

# **Módulo 1 – Compiladores**

## **1.1 Questões Teóricas**

1.1.1.

## 1.2 Questões Práticas

1.2.1.

1.2.2.

1.2.3.	Código	Tabela de Símbolos	Código Máquina	
	02. EH_MUL: MACRO &N	PRINT1: 16		1
	02.     LOAD     &N	FIM:     18	10   N	2
	04.     DIV     TRES	N:       19	04   TRES	3
	06.     MUL     TRES	TRES:    20	03   TRES	4
	08.     SUB     &N	UM:      21	02   N	5
	10.     JMPZ    PRINT1	ZERO:    22	08   PRINT1	6
	END_MACRO			7
	00.     INPUT    N		12   N	8
	02.     EH_MUL   N		; escrito acima	9
	12.     OUTPUT   ZERO		13   ZERO	10
	14.     JMP      FIM		05   FIM	11
	16. PRINT1: OUTPUT   UM		13   UM	12
	18. FIM:   STOP		14	13
	19. N:     SPACE		00	14
	20. TRES:  CONST   3		03	15
	21. UM:    CONST   1		01	16
	22. ZERO:  CONST   0		00	17
				18

1.2.4.	Código	Tabela de Símbolos	Código Máquina	
	00. INPUT    N	WHILE:  04	12   N	1
	02. LOAD     N	FIM:     28	10   N	2
	04. WHILE:  JUMPZ   FIM	N:       29	08   FIM	3
	06.     DIV    DOIS	AUX:     30	04   DOIS	4
	08.     MUL    DOIS	DOIS:    31	03   DOIS	5
	10.     STORE   AUX		11   AUX	6
	12.     LOAD    N		10   N	7
	14.     SUB     AUX		02   AUX	8
	16.     STORE   AUX		11   AUX	9
	18.     OUTPUT   AUX		13   AUX	10
	20.     LOAD    N		10   N	11
	22.     DIV     DOIS		04   DOIS	12
	24.     STORE   N		11   N	13
	26.     JUMP    WHILE		05   WHILE	14
	28. FIM:   STOP		14	15
	29. N:     SPACE		00	16
	30. AUX:   SPACE		00	17
	31. DOIS:  CONST   2		02	18
				19

1.2.5.

1.2.6.

1.2.7. Os códigos de cada módulo são apresentados a seguir. Para facilitar a ligação, é mostrado o código máquina não-ligado de cada um.

Código (fator=0)	Tabela de Símbolos	Código Máquina	
00. MOD1:   BEGIN	MOD1:    00		1
00. MOD2:   EXTERN	MOD2:    00		2
			3

00. VALS:	EXTERN	VALS:	00			4
00. PUBLIC	L1	L1:	06			5
00. PUBLIC	L2	L2:	14			6
00.	INPUT	VALS	R:	15	12 VALS	7
02.	INPUT	VALS + 1			12 VALS + 1	8
04.	JMP	MOD2	Tabela de Usos	05	MOD2	9
06. L1:	LOAD	VALS	MOD2:	5+	10 VALS	10
08.	DIV	VALS + 1	VALS:	1+ 3+ 7+ 9+	04 VALS + 1	11
10.	STORE	RES			11 RES	12
12.	OUTPUT	RES	Tabela de Definições	13	RES	13
14. L2:	STOP	L1:	06	14		14
15. R:	SPACE	L2:	14	00		15
	END	R:	15			16

Código (fator=16)		Tabela de Símbolos		Código Máquina		
00. MOD2:	BEGIN	MOD2:	00			1
00. L1:	EXTERN	L1:	00			2
00. L2:	EXTERN	L2:	00			3
00. PUBLIC	VALS	VALS:	06			4
00. PUBLIC	MOD2					5
00. LOAD	VALS + 1	Tabela de Usos	10	VALS + 1		6
02. JMPZ	L2	L1:	5+	08	L2	7
04. JMP	L1	L2:	3+	05	L1	8
06. VALS:	SPACE 2			00	02	9
	END	Tabela de Definições				10
		MOD2:	00			11
		VALS:	06			12
						13

A seguir, apresenta-se o código máquina dos módulos ligados, de forma que os códigos objeto de cada um estão sobrepostos.

