Lista de códigos

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.

```
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
```

Código 1.1.1: Código fonte do módulo A.

```
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
```

Código 1.1.2: Código fonte do módulo B.

(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.

(a) `MOV edx, 0x0` ba 00 00 00 00
(b) `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.

```
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
```

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

1.2.5. Sobre a arquitetura x64, responda:

(a) O que significa um processador ser de arquitetura híbrida RISC/CISC?
(b) Descreva como é feito o endereçamento de memória na arquitetura x64, indicando os grupos de bits dentro do endereçamento virtual.
(c) Explique brevemente o que é a tecnologia SIMD, indicando os registradores envolvidos.
(d) Indique as diferenças entre os registradores de uso geral da arquitetura IA-32 e x64.

1.2.6. Uma instrução de pulo (ou salto) pode ser classificada de diversas formas. Responda sucintamente às perguntas abaixo com relação a pulos.

(a) O que significa um pulo curto relativo? Como é calculado o valor no contador de programa após executar a instrução de pulo?
(b) O que significa um pulo distante absoluto indireto? Como é calculado o valor no contador de programa após executar a instrução de pulo?
(c) Quais os tipos de pulos distantes no Protected Mode e Real Mode?

Questões Práticas

1.2.7. 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).

```
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
```

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

1.2.8. 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.

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

Código 1.2.3: Vetor de 100 inteiros.

(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;
}
```

Código 1.2.4: Matriz indicadora de inteiros iguais.

(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;
}
```

Código 1.2.5: Soma dos elementos de um vetor.

(d) Não é permitido `MUL` ou `IMUL`.

```c
#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;
}
```

Código 1.2.6: Preenchimento de matriz 100x100.

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

```c
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;
}
```

Código 1.2.7: Soma com parâmetro na entrada do usuário.

1.2.9. Escreva uma versão em C de cada uma das funções em Assembly IA-32 (todas as funções em Assembly colocam o valor de retorno em EAX) seguindo as seguintes regras:

(i) todas as funções são compostas por uma **única instrução RETURN** sem criar variáveis locais,
(ii) em nenhuma função em C pode ser utilizado deslocamento de bits ($\ll$ ou $\gg$) e
(iii) somente pode ser feita **uma única operação aritmética e/ou uma única comparação** ($>$ ou $<$) nas funções em C.

(a)
```asm
foo1:
push ebp
mov ebp, esp
mov edx, [ebp + 8]
mov eax, edx
shl eax, 3
sub eax, edx
pop ebp
ret
```

Código 1.2.8: foo1.

(b)
```asm
foo2:
enter 0, 0
mov eax, [ebp + 8]
shr eax, 31
leave
ret
```

Código 1.2.9: foo2.

(c)
```
foo3:
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov eax, [eax]
add eax, eax
pop ebp
ret
```

Código 1.2.10: foo3.

(d)
```
foo4:
push ebp
mov ebp, esp
mov ax, [ebp+8]
sub ax, [ebp+10]
pop ebp
ret
```

Código 1.2.11: foo4.

1.2.10. O código abaixo em C chama uma função em Assembly IA-32 que retorna o valor da multiplicação entre todos os elementos de um vetor. Escreva essa função em Assembly utilizando laços.

```
#include <sdtio.h>
int main(void){
    int resa, resb;
    short int a=[1,2,3,4,5,6,7,8,9,10], b=[-1,10,-3,8,-5,6,-7,4,-9,2];
    extern int f4(short *x, int n);
    resa = f4(a, 10);
    resb = f4(b, 10);
    printf("O resultado de A: %d", resa);
    printf("O resultado de B: %d", resb);
    return 0;
}
```

Código 1.2.12: Programa em C que chama função `f4` em Assembly IA-32.

1.2.11. Considere um número em ponto flutuante baseado no formato da IEEE. O número é formado por 6 bits. Um bit para o sinal, os próximos três bits para o expoente, e os últimos dois bits para a mantissa. Como visto em sala de aula, o formato IEEE possui números normalizados, não normalizados, duas representações de zero, infinito e NaN. Assumindo que arrredondamentos são feitos utilizando o arredondamento ao inteiro mais infinito (*ceil*), preencha a tabela abaixo (os lugares marcados com "-" não precisam ser preenchidos) nos campos *binário*, *mantissa*, *expoente* e *valor*. No campo *binário*, deve-se colocar o binário do número completo, enquanto que nos outros campos deve-se colocar números decimais. Pode utilizar notação exponencial (2^{512}) ou fracionária ($2\frac{1}{3}$). Quando pede-se o maior/menor número, não deve ser considerando os infinito.

Descrição	Binário	Mantissa	Expoente	Valor decimal
Menos zero	100000	0	-2.0	-0.0
Número positivo mais próximo a zero				
Infinito negativo				
Maior número normalizado				
Menor número não-normalizado				
$5.0 - 0.75$				
$4.0 + 3.0$				

Tabela 1.1: Números a serem determinados em float de 6 bits

1.2.12. Considere um número em ponto flutuante de 9 bits baseado na representação IEEE (segue as regras para números normalizados, não normalizados, representação de 0, infinito e NaN), sendo que existe 4 bits para o expoente e 4 bits para mantissa. Preencha a tabela abaixo. Se for necessário, arredonde para a fração mais próxima. No campo valor pode usar números fracionários (por exemplo, $\frac{3}{4}$) ou inteiros por potência de 2 (por exemplo 3×2^{-3}).

