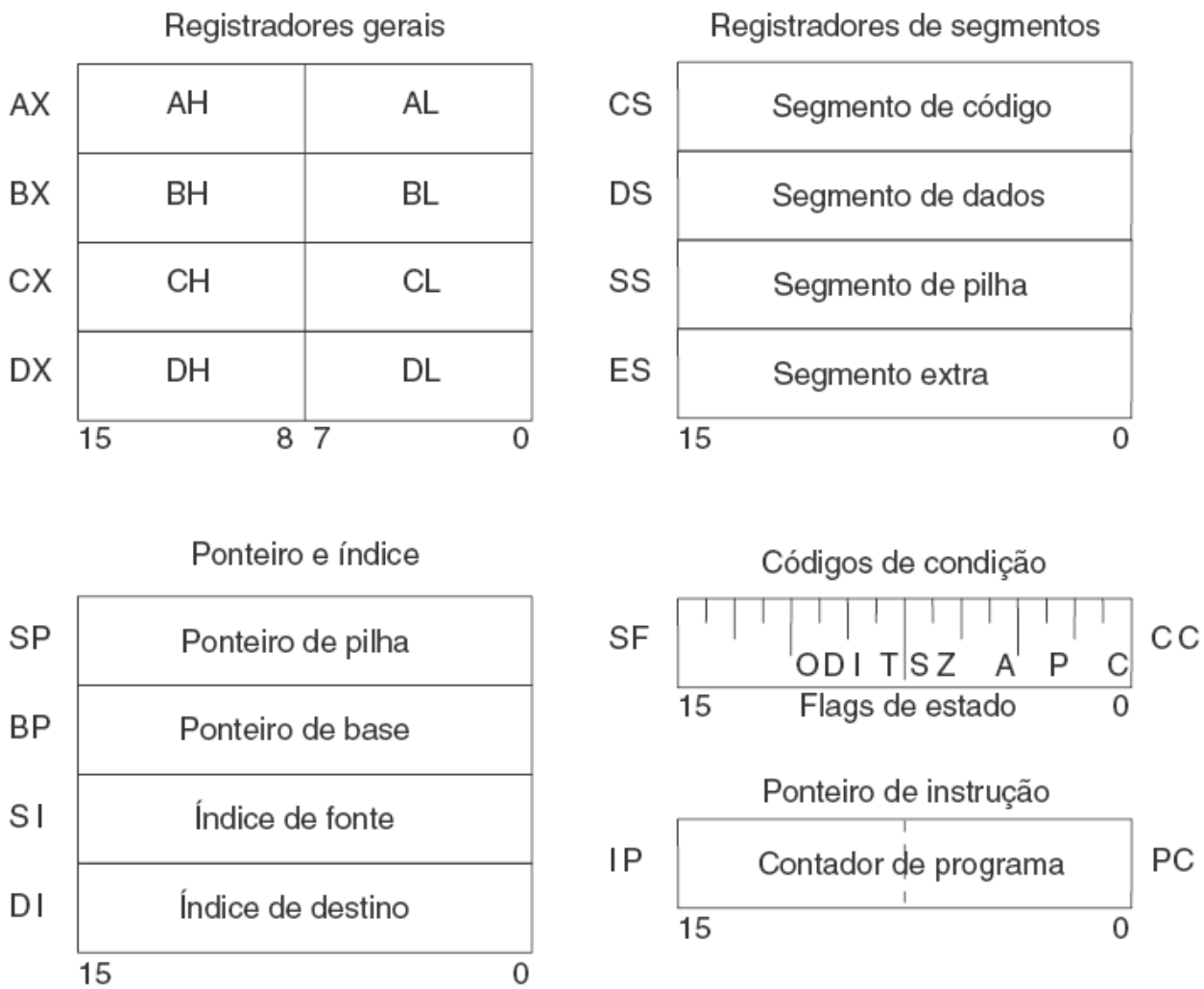
Software Básico

Programação assembly do 8088

Apêndice C

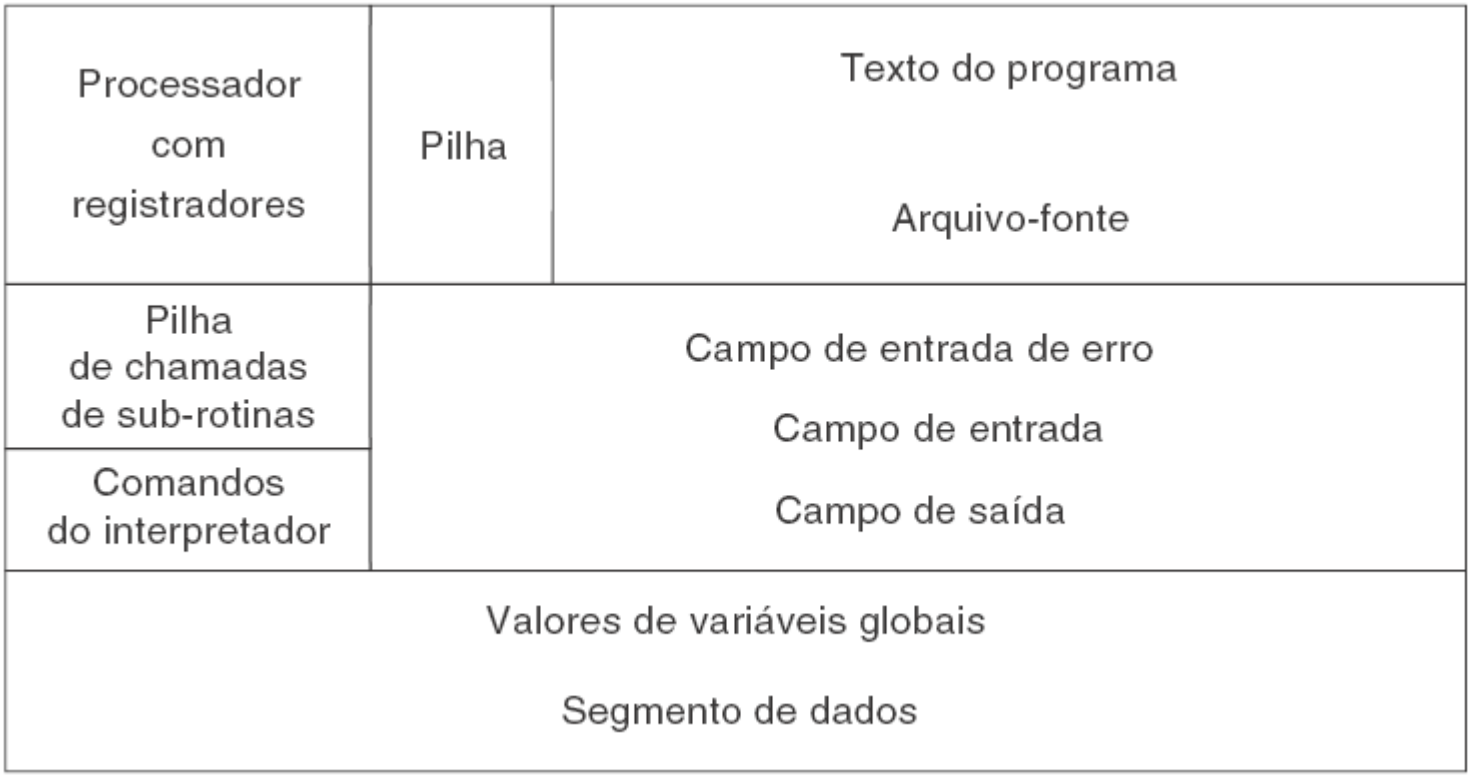
UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

O processador 8088



O rastreador/interpretador/simulador

As janelas do rastreador.



Um pequeno programa em execução

```
    _EXIT = 1                   ! 1
    _WRITE  = 4                 ! 2
    _STDOUT  = 1                ! 3
.SECT .TEXT                     ! 4
start:                          ! 5
    MOV    CX,de-hw             ! 6
    PUSH   CX                   ! 7
    PUSH   hw                   ! 8
    PUSH   _STDOUT              ! 9
    PUSH   _WRITE               !10
    SYS                         !11
    ADD    SP, 8                !12
    SUB    CX,AX                !13
    PUSH   CX                   !14
    PUSH   _EXIT                !15
    SYS                         !16
.SECT .DATA                     !17
hw:                             !18
.ASCII "Hello World\n"          ! 19
de: .BYTE 0                     ! 20
```

(a)

```
CS: 00   DS=SS=ES: 002
AH:00 AL:0c   AX:     12
BH:00 BL:00   BX:      0
CH:00 CL:0c   CX:     12
DH:00 DL:00   DX:      0
SP: 7fd8  SF  O D S Z C
BP: 0000  CC  - > p - -
SI: 0000   IP:000c:PC
DI: 0000   start + 7
```

```
=>0004
  0001
  0000
