf>

**Processador hipotético**

**1. [Stallings 1.1mod]** Considere um processador hipotético, semelhante ao usado no exemplo do slide 36, que possua os seguintes opcodes:



O operando de cada instrução e um número num (para o opcode 0), o endereço de memória mem (para os opcodes 1–6) ou o número do dispositivo de E/S (para os ´ opcodes 7 e 8), que pode ser 0 para o teclado, 1 para o v´ıdeo ou 2 para a interface de rede.

Seguindo o formato ilustrado no slide 36, mostre a execução do seguinte programa:

1: Leia um número do teclado e o coloque no acumulador (AC);

2: Adicione o conteúdo da posição de memória 940;

3: Imprima o conteúdo do acumulador no vídeo.

Suponha que o número lido seja 3 e que a posição 940 contenha o valor 2.

**2. Usando o processador do exercício 1, mostre a execução do programa abaixo, e explique o que ele faz.**

Os números à esquerda do sinal de dois pontos são endereços de memória, e os números a direita o conteúdo de cada endereço.

Suponha que o valor inicial dos registradores da CPU sejam PC=100, AC=1234 e IR=9876. (PC e o contador de programa, AC e o acumulador e IR ´ e o registrador de instrução.)

100: 0001

101: 2400

102: 8001

103: 2401

104: 8001

105: 3400

106: 2402

107: 1401

108: 2400

109: 1402

110: 2401

111: 5104

Resposta: O programa exibe a sequência de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, … onde cada termo é a soma de seus dois antecessores: Fn = Fn−1 + Fn−2, F1 = 1 F2 = 1.

**3. Usando o processador do exercício 1, escreva um programa que leia um número do teclado e imprima uma contagem regressiva, do número lido (inclusive) até zero. Use a instrução 9999 para indicar o fim do programa.**

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. Usando o processador do exercício 1, escreva um programa que leia um número do teclado e imprima uma contagem progressiva, de zero até o número lido (inclusive).**

read(n1);

n2 = 0;

do {

print(n2);

n2++;

n1--;

} while (n1 != 0);

print(n2);

**5. Usando o processador do exercício 1, escreva um programa que leia dois números do teclado e imprima o produto entre eles.**

read(n1);

read(n2);

n3 = 0;

while (n2 != 0) {

n3 += n1;

n2--;

}

print(n3);

**Programação Assembly 8088**

**6. Suponha que, em um processador 8088, os registradores tenham atualmente os valores CS=2000, DS=100, IP=200, AX=10, BX=20, CX=30.**

**(a) Determine o endereço absoluto das duas próximas instruções a serem executadas, sabendo que elas tem 4 e 6 bytes de comprimento, respectivamente.**

Endereço da próxima instrução é dado pela combinação de CS:IP

CS - 16 bits IP - 16 bits

juntando os dois, temos: 20 bits

Primeira instrução:

Endereço Físico (absoluto): (CS<<4) + IP = (CS\*16)+IP = (2000\*16)+200= 32 200

//2^4= 16

//CS=2000 de acordo com o problema

Arquitetura 8088 as instruções não têm todas o mesmo tamanho.

Segunda instrução: IP=204

Como a 1a instrução tem 4 bytes, a 2a instrução estará no endereço absoluto 32.204

Endereço físico (absoluto) = (16\*2000) + 204= 32 204

//Terceira instrução: IP=210

**(b) Determine o efeito produzido pela execução das seguintes instruções:**

**Considere que o conteúdo dos endereços absolutos de memória de 0 a 1999 seja a sequência de valores [1000, 2999], ou seja, mem[addr] = addr+1000 (para addr ∈ [0, 1999]).**

**I1) MOV DX,40**

DX <- 40

**I2) MOV AX,BX**

AX <- 20

**I3) MOV AX,(BX)**

//Tratar BX como se fosse um ponteiro para dados

AX <- \* (DS:BX)

DS=100, BX=20

DS:BX= (16\*100)+20=1620

AX=2620

**I4) ADD BX,CX**

BX <- BX+CX =50

RESPOSTA COMPLETA:

I1) DX ← 40

I2) AX ← 20

I3) AX ← 2620 (conteúdo de mem[DS:BX] = mem[(16 × 100) + 20] = mem[1620])

