Parte 1 – Exercícios de Software Básico

1.1 Módulo 1 – Compiladores

Questões Teóricas

1.1.1. Abaixo estão listadas várias afirmativas incorretas. Justificando, identifique os erros, e corrija-os.

(a) O formato .COM caracteriza-se por ter um endereço fixo (100H) para o ponto de entrada do programa. O cabeçalho de um arquivo nesse formato possui tamanho reduzido simplesmente composto pelos caracteres 'MZ' e a quantidade de segmentos de 64 kiB necessários para esse programa.

(b) Bootstrap loader é um carregador especial, já que ele consiste em um programa armazenado completamente em um único setor conhecido como MBR.

(c) A utilização de bibliotecas dinâmicas permite que um trecho de código chamado por vários problemas possa ter uma única cópia em memória, e somente carregada ao ser executada: o montador e o ligador portanto não necessitam serem informados sobre o uso de uma biblioteca dinâmica.

(d) O formato ELF é um formato de exclusivo de arquivos objeto complexo que armazena tabelas com informações de realocação para o ligador. O formato PE (portable executable) é um formato feito em base no formato ELF.

1.1.2. Abaixo estão listadas várias afirmativas, as quais podem estar erradas. Identifique o(s) erro(s), justifique o por quê, e corrija-o.

(a) Executável ligado com biblioteca estática possui a vantagem de ser mais portável que o executável com biblioteca dinâmica. O programa também é carregado mais rápido em memória. Porém, é possível ter várias cópias da mesma biblioteca em memória. Por esse motivo os formatos de arquivos mais recentes como ELF e PE não permitem ligação estática.

(b) A utilização de bibliotecas dinâmicas com carregador estático permite que um trecho de código chamado por vários programas possa ter uma única cópia em memória, e somente carregada ao ser executada. O montador e o ligador portanto não necessitam serem informados sobre o uso de uma biblioteca dinâmica.

1.1.3. Descreva as informações contidas no MBR, indicando suas partes. Indique qual é o objetivo do setor MBR no processo de carregação do Sistema Operacional. Indique onde se encontra o MBR. Descreva por que o MBR foi substituído pelo GPT em sistemas computacionais modernos.

Questões Práticas

1.1.4. Considere os módulos a seguir na Linguagem de Montagem Hipotética apresentada em sala.

Listing 1.1.1: Código fonte do módulo A.

```
MOD_A:  BEGIN
Y:      EXTERN
MOD_B:  EXTERN
        PUBLIC VAL
        PUBLIC L1
        INPUT Y
        LOAD VAL
        ADD Y
        STORE Y + 2
        JMPP MOD_B
L1:     STOP
VAL:    CONST 5
END
```

Listing 1.1.2: Código fonte do módulo B.

```
MOD_B:  BEGIN
VAL:    EXTERN
L1:     EXTERN
        PUBLIC Y
        PUBLIC MOD_B
        OUTPUT Y
        OUTPUT VAL
        OUTPUT Y + 2
        JMP L1
Y:      SPACE 3
END
```

(a) Personifique um montador e monte os módulos como uma sequência de números inteiros, utilizando a tabela em apêndice como referência. Apresenta o código montado e as tabelas resultantes.
(b) Personifique um ligador e combine os módulos em um único arquivo executável. Apresente o código ligado indicando os endereços absolutos e relativos, escrevendo no cabeçalho antes do código um mapa de bits mediante a diretiva R (ex.: R 0101010001).

1.2 Módulo 2 – Assembly x86-64

Questões Teóricas

1.2.1. Os itens abaixo possuem instruções de programas Assembly IA-32 (em modo nativo) que utilizam diversos modos de endereçamento. Classifique cada item como correto ou errado, e justifique o que estiver errado.

(a) `mov EAX, 10`
(b) `mov [M], AL`
(c) `mov AL, [CS + ESI + array]`
(d) `mov vetor[1], 0`
(e) `add AX, [X + ECX]`
(f) `mov ESI, vetor + EBX`
(g) `inc WORD [inicio + EBX*8 + ESI]`
(h) `mov [EBX + ESI*4], DWORD 5`
(i) `dec BYTE [BL]`
(j) `add [x + 1], AL`
(k) `mov EAX, [array + ECX*8 + EBX]`
(l) `mov [EAX*8 + 1], 5`
(m) `mov BL, AX`
(n) `cmp [ESI], 10`
(o) `adc AL, AH`

1.2.2. Descreva as diferenças sobre endereçamento e utilização de meória do Modo Real e o Modo Protegido. Para o modo real, indique somente como é calculado o endereço de memória. No modo protegido, faça um diagrama mostrando os diferentes segmentos de memória, indique como é calculado o endereço lógico e real, e por que o modo é chamado de protegido.