  000c
```

```
        MOV   CX,de-hw      !  6
        PUSH  CX            !  7
        PUSH  HW            !  8
        PUSH  _STDOUT       !  9
        PUSH  _WRITE        ! 10
        SYS                 ! 11
=>      ADD   SP,8          ! 12
        SUB   CX,AX         ! 13
        PUSH  CX            ! 14
```

```
hw
■
```

```
E
I


> Hello World\n
```

```
hw + 0 = 0000: 48 65 6c 6c 6f 20 57 6f Hello World 25928
```

(b)

(a) Um programa em linguagem de montagem. (b) A tela correspondente do rastreador.

Endereçamento (1)

Modos de endereçamento de operandos. O símbolo # indica um valor numérico ou rótulo.

Modo	Operando	Exemplos
Endereçamento de registrador		
Registrador de bytes	Registrador de bytes	AH,AL,BH,BL,CH,CL,DH,DL
Registrador de palavras	Registrador de palavras	AX,BX,CX,DX,SP,BP,SI,DI
Endereçamento de segmentos de dados		
Endereço direto	Endereço vem depois do opcode	(#)
Indireto de registrador	Endereço em registrador	(SI), (DI), (BX)
Deslocamento de registrador	Endereço em registrador+deslocto.	#(SI), #(DI), #(BX)
Registrador com índice	Endereço é BX + SI/DI	(BX)(SI), (BX)(DI)
Registrador com índice e deslocamento	BX + SI DI + deslocamento	#(BX)(SI), #(BX)(DI)

Endereçamento (2)

Modos de endereçamento de operandos. O símbolo # indica um valor numérico ou rótulo.