Códigos não ligados		Tabela Global		Código Máquina		
00. 12	VALS	L1:	06+ 0 = 06	12	22	1
02. 12	VALS + 1	L2:	14+ 0 = 14	12	23	2
04. 05	MOD2	RES:	15+ 0 = 15	05	16	3
06. 10	VALS	MOD2:	00+ 16 = 16	10	22	4
08. 04	VALS + 1	VALS:	06+ 16 = 22	04	23	5
10. 11	RES			11	15	6
12. 13	RES			13	15	7
14. 14				14		8
15. 00				00	fim do MOD1.o	9
16. 10	VALS + 1			10	23	10
18. 08	L2			08	14	11
20. 05	L1			05	06	12
22. 00	02			00	02 fim do MOD2.o	13
						14

1.2.8. Os códigos de cada módulo são apresentados a seguir. Para facilitar a ligação, é mostrado o código máquina não ligado de cada um.

Código (fator=0)		Tabela de Símbolos		Código Máquina		
00. MOD1:	BEGIN	MOD1:	00			1
						2

00.	MOD2:	EXTERN	MOD2:	00		3
00.	PUBLIC	N1	N1:	15		4
00.	PUBLIC	N2	N2:	16		5
00.	PUBLIC	N3	N3:	17		6
00.	PUBLIC	RETURN	RETURN:	08		7
	SECTION	TEXT				8
00.		INPUT	N1	Tabela de Usos	12 N1	9
02.		INPUT	N2	MOD2: 7+	12 N2	10
04.		INPUT	N3		12 N3	11
06.		JMP	MOD2	Tabela de Definições	05 MOD2	12
08.	RETURN:	INPUT	N1	MOD1: 00	12 N1	13
10.		LOAD	N1	N1: 15	10 N1	14
12.		JMPP	MOD1	N2: 16	07 MOD1	15
14.		STOP		N3: 17	14	16
	SECTION	DATA	RETURN:	08		17
15.	N1:	SPACE			00	18
16.	N2:	SPACE			00	19
17.	N3:	SPACE			00	20
	END					21

Código (fator=18)	Tabela de Símbolos	Código Máquina
00. MOD2: BEGIN	MOD2: 00	2
00. N1: EXTERN	N1: 00	3
00. N2: EXTERN	N2: 00	4
00. N3: EXTERN	N3: 00	5
00. RETURN: EXTERN	RETURN: 00	6
00. PUBLIC MOD2	N2_MAIOR_QUE_N1_E_N3: 18	7
	CASO_N3_N2: 30	8
00. LOAD N1	N1_MAIOR_QUE_N2: 36	10 N1 9
02. SUB N2	N1_MAIOR_QUE_N2_E_N3: 48	02 N2 10
04. JMPP N1_MAIOR_QUE_N2	CASO_N3_N1: 60	07 N1_MAIOR_QUE_N2 11
		12
06. LOAD N2	Tabela de Usos	10 N2 13
08. SUB N3	N1: 1+ 15+ 19+ 27+ 37+ 55+ 61+	02 N3 14
10. JMPP N2_MAIOR_QUE_N1_E_N3	N2: 3+ 7+ 25+ 31+ 45+ 49+ 57+	07 N2_MAIOR_QUE_N1_E_N3 15
	N3: 9+ 13+ 21+ 33+ 39+ 43+ 51+ 63+	16
12. OUTPUT N1	RETURN: 17+ 29+ 35+ 47+ 59+ 65+	13 N3 17
14. OUTPUT N3		13 N1 18
16. JMP RETURN	Tabela de Definições	05 RETURN 19
18. N2_MAIOR_QUE_N1_E_N3:	MOD2: 00	20
	N2_MAIOR_QUE_N1_E_N3: 18	10 N1 21
20. SUB N3	CASO_N3_N2: 30	02 N3 22
22. JMPP CASO_N3_N2	N1_MAIOR_QUE_N2: 36	07 CASO_N3_N2 23
	N1_MAIOR_QUE_N2_E_N3: 48	24
24. OUTPUT N1	CASO_N3_N1: 60	13 N1 25
26. OUTPUT N2		13 N2 26
28. JMP RETURN		05 RETURN 27
30. CASO_N3_N2:		28
	OUTPUT N3	13 N3 29
32. OUTPUT N2		13 N2 30