1.2.3. Dadas as seguintes instruções e os seus respectivos códigos de máquina, indique os valores dos campos OPCODE, Mod R/M, SIB, DISPLACEMENT, e IMMEDIATE. Note que uma instrução pode deixar de apresentar algum campo.

- `MOV edx, 0x0` ba 00 00 00 00
- `MOV EBP, ESP` 89 e5

1.2.4. Existem três tipos básicos de operandos: imediato, registrador e memória. O acesso a memória pode ser feito de duas maneiras: direta ou indireta. Em cada instrução com algum tipo de endereçamento do código abaixo especifique que tipo de operando está sendo usado como fonte e destino: imediato, memória direta/indireta, ou registrador. Indicar se algum endereçamento é ilegal.

Listing 1.2.1: Código com vários tipos de acesso a memória.

```
section .data
count db 2
wordList dw 0003h, 2000h
array db 0Ah, 0Bh, 0Ch, 0Dh

section .text
global _start
_start:
    mov esi, wordList
    mov ax, [esi]
    mov bx, ax
    mov al, [array + bx]
    add al, [count]
    mov ax, 40
    sub ax, [wordList + 2]
    mov [count], 50
    mov eax, 1
    mov ebx, 0
    int 80h
```

Questões Práticas

1.2.5. O programa abaixo realiza a cópia de um vetor de *double words*, convertendo-o de *little endian* para *big endian*. Complete o programa, indicando as instruções dos espaços em branco (cada espaço deve ser preenchido com uma única instrução).

Listing 1.2.2: Conversor de *little endian* para *big endian*.

```
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
------
laco2: mov dl, [eax]
mov [ebx], dl
dec ebx
inc eax
------
jae laco2
add esi, 4
------
cmp ecx, 0
------
done: mov eax, 1
mov ebx, 0
int 80h
```

1.2.6. Para cada código C abaixo, escreva o equivalente em Assembly IA-32. Diretivas em C **devem** ser substituídas por diretivas equivalentes em IA-32. Use os registradores para as variáveis locais (com exceção de estruturas de dados) e seção de Dados ou BSS para as variáveis estáticas ou globais. **Deve-se** utilizar os endereçamentos corretos para cada tipo de estrutura de dados. Não se preocupe pelo fato do programa principal em C ser uma função.

Listing 1.2.3: Vetor de 100 inteiros.

(a)
```
#define MAX 100
int main (void) {
    int a[100], i;
    for(i=0; i<MAX; i++) a[i] = i>>1;
    return 0;
}
```

Listing 1.2.4: Matriz indicadora de inteiros iguais.

(b)
```c
#include <stdio.h>
#define ROW 5
#define COL 5
int main(void) {
    int array1[ROW][COL] = {
        {1, 89, 99, 91, 92},
        {79, 2, 70, 60, 55},
        {70, 60, 3, 90, 89},
        {60, 55, 68, 4, 66},
        {51, 59, 57, 2, 5}
    };
    int array2[ROW][COL] = {
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5},
        {1, 2, 3, 4, 5}
    };
    int array3[ROW][COL];
    int i, j;
    for(i=0; i<ROW; i++)
    for(j=0; j<COL; j++) {
        if (array1[i][j] == array2[i][j]) array3[i][j] = '1';
        else array3[i][j] = '0';
        printf("%c", array3[i][j]);
    }
    return 0;
}
```

Listing 1.2.5: Soma dos elementos de um vetor.

(c)
```c
#define SIZE 11
int main(void){
    int vetor[SIZE] = {
        0x10002231, 0x80154491, 0x91929394,
        0x11223344, 0x12131415, 0x79270601,
        0x55127380, 0x16112212, 0x39089607,
        0x51557721, 0x16846676
    };
    int res=0;
    int i=0;
    while (i<SIZE)
        res += vetor[i++];
    return 0;
}
```

(d) Não é permitido `MUL` ou `IMUL`.

Listing 1.2.6: Preenchimento de matriz 100x100.

```
#define MAX 100
int main(void){
    short int a[MAX][MAX];
    int i, j;
    for(i=0; i<MAX; i++)
    for(j=0; j<MAX; j++){
        if (i==j) a[i][j] = 3*i;
        else a[i][j] = 7*i;
    }
    return 0;
}
```

(e) Assuma que o usuário vai digitar um número de 0 a 9.

Listing 1.2.7: Soma com parâmetro na entrada do usuário.

```
char Start;
int Count;

int main() {
    char sum=0;
    Count = 100;
    scanf("%d", &Start);

    while(Count) {
        if (sum%2)
            sum += Start;
        else
            sum -= Start;
        Start++;
        Count--;
    }
    return sum;
}
```