Modo	Operando	Exemplos
Endereço de segmento de pilha		
Indireto de ponteiro de base	Endereço em registrador	(BP)
Deslocamento de ponteiro de base	Endereço é BP + deslocto.	#(BP)
Ponteiro de base com índice	Endereço é BP + SI/DI	(BP)(SI), (BP)(DI)
Deslocto. de índice de ponteiro de base	BP+SI/DI + deslocamento	#(BP)(SI), #(BP)(DI)
Dados imediatos		
Byte/palavra imediato	Dados são parte da instrução	#
Endereço implícito		
Instrução push/pop	Endereço indireto (SP)	PUSH, POP, PUSHF, POPF
Flags de carregamento/armazenamento	Registrador de status de flag	LAHF, STC, CLC, CMC
Traduzir XLAT	AL, BX	XLAT
Instruções de cadeias repetidas	(SI), (DI), (CX)	MOVS, CMPS, SCAS
Instruções de E/S	AX, AL	IN #, OUT #
Converte byte, palavra	AL, AX, DX	CBW,CWD

O conjunto de instruções do 8088 (1)

Algumas das instruções mais importantes do 8088.

Mnemônico	Descrição	Operandos	Flags de status			
			O	S	Z	C
MOV(B)	Mover palavra, byte	r ← e, e ← r, e ← #	-	-	-	-
XCHG(B)	Trocar palavra	r ↔ e	-	-	-	-
LEA	Carregar endereço efetivo	r ← #e	-	-	-	-
PUSH	Passar para pilha	e,#	-	-	-	-
POP	Retirar da pilha	e	-	-	-	-
PUSHF	Push Flags	-	-	-	-	-
POPF	Pop Flags	-	-	-	-	-
XLAT	Traduzir AL	-	-	-	-	-
ADD(B)	Somar palavra	r ← e, e ← r, e ← #	*	*	*	*
ADC(B)	Somar palavra com vai-um	r ← e, e ← r, e ← #	*	*	*	*
SUB(B)	Subtrair palavra	r ← e, e ← r, e ← #	*	*	*	*
SBB(B)	Subtrair palavra com empréstimo	r ← e, e ← r, e ← #	*	*	*	*
IMUL(B)	Multiplicar com sinal	e				
MUL(B)	Multiplicar sem sinal	e	*	∪	∪	*
IDIV(B)	Dividir com sinal	e	∪	∪	∪	∪
DIV(B)	Dividir sem sinal	e	∪	∪	∪	∪

O conjunto de instruções do 8088 (2)

Algumas das instruções mais importantes do 8088.

Mnemônico	Descrição	Operandos	Flags de status			
CBW	Estender Byte/palavra com sinal	-	-	-	-	-
CWD	Estender Palavra/dupla com sinal	-	-	-	-	
NEG(B)	Nega binário e	*	*	*	*	*
NOT(B)	Complemento lógico	e	-	-	-	-
INC(B)	Incrementar destino	e	*	*	*	-
DEC(B)	Decrementar destino	e	*	*	*	-
AND(B)	AND lógico	e ← r, r ← e, e ← #	0	*	*	0
OR(B)	OR lógico	e ← r, r ← e, e ← #	0	*	*	0
XOR(B)	OR exclusivo lógico	e ← r, r ← e, e ← #	0	*	*	0
SHR(B)	Deslocamento lógico para a direita	e ← 1, e ← CL	*	*	*	*
SAR(B)	Deslocamento aritmético para a direita	e ← 1, e ← CL	*	*	*	*
SAL(B)	(=SHL(B)) deslocar para a esquerda	e ← 1, e ← CL	*	*	*	*
ROL(B)	Fazer rotação para a esquerda	e ← 1, e ← CL	*	-	-	*
ROR(B)	Fazer rotação para a direita	e ← 1, e ← CL	*	-	-	*
RCL(B)	Fazer rotação para a esquerda com vai-um	e ← 1, e ← CL	*	-	-	*
RCR(B)	Fazer rotação para a direita com vai-um	e ← 1, e ← CL	*	-	-	*

O conjunto de instruções do 8088 (3)

Algumas das instruções mais importantes do 8088.