34. JMP	RETURN	05	RETURN	
36. N1_MAIOR_QUE_N2:				32
LOAD	N1	10	N1	33
38. SUB	N3	02	N3	34
40. JMPP	N1_MAIOR_QUE_N2_E_N3	07	N1_MAIOR_QUE_N2_E_N3	
;CASO_N2_N3:				36
42. OUTPUT	N2	13	N2	37
44. OUTPUT	N3	13	N3	38
46. JMP	RETURN	05	RETURN	
48. N1_MAIOR_QUE_N2_E_N3:				40
LOAD	N2	10	N2	41
50. SUB	N3	02	N3	42
52. JMPP	CASO_N3_N1	07	CASO_N3_N1	
;CASO_N2_N1:				44
54. OUTPUT	N2	13	N2	45
56. OUTPUT	N1	13	N1	46
58. JMP	RETURN	05	RETURN	
60. CASO_N3_N1:				48
OUTPUT	N3	13	N3	49
62. OUTPUT	N1	13	N1	50
64. JMP	RETURN	05	RETURN	
SECTION DATA				52
END				53

A seguir, apresenta-se o código máquina dos módulos ligados.

Códigos não ligados	Tabela Global	Código Máquina	
00. 12 N1	MOD1:	00+ 0 = 0	12 15
02. 12 N2	N1:	15+ 0 = 15	12 16
04. 12 N3	N2:	16+ 0 = 16	12 17
06. 05 MOD2	N3:	17+ 0 = 17	05 18
08. 12 N1	RETURN:	08+ 0 = 08	12 15
10. 10 N1	MOD2:	00+ 18 = 18	10 15
12. 07 MOD1	N2_MAIOR_QUE_N1_E_N3:	18+ 18 = 36	07 00
14. 14	CASO_N3_N2:	30+ 18 = 48	14
15. 00	N1_MAIOR_QUE_N2:	36+ 18 = 54	00
16. 00	N1_MAIOR_QUE_N2_E_N3:	48+ 18 = 66	00
17. 00	CASO_N3_N1:	60+ 18 = 78	00
18. 10 N1			10 15
20. 02 N2			02 16
22. 07 N1_MAIOR_QUE_N2			07 54
24. 10 N2			10 16
26. 02 N3			02 17
28. 07 N2_MAIOR_QUE_N1_E_N3			07 36
30. 13 N1			13 15
32. 13 N3			13 17
34. 05 RETURN			05 08
36. 10 N1			10 15
38. 02 N3			02 17
40. 07 CASO_N2_N3			07 48
42. 13 N1			13 15
44. 13 N2			13 16

46.	05	RETURN	05	08	27
48.	13	N3	13	17	28
50.	13	N2	13	16	29
52.	05	RETURN	05	08	30
54.	10	N1	10	15	31
56.	02	N3	02	17	32
58.	07	N1_MAIOR_QUE_N2_E_N3	07	66	33
60.	13	N2	13	16	34
62.	13	N3	13	17	35
64.	05	RETURN	05	08	36
66.	10	N2	10	16	37
68.	02	N3	02	17	38
70.	07	CASO_N1_N3	07	78	39
72.	13	N2	13	16	40
74.	13	N1	13	15	41
76.	05	RETURN	05	08	42
78.	13	N3	13	17	43
80.	13	N1	13	15	44
82.	05	RETURN	05	08	45

1.2.9.

## **Módulo 2 – Assembly x86-64**

### **2.1 Questões Teóricas**

2.1.1.

## 2.2 Questões Práticas

2.2.1.

```

SIZE EQU 6
section .data
little dd 42434445h, 45454545h, 4A4B4C4Dh,
        dd 414D4E4Fh, 46454948h, 4C474D46h

section .bss
big resd SIZE
temp resd 1

section .start
global _start
_start:
    mov ecx, SIZE
    mov eax, little
    mov esi, big
laco1: mov ebx, esi
    add ebx, 3          ; ebx aponta para o último byte da dword big endian
laco2: mov dl, [eax]
    mov [ebx], dl
    dec ebx
    inc eax
    cmp ebx, esi        ; 4 bytes foram preenchidos? se não, repete
    jae laco2
    add esi, 4
    dec ecx              ; mais um número convertido
    cmp ecx, 0
    ja laco1             ; tem mais número? se sim, repete
done:  mov eax, 1
    mov ebx, 0
    int 80h

```

2.2.2.