I4) BX ← 50

**7. Determine o efeito produzido pela execução do seguinte trecho de código Assembly:**

1 MOV AX,1 //transfere 16 bits

2 MOVB CL,8 //transfere 8 bits (B=byte)

3 SHL AX,CL //desloca os bits para esquerda um determinado número de vezes, nesse caso 8 vezes.

AX <- 1

CL <- 8

AX <- (AX << CL) = (1<<8) = 256 (2^8, 2 por causa do binário)

//1 deslocado 8 bits para a esquerda

RESPOSTA:

AX ← 256 (1 << 8, ou 1 deslocado 8 bits para a esquerda)

**8. Determine o efeito produzido pela execução do seguinte trecho de código Assembly:**

A instrução SHL é executada CX=10 vezes. Portanto, o resultado é que AX fica com o valor 1024 (1 << 10).

MOV: transfere 16 bits

1 MOV CX,10

2 MOV AX,1

3 inicio:

4 SHL AX,1 //deslocam os bits de dst para a esquerda um determinado número de unidades

5 LOOP inicio

AX <- (1<<CX)=1 <<10 = 1024 (ou seja 2^10)

RESPOSTA:

A instrução SHL é executada CX=10 vezes. Portanto, o resultado é que AX fica com o valor 1024 (1 << 10)

**9. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x e um rótulo na seção de dados):**

1 MOV AX,(x)

2 DEC AX //decrementa: Exemplo: dec reg-mem reg-mem ← reg-mem − 1

3 MOV (x),AX

AX <- x

AX -- x--

x <- AX

MOV CX, x => CX <- 344

CX <- x

Rótulos são endereços e não variáveis.

Rótulos equivalem a ponteiros.

p: word 345 => int \*p;

\*p= 345;

MOV AX, p => AX=p; /\*endereço \*/

MOV AX, (p) => AX=\*p; /\*conteúdo \*/

RESPOSTA:

O trecho é equivalente a x-- em C.

**10. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x e y são rótulos na seção de dados):**

1 MOV AX,250

2 ADD AX,10

3 MOVB (x),AL //movb transfere 8 bits

4 MOVB (y),AH

AX <- 250

AX <- 10 + 250 = 260

x <- 4 (260%256)

y <- 1 (260/256)

**11. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x é um rótulo na seção de dados):**

1 MOV AX,532

2 ADDB AH,10

3 MOV (x),AX

AX <- 532 (AH=532/256=2; AL=532%256=20)

AH <- 12

x <- 3092 (AX=256\*AH+AL=256\*12+20=3092)

**12. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x é um rótulo na seção de dados):**

1 MOV AX,255

2 INCB AL

3 MOVB (x),AL

4 MOVB (y),AH

RESPOSTA:

AX <- 255 (AH=0; AL=255)

AL <- 0 (AH nÃo É alterado)

x <- 0

y <- 0

**13. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x e y são rótulos na seção de dados):**

1 MOV AX,25

2 MOV BX,40

3 IMUL BX

4 MOV (x),AX

5 MOV (y),DX

RESPOSTA:

AX <- 25

BX <- 40

DX:AX <- 1000 (DX=0, AX=1000)

x <- 1000 5 y <- 0

**14. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x e y são rótulos na seção de dados):**

1 MOV AX,2000

2 MOV BX,34

3 IMUL BX

4 MOV (x),AX

5 MOV (y),DX

RESPOSTA:

AX <- 2000

BX <- 34

DX:AX <- 68000 (DX=68000/65536=1, AX=68000%65536=2464)

x <- 2464 5 y <- 1

**15. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x é um rótulo na seção de dados):**

1 MOV AX,10 AX <- 10

2 MOV BX,(x) BX <- x

3 CMP AX,BX

4 JB 1f Se (AX >= BX) então x<- AX //isso é para 3, 4 e 5

5 MOV (x),AX

6 1: …

1f: 1 para frente

1p: 1 para trás

RESPOSTA:

x <- max(x, 10)

**16. Determine o efeito produzido pela execução do seguinte trecho de código Assembly (x é um rótulo na seção de dados):**

1 MOV AX,10

2 MOV BX,(x)

3 CMP AX,BX //Compara dois operandos

4 JA 1f

5 MOV (x),AX

6 1: ...

se (x>=10)

x<-10 x<-min(x,10)

RESPOSTA:

x <- min(x, 10)