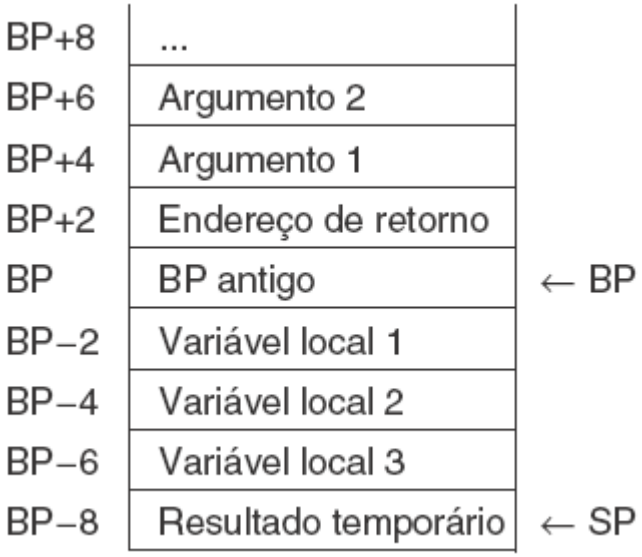
Mnemônico	Descrição	Operandos	Flags de status			
TEST(B)	Testar operandos	e ↔ r, e ↔ #	0	*	*	0
CMP(B)	Compare operandos	e ↔ r, e ↔ #	*	*	*	*
STD	Ajuste flag de direção(↓)	-	-	-	-	-
CLD	Liberar flag de direção (↑)	-	-	-	-	-
STC	Ajustar flag de vai-um	-	-	-	-	1
CLC	Liberar flag de vai-um	-	-	-	-	0
CMC	Complementar vai-um	-	-	-	-	*
LOOP	Saltar para trás se CX i ≥ 0 decrementado	rótulo	-	-	-	-
LOOPZ LOOPE	Para trás se Z=1 e DEC(CX)i ≥ 0	rótulo	-	-	-	-
LOOPNZ LOOPNE	Para trás se Z=0 e DEC(CX)i ≥ 0	rótulo	-	-	-	-
REP REPZ REPNZ	Repetir instrução de cadeia	instrução de cadeia	-	-	-	-
MOVS(B)	Mover cadeia de palavra	-	-	-	-	-
LODS(B)	Carregar cadeia de palavra	-	-	-	-	-
STOS(B)	Armazenar cadeia de palavra	-	-	-	-	-
SCAS(B)	Examinar cadeia de palavra	-	-	-	-	-
CMPS(B)	Comparar cadeia de palavra	-	*	*	*	*
JCC	Saltar conforme condições	rótulo	*	*	*	*
JMP	Saltar para rótulo	e, rótulo	-	-	-	-
CALL	Saltar para sub-rotina	e, rótulo	-	-	-	-
RET	Retornar da sub-rotina	-,#	-	-	-	-
SYS	Exceção de chamada de sistema	-	-	-	-	-

Desvios condicionais

Saltos condicionais.

Instrução	Descrição	Quando saltar
JNA, JBE	Abaixo ou igual	CF=1 or ZF=1
JNB, JAE, JNC	Não abaixo de	CF=0
JE, JZ	Zero, igual	ZF=1
JNLE, JG	Maior que	SF=OF e ZF=0
JGE, JNL	Maior que ou igual	SF=OF
JO	Excesso	OF=1
JS	Sinal negativo	SF=1
JCXZ	CX é zero	CX=0
JB, JNAE, JC	Abaixo	CF=1
JNBE, JA	Acima CF50&ZF50	
JNE, JNZ	Não-zero, não-igual	ZF=0
JL, JNGE	Menos que	SF ≠, OF
JLE, JNG	Menos que ou igual a	SF ≠ OF or ZF=1
JNO	Não-excesso	OF=0
JNS	Não-negativo	SF=0

Chamadas de sub-rotinas



Chamadas de sistema e sub-rotinas de sistema

Algumas chamadas de sistema e sub-rotinas UNIX disponíveis no interpretador.

No	Nome	Argumentos	Valor de retorno	Descrição
5	–OPEN	*nome, 0/1/2	descritor de arquivo	Abra arquivo
8	–CREAT	*nome, *modo	descritor de arquivo	Crie arquivo
3	–READ	fd, buf, nbytes	# bytes	Leia nbytes para buffer buf
4	–WRITE	fd, buf, nbytes	# bytes	Escreva nbytes a partir do buffer buf
6	–CLOSE	fd	0 para sucesso	Feche arquivo com fd
19	–LSEEK	fd, offset(long), 0/1/2	posição (longo)	Mova ponteiro de arquivo
1	–EXIT	status		Feche arquivos Pare processo
117	–GETCHAR		leia caractere	Leia caractere da entrada-padrão
122	–PUTCHAR	char	escreva byte	Escreva caractere para saída-padrão
127	–PRINTF	*format, arg		Imprima formatado na saída-padrão
121	–SPRINTF	buf, *format, arg		Imprima formatado em buffer buf
125	–SSCANF	buf, *format, arg		Leia argumentos de buffer buf

Empilhe argumentos da direita pra esquerda, empilhe # da chamada, execute instrução SYS, sem operandos

O assembler as88

As pseudo-instruções do as88.

Instrução	Descrição
.SECT .TEXT	Monte as linhas seguintes na seção TEXT
.SECT .DATA	Monte as linhas seguintes na seção DATA
.SECT .BSS	Monte as linhas seguintes na seção BSS
.BYTE	Monte os argumentos como uma seqüência de bytes
.WORD	Monte os argumentos como uma seqüência de palavras
.LONG	Monte os argumentos como uma seqüência de longos
.ASCII "str"	Armazene str como ascii uma cadeia sem um byte zero no final
.ASCIZ "str"	Armazene str como ascii uma cadeia com um byte zero no final
.SPACEn	Avance o contador de localização n posições
.ALIGNn	Avance o contador de localização até uma fronteira de n bytes
.EXTERN	Identificador é um nome externo

Não tem suporte à definição de macros

O assembler as88

Alguns dos caracteres especiais permitidos por as88.