(a)

```

section .data
MAX equ 100
section .bss
a resd MAX
section .text
global _start
_start:
sub esi, esi      ; i=0
for:
    cmp esi, MAX
    jae end_for
    mov eax, esi
    shr eax, 1      ; eax = i >> 1
    mov DWORD [a + 4*esi], eax ; a[i] = i >> i
    inc esi
    jmp for
end_for

```



```

end_for:
mov eax, 1
mov ebx, 0
int 80h

```

(b)

```

section .data
ROW equ 5
COL equ 5
array1 dd 1, 89, 99, 91, 92,
        dd 79, 2, 70, 60, 55,
        dd 70, 60, 3, 90, 89,
        dd 60, 55, 68, 4, 66,
        dd 51, 59, 57, 2, 5
array2 TIMES ROW dd 1, 2, 3, 4, 5
section .bss
array3 TIMES ROW resb COL
section .text
global _start
_start:
    mov esi, 0          ; i=0
    for_i:
        cmp esi, ROW
        jae end_for_i
        mov edi, 0
        for_j:
            cmp edi, COL
            jae end_for_j
            imul eax, esi, ROW ; eax = offset da linha em elementos
            mov ebx, edi
            shl ebx, 2         ; ebx = offset da coluna em bytes
            mov ecx, DWORD [array1 + 4*eax + ebx]
            cmp ecx, DWORD [array2 + 4*eax + ebx] ; array1 == array2 ?
            je set_1
            mov BYTE [array3 + eax + edi], "0"
            jmp continue
        set_1:
            mov BYTE [array3 + eax + edi], "1"
        continue:
            ; printf("%c", array3[i][j])
            mov ecx, eax
            mov eax, 4
            mov ebx, 1
            add ecx, array3
            add ecx, edi
            mov edx, 1
            int 80h
            inc edi          ; j++
            jmp for_j
        end_for_j:
            inc esi
            jmp for_i

```

```

end_for_i:
mov eax, 1
mov ebx, 0
int 80h

```

(c)

```

section .data
SIZE equ 11
vetor dd 0x10002231, 0x80154491, 0x91929394,
        dd 0x11223344, 0x12131415, 0x79270601,
        dd 0x55127380, 0x16112212, 0x39089607,
        dd 0x51557721, 0x16846676
section .text
global _start
_start:
sub eax, eax ; res=0
mov ecx, SIZE ; i=SIZE
while:
    ; res += vetor[i++]
    add eax, DWORD [vetor + 4*ecx - 4]
    loop while
mov eax, 1
mov ebx, 0
int 80h

```

(d)

```

#include "io.mac"
section .data
MAX equ 100
section .bss
a TIMES MAX resw MAX
section .text
global _start
_start:
mov esi, 0 ; i=0
for_i:
    cmp esi, MAX
    jae end_for_i
    mov edi, 0 ; j=0
    for_j:
        cmp edi, MAX
        jae end_for_j
        mov ecx, esi ; ecx = i
        cmp esi, edi
        je set_as_3i ; i==j?
        shl ecx, 3 ; ecx = 8*i
        sub ecx, esi ; ecx = 7*i
        jmp continue
set_as_3i:
    shl ecx, 2 ; ecx = 4*i
    sub ecx, esi ; ecx = 3*i
continue:

```

```

    imul eax, esi, MAX
    imul ebx, edi, 2
    mov WORD [a + 2*eax + ebx], cx ; atualiza matriz
    inc edi
    jmp for_j
end_for_j:
    inc esi
    jmp for_i
end_for_i: ; return 0
mov eax, 1
mov ebx, 0
int 80h