Número	Valor	Bit sinal	Bits expoente	Bits mantissa
Zero	0.0	0	0000	0000
Negativo mais próximo a zero				
Maior positivo				
n/a	-5.0			
n/a	$1\frac{9}{16} \times 2^{-2}$			
Menos um	-1.0			
O resultado de $4 - 1\frac{9}{16}$				

Tabela 1.2: Números a serem determinados em float de 9 bits

1.2.13. Considere um número em ponto flutuante baseado no formato da IEEE. O número é formado por 10 bits. Um bit para o sinal, os próximos cinco bits para o expoente, e os últimos quatro bits para a mantissa. Como visto em sala de aula, o formato IEEE possui números normalizados, não normalizados, duas representações de zero, infinito e NaN. Assumindo que arrredondamentos são feitos utilizando o arredondamento para o par mais próximo, preencha a tabela abaixo nos campos *binário*, *mantissa*, *expoente* e *valor*. No campo *binário*, deve-se colocar o binário do número completo, enquanto que nos outros campos deve-se colocar números decimais. Pode utilizar notação exponencial (25^{12}) ou fracionária ($2\frac{1}{3}$). Quando pede-se o maior/menor número, não deve ser considerando os infinito. A base do expoente é 2.

Descrição	Binário	Mantissa	Expoente	Valor decimal
Número negativo mais próximo a zero				
Maior número				
Menor número não-normalizado				
Menos um				

Tabela 1.3: Números a serem determinados em float de 10 bits

1.2.14. O programa em C abaixo solicita ao usuário os valoers de uma matriz 10x10 e depois calcula a soma dos elementos da diagonal principal. Altere o programa para uma versão em que o programa principal continua em C mas a função `soma()` esteja em Assembly IA-32. A função deve calcular a soma dos elementos da diagonal principal, recebendo o ponteiro da

matriz, o tamanho dela (NxN), e a variável de retorno da soma como parâmetros mediante a pilha. Mostre o código Assembly da função e indique se é necessário fazer alguma alteração no código principal em C.

```c
#include <stdio.h>
#define N 10

void soma(int M[][N], int *valor) {
    *valor=0;
    int i;
    for (i=0; i<N; i++)
        *valor += M[i][i];
}

int main(void){
    int A[N][N];
    int i, j, res;
    for (i=0; i<N; i++)
        for(j=0; j<N; j++){
            printf("Digite elemento A[%d][%d]: ", i+1, j+1);
            scanf("%d", &A[i][j]);
        }
    soma(A, &res);
    printf("Soma dos elementos da diagonal principal: %d \n", res);
    return 0;
}
```

Código 1.2.13: Programa C que chama função `soma` em Assembly.

1.2.15. Considere o seguinte fragmento de código IA-32 obtido pelo comando **objdump**. Como pode ser visto, a função `g()` é chamada pela função `f()`.

```
08048384 <g>:
 8048384        55                      push ebp
 8048385        89 e5                   mov ebp, esp
 8048387        8b 45 0c                mov eax, [ebp+0xc]
 804838a        03 45 08                add eax, [ebp+0x8]
 804838d        5d                      pop ebp
 804838e        c3                      ret

0804838f <f>:
 804838f        55                      push ebp
 8048390        89 e5                   mov ebp, esp
 8048392        83 ec 08                sub esp, 0x8
 8048395        c7 44 24 04 18 00 00    mov [esp+0x4], 0x18
 804839c        00
 804839d        c7 04 24 0c 00 00 00    mov [esp], 0xc
 80483a4        e8 db ff ff ff          call 8048384 <g>
 80483a9        89 ec                   mov esp, ebp
 80483ab        5d                      pop ebp
 80483ac        c3                      ret
```

Código 1.2.14: Resultado de um `objdump`.

(a) Quantos argumentos cada uma das funções `f()` e `g()` recebem?
(b) Quando o CPU está a ponto de executar a instrução `add` no endereço `0x0804838a` em `g()`, mostre os valores na pilha, preenchendo a tabela abaixo.

ESP+12	
ESP+8	
ESP+4	
ESP	

1.2.16. Escreva o seguinte código em Assembly IA-32. Para isso é permitido o uso da bibiloteca io.mac para ler e escrever strings (PutStr, GetStr). O programa em Assembly deve mostrar todas as mensagens indicadas no programa em C. Deve manter os ponteiros para arquivo em memória. Assuma que ambos arquivos, entrada e saída, já existem (o arquivo de saída existe, mas está vazio).

```c
#include <stdio.h>
#define BUF_SIZE 256

int main(){
    FILE *fd1, *fd2;
    char file_in[30];
    char file_out[30];
    char buf[BUF_SIZE];

    printf("Digite o nome do arquivo de entrada:");
    scanf("%s", file_in);

    printf("Digite o nome do arquivo de saída:");
    scanf("%s", file_out);

    fd1 = fopen(file_in, "r");
    fd2 = fopen(file_out, "w");

    fread(buf, sizeof(char), BUF_SIZE, fd1);
    fwrite(buf, sizeof(char), BUF_SIZE, fd2);

    fclose(fd1);
    fclose(fd2);

    return 0;
}
```

Código 1.2.15: Programa em C que lê um arquivo de entrada e escreve no de saída.

1.2.17. Faça um programa em Assembly IA-32 que multiplique duas matrizes 5x5 de inteiros. O programa deve primeiro fazer um laço para preencher a primeira matriz a ser digitada pelo usuário. Depois deve fazer um laço para preencher a segunda matriz digitada pelo usuário. Em seguida, deve multiplicar as duas matrizes e salvar o resultado em outra matriz. Utilize o endereçamento correto para matrizes. Os labals das matrizes devem ser declarados no SECTION BSS. Não é necessário imprimir mensagens para o usuário, nem mostrar o resultado final.