Símbolo de escape	Descrição
\n	Nova linha (avanço de linha)
\t	Tab
\\	Barra invertida
\b	Retrocesso
\f	Alimentação de formulário
\r	Retorno de carro
\"	Aspas duplas

Exemplo Hello World

```
   _EXIT = 1                    ! 1
   _WRITE  = 4                  ! 2
   _STDOUT  = 1                 ! 3
.SECT .TEXT                     ! 4
start:                          ! 5
   MOV   CX,de-hw               ! 6
   PUSH  CX                     ! 7
   PUSH  hw                     ! 8
   PUSH  _STDOUT                ! 9
   PUSH  _WRITE                 !10
   SYS                          !11
   ADD   SP, 8                  !12
   SUB   CX,AX                  !13
   PUSH  CX                     !14
   PUSH  _EXIT                  !15
   SYS                          !16
.SECT .DATA                     !17
hw:                             !18
.ASCII "Hello World\n"          !19
de: .BYTE 0                     !20
```

(a)

```
CS: 00   DS=SS=ES: 002
AH:00 AL:0c   AX:     12
BH:00 BL:00   BX:      0
CH:00 CL:0c   CX:     12
DH:00 DL:00   DX:      0
SP: 7fd8  SF  O D S Z C
BP: 0000  CC  - > p - -
SI: 0000   IP:000c:PC
DI: 0000   start + 7
```

```
=>0004
  0001
  0000
  000c
```

```
        MOV   CX,de-hw    !  6
        PUSH  CX          !  7
        PUSH  HW          !  8
        PUSH  _STDOUT     !  9
        PUSH  _WRITE      ! 10
        SYS               ! 11
=>      ADD   SP,8        ! 12
        SUB   CX,AX       ! 13
        PUSH  CX          ! 14
```

```
hw
■
```

```
E
I


> Hello World\n
```

```
hw + 0 = 0000: 48 65 6c 6c 6f 20 57 6f  Hello World 25928
```

(b)

Programa vecprod.s *(1)*

Calcula produto interno de dois vetores, vec1 e vec2

```
_EXIT      = 1                  ! 1  defina o valor de _EXIT
_PRINTF    = 127                ! 2  defina o valor de _PRINTF
.SECT .TEXT                     ! 3  inicie o segmento TEXT
inpstart:                       ! 4  defina rótulo inpstart
        MOV  BP,SP              ! 5  salve SP em BP
        PUSH vec2               ! 6  passe endereço de vec2
        PUSH vec1               ! 7  passe endereço de vec1
        MOV CX,vec2-vec1        ! 8  CX = número de bytes em vetor
        SHR  CX,1               ! 9  CX = número de palavras em vetor
        PUSH CX                 ! 10 passe contagem de palavra
        CALL vecmul             ! 11 chame vecmul
        MOV  (inprod),AX        ! 12 mova AX
        PUSH AX                 ! 13 passe resultado a ser impresso
        PUSH pfmt               ! 14 passe endereço de cadeia de formato
        PUSH _PRINTF            ! 15 passe código de função para PRINTF
        SYS                     ! 16 chame a função PRINTF
        ADD  SP,12              ! 17 limpe a pilha
        PUSH 0                  ! 18 passe código de status
        PUSH _EXIT              ! 19 passe código de função para EXIT
        SYS                     ! 20 chame a função EXIT
```

Programa vecprod.s *(2)*

```
vecmul:
        PUSH BP          ! 21  início de vecmul(count, vec1, vec2)
        MOV  BP,SP       ! 22  salve BP na pilha
        MOV  CX,4(BP)    ! 23  copie SP para BP para acessar argumentos
        MOV  SI,6(BP)    ! 24  ponha contagem em CX para controlar laço
        MOV  DI,8(BP)    ! 25  SI = vec1
        PUSH 0           ! 26  DI = vec2
                         ! 27  passe 0 para pilha
1:      LODS             ! 28  mova (SI) para AX
        MUL  (DI)        ! 29  multiplique AX por (DI)
        ADD  -2(BP),AX   ! 30  Some AX com valor acumulado na memória
        ADD  DI,2        ! 31  Incremente DI para apontar o próximo elemento
        LOOP 1b          ! 32  se CX > 0, volte ao rótulo 1
        POP  AX          ! 33  Retire topo da pilha para AX
        POP  BP          ! 34  Restaure BP
        RET              ! 35  Retorne da sub-rotina
```

Programa vecprod.s *(3)*

O programa *vec-prod.s*.

```
.SECT .DATA                                  ! 36  inicie segmento DATA
pfmt: .ASCIZ "Produto interno é : %d\n"      ! 37  defina cadeia
.ALIGN 2                                     ! 38  force endereço par
vec1:.WORD 3,4,7,11,3                        ! 39  vetor 1
vec2:.WORD 2,6,3,1,0                         ! 40  vetor 2
.SECT .BSS                                   ! 41  inicie segmento BSS
inprod:     .SPACE 2                         ! 42  aloque espaço para inprod
```