```

(e)

```

#include "io.mac"
section .bss
count resd 1
start resb 1
section .text
global _start
_start:
sub esi, esi ; sum=0
mov DWORD [count], 100 ; count=100
; lê um caracter (dígito) do usuário
mov eax, 3
mov ebx, 0
mov ecx, start
mov edx, 1
int 80h
; converte para número
sub BYTE [start], "0"
while:
    mov eax, esi ; eax = sum
    shr eax, 1 ; eax = sum // 2
    shl eax, 1 ; eax = (sum // 2) * 2
    sub eax, esi ; eax = -(sum % 2)
    cmp eax, 0 ; sum%2 == 0 ?
    je sub_start
    add esi, DWORD [start] ; sum += start
    jmp continue
sub_start:
    sub esi, DWORD [start] ; sum -= start
continue:
    inc BYTE [start] ; start++
    dec DWORD [count] ; count--
    cmp DWORD [count], 0
    ja while
; return sum
mov eax, 1
mov ebx, esi
int 80h

```

## 2.2.3.

- (a) `int foo1(int n) {  
    return 7*n;  
}` 1  
2  
3
- (b) `int foo2 (int n) {  
    return n / 2147483648; // n / 2^31  
}` 1  
2  
3
- (c) `int foo3 (int *p) {  
    return *p + *p;  
}` 1  
2  
3
- (d) `short int foo4 (short int x, short int y) {  
    return x - y; // C é right-pusher  
}` 1  
2  
3

2.2.4. `f4:` 1  
`enter 0, 0` 2  
`mov ebx, DWORD [ebp + 8]` ; ebx recebe o ponteiro/vetor de shorts x 3  
`mov ecx, DWORD [ebp + 12]` ; ecx recebe o número de elementos n 4  
`mov ax, WORD 1` 5  
`for_loop:` 6  
`cmp ecx, 0` 7  
`jle end_f4` 8  
`cwd` ; estende o sinal de ax em eax 9  
`cdq` ; estende o sinal de eax em edx 10  
`mul WORD [ebx]` ; dx.ax = ax \* elemento 11  
`dec ecx` ; um elemento a menos a ser multiplicado 12  
`add ebx, 2` ; ebx aponta para o próximo elemento 13  
`jmp for_loop` 14  
`end_f4:` 15  
`shl edx, 16` ; edx = dx.000... 16  
`shl eax, 16` ; eax = ax.000... 17  
`shr eax, 16` ; eax = ...000.ax 18  
`add eax, edx` ; eax = dx.ax 19  
`leave` 20  
`ret` 21

2.2.5. `f4:` 1  
`enter 4, 0` ; variável local: DWORD para o resultado 2  
`mov esi, DWORD [ebp + 8]` ; esi recebe o ponteiro/vetor de shorts x 3  
`mov edi, DWORD [ebp + 12]` ; edi recebe o ponteiro/vetor de shorts y 4  
`mov ecx, DWORD [ebp + 16]` ; ecx recebe o número de elementos n 5  
`for_loop:` 6  
`cmp ecx, 0` 7  
`jle end_f4` 8  
`mov ax, WORD [esi]` 9

```

    cwd                ; estende o sinal de ax em eax      10
    cdq                ; estende o sinal de eax em edx      11
    mul WORD [edi]      ; dx.ax = short_x * short_y        12
    shl edx, 16         ; edx = dx.000...                 13
    shl eax, 16         ; eax = ax.000...                 14
    shr eax, 16         ; eax = ...000.ax                 15
    add eax, edx        ; eax = dx.ax                     16
    add DWORD [ebp - 4], eax ; atualiza montante           17
    dec ecx             ; um par a menos a ser multiplicado 18
    add esi, 2          ; esi aponta para o próximo elemento 19
    add edi, 2          ; edi aponta para o próximo elemento 20
    jmp for_loop        21
end_f4:                22
mov eax, DWORD [ebp - 4] ; eax recebe o resultado         23
leave                 24
ret                   25

```

- 2.2.6. Modificações necessárias no código C original: além de eliminar a definição original da função, deve também declarar a assinatura da soma em Assembly por `extern void soma(int *M, int N, int *valor)`.

```

soma:
enter 0, 0
mov esi, DWORD [ebp + 8] ; esi = ponteiro de inteiros/matriz
mov ecx, DWORD [ebp + 12] ; ecx = contador
mov edi, 0               ; edi = offset da matriz
mov eax, 0               ; eax = resultado da soma
do_while:
    add eax, DWORD [esi + 4*edi] ; incrementa montante
    add edi, DWORD [ebp + 12]    ; desce um "linha"
    inc edi                     ; avança uma "coluna"
    loop do_while               ; --ecx > 0 ? repete
mov ecx, DWORD [ebp + 16] ; ecx = ponteiro de saída
mov DWORD [ecx], eax      ; resultado da saída = montante
leave
ret