**17. Escreva trechos de código Assembly que implementem o trechos de código C abaixo. Considere que as variáveis em C são inteiros com sinal de 2 bytes (a exceção de ptr, que e um ponteiro, também de 2 bytes), e que existem no programa Assembly rótulos de mesmo nome na seção de dados para representá-las.**

**(a) y = 3\*x + 1;**

MOV AX,(x)

MOV CX,3

MUL CX

ADD AX,1 ! ou INC AX

MOV (y),AX

**(b) delta = b\*b - 4\*a\*c;**

MOV BX,(b) ! calcula b\*b e armazena em BX

MOV AX,BX   
MUL BX   
MOV BX,AX

MOV AX,(a) ! calcula a\*c e armazena em AX

MOV CX,(c)

MUL CX

SHL AX,1 ! multiplica AX por 4 (AX=4\*a\*c)

SHL AX,1

SUB BX,AX ! BX = b\*b - 4\*a\*c

MOV (delta),BX

**(c) if (x > y) max = x; else max = y;**

MOV AX,(x)

MOV BX,(y)   
CMP AX,BX

JG maior ! usar JA se ’x’ e ’y’ fossem inteiros sem sinal

MOV (max),BX

JMP 1f

maior:

MOV (max),AX

1: ... ! sequência do código

**(d) if (a >= b) ptr = &a; else ptr = &b; \*ptr++;**

MOV AX,(a)

MOV BX,(b)

CMP AX,BX

JGE maiorigual ! usar JAE se ’a’ e ’b’ fossem inteiros sem sinal

MOV (ptr),b

JMP 1f

maiorigual:

MOV (ptr),a

1: MOV AX,(ptr)   
INC AX

MOV (ptr),AX

... ! sequˆencia do c´odigo

**(e) while (x > 0) { y = y\*2; x--; }**

MOV CX,(x)   
MOV AX,(y)   
inicio:

SHL AX,1 ! multiplica AX por 2

MOV (y),AX ! y <- AX

LOOP inicio ! decrementa CX e volta para ’inicio’ se CX != 0

**18. Modifique o programa exemplo HelloWrld.s para que ele imprima uma mensagem diferente de “Hello, World!”. Monte o programa e teste seu funcionamento.**

Basta modificar a string definida no rótulo hw (linha 19).

**19. Um dos exemplos em Assembly disponíveis no Moodle é parimp2.s, que lê um número inteiro e informa se ele é par ou ímpar. Seguindo as orientações encontradas no início do código fonte, monte o programa e teste o seu funcionamento.**

**20. Usando parimp2.s como base, escreva um programa em Assembly que leia um número inteiro e informe se ele é maior que, menor que, ou igual a 51.**

**21. Escreva um programa em Assembly que leia dois números inteiros e informe se o primeiro é maior que, menor que, ou igual ao segundo.**

**22. Escreva um programa em Assembly que exiba a média entre dois números inteiros lidos da entrada padrão.**

**23. Escreva um programa em Assembly que mostre a soma e a diferenc¸a de dois números inteiros lidos da entrada padrão. A diferença deve ser zero (caso os dois números sejam iguais) ou positiva (o maior subtraído do menor).**

**24. Escreva um programa em Assembly que leia dois números inteiros x e y calcule x^y usando:**

**(a) multiplicação**

BX <- x

CX <- y

AX <- BX \* BX \* //CX vezes

**(b) apenas somas**

Loop interno

AX <- prod + prod + prod + //DX vezes //loop soma

**25. Uma subrotina em Assembly tem três argumentos (parâmetros de entrada), a, b e c, e duas variáveis locais, i e j. Tanto os parâmetros quanto às variáveis locais s o inteiros de 2 bytes.**

func (int a, int b, int c)

int i, j;

PUSH c BP+8 c

PUSH b BP+6 b

PUSH a BP+4 a

CALL func BP+2 end.ret.

