Módulo 1 – Compiladores

1.1 Questões Teóricas

1.2 Questões Práticas

Módulo 2 – Assembly x86-64

2.1 Questões Teóricas

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.

2.2.3.

(a)
```
int foo1(int n) {
    return 7*n;
}
```

(b)
```
int foo2 (int n) {
    return n / 2147483648;  // n / 2^31
}
```

(c)
```
int foo3 (int *p) {
    return *p + *p;
}
```

(d)
```c
short int foo4 (short int x, short int y) {
    return x - y;   // C é right-pusher
}
```

2.2.4.
```asm
f4:
enter 0, 0
mov ebx, DWORD [ebp + 8]     ; ebx recebe o ponteiro/vetor de shorts x
mov ecx, DWORD [ebp + 12]    ; ecx recebe o número de elementos n
mov ax, WORD 1
for_loop:
    cmp ecx, 0
    jle end_f4
    cwd                      ; estende o sinal de ax em eax
    cdq                      ; estende o sinal de eax em edx
    mul WORD [ebx]           ; dx.ax = ax * elemento
    dec ecx                  ; um elemento a menos a ser multiplicado
    add ebx, 2               ; ebx aponta para o próximo elemento
    jmp for_loop
end_f4:
shl edx, 16                  ; edx = dx.000...
shl eax, 16                  ; eax = ax.000...
shr eax, 16                  ; eax = ...000.ax
add eax, edx                 ; eax = dx.ax
leave
ret
```

2.2.5.
```asm
f4:
enter 4, 0                   ; variável local: DWORD para o resultado
mov esi, DWORD [ebp + 8]     ; esi recebe o ponteiro/vetor de shorts x
mov edi, DWORD [ebp + 12]    ; edi recebe o ponteiro/vetor de shorts y
mov ecx, DWORD [ebp + 16]    ; ecx recebe o número de elementos n
for_loop:
    cmp ecx, 0
    jle end_f4
    mov ax, WORD [esi]
    cwd                          ; estende o sinal de ax em eax
    cdq                          ; estende o sinal de eax em edx
    mul WORD [edi]               ; dx.ax = short_x * short_y
    shl edx, 16                  ; edx = dx.000...
    shl eax, 16                  ; eax = ax.000...
    shr eax, 16                  ; eax = ...000.ax
    add eax, edx                 ; eax = dx.ax
    add DWORD [ebp - 4], eax     ; atualiza montante
    dec ecx                      ; um par a menos a ser multiplicado
    add esi, 2                   ; esi aponta para o próximo elemento
    add edi, 2                   ; edi aponta para o próximo elemento
    jmp for_loop
end_f4:
mov eax, DWORD [ebp - 4]         ; eax recebe o resultado
leave
```

```
ret
```

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.
2.2.8.
2.2.9.
2.2.10.
2.2.11.
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	$1\,3/4$	3	$1\,3/4 \times 2^{3}$
Menor número não-normalizado	1 000 11	$3/4$	-2	$-3/4 \times 2^{-2}$
$5.0 - 0.75 = 4.25$	0 101 01	$1\,1/4$	2	$1\,1/4 \times 2^{2} = 5.0$
$4.0 + 3.0 = 7.0$	0 101 11	$1\,3/4$	2	$1\,3/4 \times 2^{2} = 7.0$

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	$1\,3/4$	7	$1\,3/4 \times 2^{7}$
Menor número não-normalizado	1 0000 11	$3/4$	-6	$-3/4 \times 2^{-6}$
$4.0 + 3.0 = 7.0$	0 1001 11	$1\,3/4$	2	$1\,3/4 \times 2^{2} = 7.0$
$7.0 + 8.0 = 15.0$	0 1010 11	$1\,3/4$	3	$1\,3/4 \times 2^{3} = 14.0$

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
$4 - 1\frac{9}{16} = {}^{39}/16 = 2.4375$	${}^{40}/16 = 2.5$	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	$15/16$	-14	$15/16 \times 2^{-14}$
Menos um	1 01111 0000	1.0	0	-1.0