```

```

2.2.7. %include "io.mac"
f1:
enter 0, 0
mov esi, DWORD [ebp + 8] ; esi recebe a matriz 1/ponteiro de int 1
mov ebx, 0               ; ebx = offset da linha em elementos (0,m,...)
mov ecx, 0               ; ecx = contador c
for_c:
    cmp ecx, DWORD [ebp + 20]
    jae end_f1           ; c >= n ? fim da função
    mov edx, 0           ; edx = contador d
    for_d:
        cmp edx, DWORD [ebp + 16]
        jae end_for_d    ; d >= m ? fim do laço
        mov eax, ebx      ; eax = c*sizeof(int)
        add eax, edx      ; eax = c*sizeof(int) + d

```

```

        ; printf("%d\t", (matrix + c*sizeof(int) + d))
        PutLInt DWORD [esi + 4*eax]
        PutCh 9 ; ascii 9 = \t
        inc edx ; d++
        jmp for_d
end_for_d:
nwnln
add ebx, edx ; ebx = c*sizeof(int)
inc ecx ; c++
jmp for_c
end_f1:
leave
ret

```

2.2.8. `%include "io.mac"`

```

section .data
    BUF_SIZE equ 256
    type_entry_name db "Digite o nome do arquivo de entrada: "
    type_output_name db "Digite o nome do arquivo de saída: "
section .bss
    fd1 resd 1
    fd2 resd 1
    file_in resb 30
    file_out resb 30
    buf resb BUF_SIZE
section .text
global _start
_start:
    ; printf/scanf
    PutStr type_entry_name
    GetStr file_in
    PutStr type_output_name
    GetStr file_out
    ; fd1 = fopen(file_in, "r")
    mov eax, 5
    mov ebx, file_in
    mov ecx, 00 ; modo leitura
    mov edx, 777 ; permissão completa a todos
    int 80h
    mov DWORD [fd1], eax ; fd1 = file descriptor da entrada
    ; fd2 = fopen(file_out, "w")
    mov eax, 5 ; syscall open file
    mov ebx, file_out
    mov ecx, 01 ; modo escrita
    mov edx, 777 ; permissão completa a todos
    int 80h
    mov DWORD [fd2], eax ; fd2 = file descriptor da saída
    ; fread(buf, sizeof(char), BUF_SIZE, fd1)
    mov eax, 3
    mov ebx, DWORD [fd1]
    mov ecx, buf

```

```

mov edx, BUF_SIZE
int 80h
; fwrite(buf, sizeof(char), BUF_SIZE, fd2)
mov eax, 4
mov ebx, DWORD [fd2]
mov ecx, buf
mov edx, BUF_SIZE
int 80h
; fclose(fd1)
mov eax, 6
mov ebx, DWORD [fd1]
int 80h
; fclose(fd2)
mov eax, 6
mov ebx, DWORD [fd2]
int 80h
; return 0
mov eax, 1
mov ebx, 0
int 80h

```

2.2.9.

```

section .data
file_in    db "myfile1.txt"
file_out   db "myfile2.txt"
n          equ 100
section .bss
x          resb n
soma       resd 1
section .text
global _start
_start:
; abre arquivo de entrada
mov eax, 5
mov ebx, file_in
mov ecx, 00
mov edx, 777          ; permissão total a todos
int 80h
; lê arquivo de entrada, preenchendo x
mov ebx, eax          ; ebx = file descriptor da entrada
mov eax, 3
mov ecx, x
mov edx, n
int 80h
; fecha o arquivo
mov eax, 6
int 80h
; laço for para somar os elementos
mov esi, 0
mov eax, 0
for_x:
    cmp esi, n