Programa vecprod.s *(4)*

Execução de vecprod.s
quando ele alcança a linha
28 pela primeira vez.

```
    MOV  BP,SP          ! 5
    PUSH vec2           ! 6
    PUSH vec1           ! 7
    MOV  CX,vec2-vec1   ! 8
    SHR  CX,1           ! 9
    PUSH CX             !10
    CALL vecmul         !11
    ---------
vecmul:                 !21
    PUSH BP             !22
    MOV  BP,SP          !23
    MOV  CX,4(BP)       !24
    MOV  SI,6(BP)       !25
    MOV  DI,8(BP)       !26
    PUSH 0              !27
1:  LODS                !28
    MUL  (DI)           !29
    ADD  -2(BP),AX      !30
    ADD  DI,2           !31
    LOOP 1b             !32
```

```
CS:00  DS=SS=ES0004
AH:00 AL:00  AX:     0
BH:00 BL:00  BX:     0
CH:00 CL:05  CX:     5
DH:00 DL:00  DX:     0
SP: 7fb4 SF  O D S Z C
BP: 7fb6 CC  - > p z -
SI: 0018  IP:0031:PC
DI: 0022  vecmul+7
```

```
=>0000
  7fc0
1 0011
  0005
  0018
  0022
```

```
    PUSH BP             ! 22
    MOV  BP,SP          ! 23
    MOV  CX,4(BP)       ! 24
    MOV  SI,6(BP)       ! 25
    MOV  DI,8(BP)       ! 26
    PUSH                  27
=>1:LODS                ! 28
    MUL  (DI              29
    ADD  -2(BP),AX      ! 30
```

```
1 <= inpstart + 7
```

```
■
```

```
I


>
```

```
vec1+0  =0018: 3  0  4  0  7  0  b  0 ..........    3
vec2+0  =0022: 2  0  6  0  3  0  1  0 ..........    2
pfmt+0  =0000:54 68 65 20 69 6e 20 70 The in  prod 26708
pfmt+18 =0012:25 64 21  a  0  0  3  0 %d!........25637
```

Programa arrayprt

Imprime vetor de inteiros vec1

```
#include "../syscalnr.h"          !  1

.SECT .TEXT                       !  2
vecpstrt:                         !  3
   MOV  BP,SP                     !  4
   PUSH vec1                      !  5
   MOV  CX,frmatstr-vec1          !  6
   SHR  CX                        !  7
   PUSH CX                        !  8
   CALL vecprint                  !  9
   MOV  SP,BP                     ! 10
   PUSH 0                         ! 11
   PUSH _EXIT                     ! 12
   SYS                            ! 13

.SECT .DATA                       ! 14
vec1:   .WORD 3,4,7,11,3          ! 15
frmatstr: .ASCIZ "%s"             ! 16

frmatkop:                         ! 17
.ASCIZ "The array contains "      ! 18
frmatint: .ASCIZ " %d"            ! 19

.SECT .TEXT                       ! 20
vecprint:                         ! 21
   PUSH BP                        ! 22
   MOV  BP,SP                     ! 23
   MOV  CX,4(BP)                  ! 24
   MOV  BX,6(BP)                  ! 25
   MOV  SI,0                      ! 26
   PUSH frmatkop                  ! 27
   PUSH frmatstr                  ! 28
   PUSH _PRINTF                   ! 29
   SYS                            ! 30
   MOV  -4(BP),frmatint           ! 31
1: MOV  DI,(BX)(SI)               ! 32
   MOV  -2(BP),DI                 ! 33
   SYS                            ! 34
   INC  SI                        ! 35
   LOOP 1b                        ! 36
   PUSH '\n'                      ! 37
   PUSH _PUTCHAR                  ! 38
   SYS                            ! 39
   MOV  SP,BP                     ! 40
   RET                            ! 41
```

Programa strngcpy.s

Copia uma cadeia de caracteres (mesg2) para um destino (mesg1)

```
.SECT .TEXT
stcstart:                          ! 1
   PUSH mesg1                      ! 2
   PUSH mesg2                      ! 3
   CALL strngcpy                   ! 4
   ADD  SP,4                       ! 5
   PUSH 0                          ! 6
   PUSH 1                          ! 7
   SYS                             ! 8
strngcpy:                          ! 9
   PUSH CX                         ! 10
   PUSH SI                         ! 11
   PUSH DI                         ! 12
   PUSH BP                         ! 13
   MOV  BP,SP                      ! 14
   MOV  AX,0                       ! 15
   MOV  DI,10(BP)                  ! 16
   MOV  CX,–1                      ! 17
   REPNZ SCASB                     ! 18
   NEG  CX                         ! 19
   DEC CX                          ! 20
   MOV  SI,10(BP)                  ! 21
   MOV  DI,12(BP)                  ! 22
   PUSH DI                         ! 23
   REP  MOVSB                      ! 24
   CALL stringpr                   ! 25
   MOV  SP,BP                      ! 26
   POP  BP                         ! 27
   POP  DI                         ! 28
   POP  SI                         ! 29
   POP  CX                         ! 30
   RET                             ! 31
.SECT .DATA                        ! 32
mesg1: .ASCIZ "Have a look\n"      ! 33
mesg2: .ASCIZ "qrst\n"             ! 34
.SECT .BSS
```