BP -> BP salvo

BP-2 i

BP-4 j

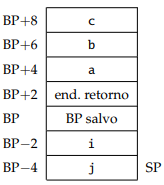
200

300 <- SP

SI <- 250

**(a) Seguindo o modelo da Figura C.3 (página 421 do livro), mostre como ficaria a pilha durante a execução desta subrotina. Associe os argumentos e variáveis locais aos respectivos endereços relativos a BP, e indique onde ficaria SP.**

A pilha fica com a seguinte configuração:



**(b) Mostre como ficaria essa pilha após a execução do seguinte conjunto de instruções:**

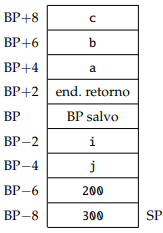
**1 PUSH 200**

**2 PUSH 250**

**3 POP SI**

**4 PUSH 300**

**Qual o conteúdo do registrador SI?**



**26. Escreva uma subrotina print3 em Assembly que imprima três números passados como parâmetro na pilha, equivalente ao código C void**

**print3 (int x, int y, int z) {**

**printf("x=%d, y=%d, z=%d\n", x, y, z);**

**}**

**27. Usando readnum.s como base, escreva uma nova versão da subrotina readnum que armazene o número lido em um endereço passado como parâmetro na pilha, em vez de devolvê-lo em AX. Isso seria o equivalente ao código C void readnum(int &x)**

**onde o valor lido seria armazenado em x.**

**28. Usando parimp1.s e readnum.s como base, escreva uma subrotina em Assembly que implemente a função strlen(), retornando em AX o número de caracteres de uma ´ string passada como parâmetro na pilha.**

**29. Escreva uma subrotina em Assembly que calcule a média aritmética de quatro números passados como parâmetro na pilha. A média calculada deve ser retornada em AX.**

**30. Escreva uma subrotina em Assembly que calcule a média aritmética de uma quantidade arbitrária de números passados como parâmetro na pilha. O primeiro parâmetro deve ser a quantidade de números a processar. Por exemplo, para calcular a média de 4, 9 e 5, a chamada em C equivalente seria média(3, 4, 9, 5). A média calculada deve ser retornada em AX.**

**31. Foi solicitado a um aluno de SOP que escrevesse um programa Assembly para ler dois números x e y e determinar se x e divisível por y. A interface com o programa deveria se parecer com o seguinte:**

Entre o 1o numero: 45

Entre o 2o número: 5

45 eh divisível por 5

Entre o 1o número: 47

Entre o 2o número: 5

47 nao eh divisivel por 5

O código produzido pelo aluno está mostrado a seguir. Infelizmente, ele possui alguns erros que impedem o seu bom funcionamento. Identifique esses erros, e corrija-os de modo que o programa funcione corretamente.

1 ! testa divisibilidade em Assembly as88

2 ! para montar o programa:

3 ! $ ./as88 diviserr.s readnum.s

4

5 #include "../syscalnr.h"

6

7 .SECT .TEXT

8 start:

9 MOV BP,SP

10 PUSH pr1

11 PUSH \_PRINTF

12 SYS

13 CALL readnum

14 MOV (x),AX

15

16 PUSH pr2

17 PUSH \_PRINTF

18 SYS

19 CALL readnum

20 MOV (y),AX

21

22 MOV CX,(x)

23 MOV DX,0

24 MOV AX,(y)

25 IDIV CX

26 CMP DX,0

27 JNZ divisivel

28

29 ndivisivel:

30 MOV BX,str\_ndiv

31 JMP imprime

32

33 divisivel:

34 MOV BX,str\_div

35

36 imprime:

37 PUSH pr3

38 PUSH (x)

39 PUSH BX

40 PUSH (y)

41 PUSH \_PRINTF

42 SYS

43

44 encerra:

45 MOV SP,BP

46 PUSH 0

47 PUSH \_EXIT

48 SYS

49 fim:

50

51 .SECT .DATA

52 pr1: .ASCIZ ’Entre o 1o numero: ’

53 pr2: .ASCIZ ’Entre o 2o número: ’

54 str\_div: .ASCIZ ’eh’

55 str\_ndiv: .ASCIZ ’nao eh’

56 pr3: .ASCIZ ’\n%d %s divisível por %d\n’

57 x: .WORD 0

58 y: .WORD 0

59

60 .SECT .BSS

RESPOSTA:

O programa possui três erros:

i. Linhas 22 e 24: o dividendo está trocado com o divisor. O correto seria armazenar o dividendo em AX e o divisor em CX:

MOV AX,(x)

MOV DX,0

PUSH CX,(y)

ii. Linha 27: a condição do desvio está errada. Se x for divisível por y, o resto da divisão (em DX) ser a zero, de modo que a instrução correta seria JZ divisível.

iii. Linhas 37–40: os parâmetros de \_PRINTF deveriam ser empilhados na ordem inversa:

PUSH (y)

PUSH BX

PUSH (x)

PUSH pr3