```

```

    jae end_for_x
    mov al, BYTE [x + esi]
    movsx eax, al
    add DWORD [soma], eax
    inc esi
    jmp for_x
end_for_x:
; abre arquivo de saída
mov eax, 5
mov ebx, file_out
mov ecx, 01
mov edx, 700      ; permissão total ao dono, nada ao resto
int 80h
; escreve a soma
mov ebx, eax      ; ebx = file descriptor da saída
mov eax, 4
mov ecx, soma
mov edx, 4        ; soma é inteiro => 4 bytes
int 80h
; fecha o arquivo
mov eax, 6
int 80h
; fim do programa
mov eax, 1
mov ebx, 0
int 80h

```

2.2.10. Note que os arrays/buffers *x* e *y* têm 200 bytes de conteúdo, e não 100.

```

section .data
    file_in  db "myfile1.txt"
    file_out db "myfile2.txt"
    BUF_SIZE equ 100
section .bss
    x  resw BUF_SIZE
    y  resw BUF_SIZE
section .text
global _start
_start:
; abre arquivo de entrada
mov eax, 5
mov ebx, file_in
mov ecx, 00
mov edx, 777      ; permissão total a todos
int 80h
; lê arquivo de entrada, preenchendo x
mov ebx, eax      ; ebx = file descriptor da entrada
mov eax, 3
mov ecx, x
mov edx, BUF_SIZE
add edx, edx
int 80h

```



```

; fecha o arquivo
mov eax, 6
int 80h
; laço for para preencher y
mov esi, 0
for_y:
    cmp esi, BUF_SIZE
    jae end_for_y
    cmp WORD [x + 2*esi], 0
    ja write_1
    mov WORD [y + 2*esi], 0
    jmp continue
write_1:
    mov WORD [y + 2*esi], 1
continue:
    inc esi
    jmp for_y
end_for_y:
; abre arquivo de saída
mov eax, 5
mov ebx, file_out
mov ecx, 01
mov edx, 744 ; permissão total ao dono, leitura ao resto
int 80h
; escreve o array y
mov ebx, eax ; ebx = file descriptor da saída
mov eax, 4
mov ecx, y
mov edx, BUF_SIZE
add edx, edx
int 80h
; fecha o arquivo
mov eax, 6
int 80h
; fim do programa
mov eax, 1
mov ebx, 0
int 80h

```

2.2.11. O programa a seguir multiplica matrizes de tamanhos arbitrários e compatíveis. O procedimento auxiliar `GetMat` realiza os laços de preenchimento das matrizes.

```

#include "io.mac"
section .data
    ROWS1 equ 5
    COLS1 equ 5
    ROWS2 equ COLS1
    COLS2 equ 5
section .bss
    mat1 TIMES ROWS1 resd COLS1
    mat2 TIMES ROWS2 resd COLS2
    mat3 TIMES ROWS1 resd COLS2

```

```

section .text
global _start
_start:
; preenche a matriz do lado esquerdo do produto
push ROWS1
push COLS1
push mat1
call GetMat
add esp, 12 ; esp restaurado
; preenche a matriz do lado direito do produto
push COLS2
push ROWS2
push mat2
call GetMat
add esp, 12 ; esp restaurado
; realiza a operação mat1 * mat2 = mat3
; mat1 -> m x l = ROWS1 x COLS1
; mat2 -> l x n = ROWS2 x COLS2
; mat3 -> m x n = ROWS1 x COLS2
mov esi, 0
for_i:
    cmp esi, ROWS1
    jae end_prod ; 0 <= i < m
    mov edi, 0
    for_j:
        cmp edi, COLS2
        jae end_for_j ; 0 <= j < n
        imul eax, esi, COLS2
        add eax, edi ; offset3 = n*i+j = COLS2*esi+edi
        mov DWORD [mat3 + 4*eax], 0
        mov ecx, 0
        for_k:
            cmp ecx, ROWS2
            jae end_for_k ; 0 <= k < l
            imul eax, esi, COLS1
            add eax, ecx ; offset1 = l*i+k = COLS1*esi+ecx
            mov ebx, DWORD [mat1 + 4*eax]
            imul eax, ecx, COLS2
            add eax, edi ; offset2 = n*k+j = COLS2*ecx+edi
            imul ebx, DWORD [mat2 + 4*eax] ; m1[i][k] * m2[k][j]
            imul eax, esi, COLS2
            add eax, edi ; offset3 = n*i+j = COLS2*esi+edi
            add DWORD [mat3 + 4*eax], ebx ; m3[i][j] += m1[i][k] * m2[k][j]
            inc ecx
            jmp for_k
        end_for_k:
            inc edi ; j++
            jmp for_j ; próxima coluna da m2
    end_for_j:
        inc esi ; i++
        jmp for_i ; próxima linha da m1