(a)

Salva regs que vao ser usados localmente

MOVSB = move byte apontado por SI para DI

REP : repete instrução enquanto CX ≠ 0

SCASB = compara caracter apontado por DI com AL (ajusta flag 0 quando achar 0)

REPNZ: repete enquanto CX ≠ 0 e bit Zero não for setado

Programa reverspr.s

Imprime cadeia na ordem inversa

```
#include "../syscalnr.h"          ! 1

start: MOV DI,str                 ! 2
   PUSH AX                        ! 3
   MOV BP,SP                      ! 4
   PUSH _PUTCHAR                  ! 5
   MOVB AL,'\n'                   ! 6
   MOV CX,-1                      ! 7
   REPNZ SCASB                    ! 8
   NEG CX                         ! 9
   STD                            ! 10
   DEC CX                         ! 11
   SUB DI,2                       ! 12
MOV SI,DI                         ! 13
   1: LODSB                       ! 14
   MOV (BP),AX                    ! 15
   SYS                            ! 16
   LOOP  1b                       ! 17
   MOVB  (BP),'\n'                ! 18
   SYS                            ! 19
   PUSH 0                         ! 20
   PUSH _EXIT                     ! 21
SYS                               ! 22
.SECT .DATA                       ! 23
str: .ASCIZ "reverse\n"           ! 24
```

LODSB : copia um byte apontado por SI para AX (AL)

Este programa tem um erro semântico

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Tabelas de despacho (1)

Um programa que demonstra um desvio multivias usando uma tabela de despacho.

```
#include "../syscalnr.h"  ! 1
.SECT .TEXT               ! 2
jumpstrt:                 ! 3
    PUSH strt             ! 4
    MOV  BP,SP            ! 5
    PUSH _PRINTF          ! 6
    SYS                   ! 7
    PUSH _GETCHAR         ! 8
1:  SYS                   ! 9
    CMP  AX,5             ! 10
    JL   8f               ! 11
    CMPB AL,'0'           ! 12
    JL   1b               ! 13
    CMPB AL,'9'           ! 14
    JLE  2f               ! 15
    MOVB AL,'9'+1         ! 16
2:  MOV  BX,AX            ! 17
    AND  BX,0Xf           ! 18
    SAL  BX,1             ! 19
    CALL tbl(BX)          ! 20
    JMP  1b               ! 21
8:  PUSH 0                ! 22
    PUSH _EXIT            ! 23
    SYS                   ! 24
```

```
r out0:  MOV  AX,mes0     ! 25
         JMP  9f          ! 26
r out1:  MOV  AX,mes1     ! 27
         JMP  9f          ! 28
 rout2:  MOV  AX,mes2     ! 29
         JMP  9f          ! 30
 rout3:  MOV  AX,mes3     ! 31
         JMP  9f          ! 32
 rout4:  MOV  AX,mes4     ! 33
         JMP  9f          ! 34
 rout5:  MOV  AX,mes5     ! 35
         JMP  9f          ! 36
 rout6:  MOV  AX,mes6     ! 37
         JMP  9f          ! 38
 rout7:  MOV  AX,mes7     ! 39
         JMP  9f          ! 40
 rout8:  MOV  AX,mes8     ! 41
         JMP  9f          ! 42
 erout:  MOV  AX,emes     ! 43
 9:      PUSH AX          ! 44
         PUSH _PRINTF     ! 45
         SYS              ! 46
         ADD  SP,4        ! 47
         RET              ! 48
```

UFMG - ICEx
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

CALL tbl(BX) : chamada de função cujo endereço é dado pela soma do valor de BX com o endereço tbl

Tabelas de despacho (2)

Um programa que demonstra um desvio multivias usando uma tabela de despacho.

```
.SECT .DATA                                                              ! 49
tbl: .WORD rout0,rout1,rout2,rout3,rout4,rout5,rout6,rout7,rout8,rout8,erout   ! 50
mes0: .ASCIZ "Isso é um zero.\n"                                         ! 51
mes1: .ASCIZ "Que tal um um.\n"                                          ! 52
mes2: .ASCIZ "Você pediu um dois.\n"                                     ! 53
mes3: .ASCIZ "O dígito era um três.\n"                                   ! 54
mes4: .ASCIZ "Você digitou um quatro.\n"                                 ! 55
mes5: .ASCIZ "Você preferiu um cinco.\n"                                 ! 56
mes6: .ASCIZ "Um seis foi encontrado.\n"                                 ! 57
mes7: .ASCIZ "Esse é o número sete.\n"                                   ! 58
mes8: .ASCIZ "Esse dígito não é aceito como um octal.\n"                 ! 59
emes: .ASCIZ "Isso não é um dígito. Tente novamente.\n"                  ! 60
strt:    .ASCIZ  "Digite um dígito octal com um retorno. Pare em final de arquivo.\n"   ! 61
```