```

```

end_prod:
; fim do programa
mov eax, 1
mov ebx, 0
int 80h

GetMat:
enter 0, 0
mov esi, DWORD [ebp + 8]      ; esi recebe a matriz 1/ponteiro de int
mov ebx, 0                    ; ebx = offset da linha em elementos
mov ecx, 0                    ; ecx = contador i
row_for:
    cmp ecx, DWORD [ebp + 16]
    jae end_GetMat            ; i >= ROWS ? fim da função
    mov edx, 0                ; edx = contador j
    column_for:
        cmp edx, DWORD [ebp + 12]
        jae end_column_for    ; j >= COLS ? fim do laço
        mov eax, ebx           ; eax = i*sizeof(int)
        add eax, edx           ; eax = i*sizeof(int) + j
        GetLInt edi
        mov DWORD [esi + 4*eax], edi
        inc edx                ; j++
        jmp column_for
    end_column_for:
        add ebx, edx           ; ebx = i*sizeof(int)
        inc ecx                ; i++
        jmp row_for
end_GetMat:
leave
ret

```

2.2.12. Bias = 011 = 3, Regra = *ceil*.

Descrição	Binário	Mantissa	Expoente	Valor decimal
Menos zero	1 000 00	0.0	-2	-0.0
Número positivo mais próximo a zero	0 000 01	$1/4$	-2	$1/4 \times 2^{-2}$
Infinito negativo	1 111 00	-	-	-
Maior número normalizado	0 110 11	$13/4$	3	$13/4 \times 2^3$
Menor número não-normalizado	1 000 11	$3/4$	-2	$-3/4 \times 2^{-2}$
$5.0 - 0.75 = 4.25$	0 101 01	$11/4$	2	$11/4 \times 2^2 = 5.0$
$4.0 + 3.0 = 7.0$	0 101 11	$13/4$	2	$13/4 \times 2^2 = 7.0$

2.2.13. Bias = 0111 = 7, Regra = *round*

Descrição	Binário	Mantissa	Expoente	Valor decimal
Menos zero	1 0000 00	0	-2	-0.0
Número positivo mais próximo a zero	0 0000 01	$1/4$	-6	$1/4 \times 2^{-6}$
Maior número normalizado	0 1110 11	$13/4$	7	$13/4 \times 2^7$
Menor número não-normalizado	1 0000 11	$3/4$	-6	$-3/4 \times 2^{-6}$
$4.0 + 3.0 = 7.0$	0 1001 11	$13/4$	2	$13/4 \times 2^2 = 7.0$
$7.0 + 8.0 = 15.0$	0 1010 11	$13/4$	3	$13/4 \times 2^3 = 14.0$

2.2.14. Bias = 0111 = 7, Regra = **fração mais próxima**.

Número	Valor	Bit sinal	Bits expoente	Bits mantissa
Zero	0.0	0	0000	0000
Negativo mais próximo a zero	$-1/16 \times 2^{-6}$	1	0000	0001
Maior positivo	$1^{15}/16 \times 2^7$	0	1110	1111
n/a	-5.0	1	1001	0100
n/a	$1^9/16 \times 2^{-2}$	0	0101	1001
Menos um	-1.0	1	0111	0000
$4 - 1^9/16 = 3^9/16 = 2.4375$	$4^0/16 = 2.5$	0	1000	0100

2.2.15. Bias = 01111 = 15, Regra = **par mais próximo**.

Descrição	Binário	Mantissa	Expoente	Valor decimal
Número negativo mais próximo a zero	1 00000 0001	$1/16$	-14	$-1/16 \times 2^{-14}$
Maior número	0 11110 1111	$1^{15}/16$	15	$1^{15}/16 \times 2^{15}$
Menor número não-normalizado	1 00000 1111	$1^5/16$	-14	$1^5/16 \times 2^{-14}$
Menos um	1 01111 0000	1.0	0	-1.0