Acesso a arquivo com buffer e aleatório (1)

Um programa de acesso a arquivo por leitura de *buffer* e aleatório.

```
#include "../syscalnr.h" ! 1
bufsiz = 512             ! 2
.SECT .TEXT              ! 3
infbufst:                ! 4
    MOV  BP,SP           ! 5
    MOV  DI,linein       ! 6
    PUSH _GETCHAR        ! 7
1:  SYS                  ! 8
    CMPB AL,'\n'         ! 9
    JL   9f              ! 10
    JE   1f              ! 11
    STOSB                ! 12
    JMP  1b              ! 13
1:  PUSH 0               ! 14
    PUSH linein          ! 15
    PUSH _OPEN           ! 16
    SYS                  ! 17
    CMP  AX,0            ! 18
    JL   9f              ! 19
    MOV  (fildes),AX     ! 20
    MOV  SI,linh+2       ! 21
    MOV  BX,0            ! 22
1:  CALL fillbuf         ! 23
    CMP  CX,0            ! 24
    JLE  3f              ! 25
2:  MOVB AL,'\n'         ! 26
    REPNE SCASB          ! 27
    JNE  1b              ! 28
    INC  (count)         ! 29
    MOV  AX,BX           ! 30
    SUB  AX,CX           ! 31
    XCHG SI,DI           ! 32
    STOS                 ! 33
    XCHG SI,DI           ! 34
    CMP  CX,0            ! 35
    JNE  2b              ! 36
    JMP  1b              ! 37
9:  MOV  SP,BP           ! 38
    PUSH linein          ! 39
    PUSH errmess         ! 40
    PUSH _PRINTF         ! 41
    SYS                  ! 42
```

Acesso a arquivo com buffer e aleatório (2)

Um programa de acesso a arquivo por leitura de *buffer* e aleatório.

```
        PUSH _EXIT          ! 43
        PUSH _EXIT          ! 44
        SYS                 ! 45
3:      CALL getnum         ! 46
        CMP  AX,0           ! 47
        JLE  8f             ! 48
        MOV  BX,(curlin)    ! 49
        CMP  BX,0           ! 50
        JLE  7f             ! 51
        CMP  BX,(count)     ! 52
        JG   7f             ! 53
        SHL  BX,1           ! 54
        MOV  AX,linh-2(BX)  ! 55
        MOV  CX,linh(BX)    ! 56
        PUSH 0              ! 57
        PUSH 0              ! 58
        PUSH AX             ! 59
        PUSH (fildes)       ! 60
        PUSH _LSEEK         ! 61
        SYS                 ! 62
        SUB  CX,AX          ! 63
        PUSH CX             ! 64
        PUSH buf            ! 65
        PUSH (fildes)       ! 66
        PUSH _READ          ! 67
        SYS                 ! 68
        ADD  SP,4           ! 69
        PUSH 1              ! 70
        PUSH _WRITE         ! 71
        SYS                 ! 72
        ADD  SP,14          ! 73
        JMP  3b             ! 74
8:      PUSH scanerr        ! 75
        PUSH _PRINTF        ! 76
        SYS                 ! 77
        ADD  SP,4           ! 78
        JMP  3b             ! 79
7:      PUSH 0              ! 80
        PUSH _EXIT          ! 81
        SYS                 ! 82
fillbuf:                    ! 83
        PUSH bufsiz         ! 84
```

Acesso a arquivo com buffer e aleatório (3)

Um programa de acesso a arquivo por leitura de *buffer* e aleatório.

```
        PUSH buf                ! 85
        PUSH (fildes)           ! 86
        PUSH _READ              ! 87
        SYS                     ! 88
        ADD  SP,8               ! 89
        MOV  CX,AX              ! 90
        ADD  BX,CX              ! 91
        MOV  DI,buf             ! 92
        RET                     ! 93

getnum:                         ! 94
        MOV  DI,linein          ! 95
        PUSH _GETCHAR           ! 96
1:      SYS                     ! 97
        CMPB AL,'\n'            ! 98
        JL   9b                 ! 99
        JE   1f                 !100
        STOSB                   !101
        JMP  1b                 !102
1:      MOVB (DI),' '           !103
        PUSH curlin             !104
        PUSH numfmt             !105
        PUSH linein             !106
        PUSH _SSCANF            !107
        SYS                     !108
        ADD  SP,10              !109
        RET                     !110

.SECT .DATA                     !111
errmess:                        !112
.ASCIZ "Open %s failed\n"       !113
numfmt:  .ASCIZ "%d"            !114
scanerr:                        !115
.ASCIZ "Type a number.\n"       !116
.ALIGN 2                        !117
.SECT .BSS                      !118
linein:  .SPACE 80              !119
fildes:  .SPACE 2               !120
linh:    .SPACE 8192            !121
curlin:  .SPACE 4               !122
buf:     .SPACE bufsiz+2        !123
count:   .SPACE 2               !124
```