FACULDADE ANHANGUERA DE SÃO JOSÉ DOS AMPOS
2º/ 3º CIÊNCIA DA COMPUTAÇÃO
PROGR. APLICADA À CIÊNCIA DA COMPUTAÇÃO
PROF. EDUARDO S. PEREIRA

Lista de Exercícios 2

Breno Renan da Cunha **RA:231910411919**
Lucas Junio de Oliveira **RA:223815511919**

1. Faça a conversão de Decimal para Binário e para Hexadecimal dos seguintes números:
a) 42
101010

b) 447
110111011

c) 124
1111100

d) 255
10011011

e) 69
1000101

2. Faça as seguintes operações binárias:
(a)
1001001 -11011011= 10110110
(b)
1010101 -110011= 00100010
(c)
111100011001 + 101010100101= 010001110100

3. Escreva um código em Assembly que acesse a seguinte tabela de dados: 42, 427, 533, 69, 25.

```
%include "io.inc"
section .data

    msg: db 'Valor %d, Posicao %d', 10, 0

    DADOS: dd 42, 427, 533, 69, 25

    DADOSLEN: equ ($-DADOS)/4

section .text

global CMAIN

extern printf

CMAIN:

    mov ebx, 0;Incializa contador com 0

    for_each:

    ;Compara ebx com o tamanho do vetor DADOSLEN

    cmp ebx, DADOSLEN

    ;Se chegou no fim do vetor, finaliza programa
```

```
    je end_for_each


    ;Pega valor contido no vetor

    mov eax, [DADOS+4*ebx]


    ;Exibe o valor dos dados na tela.

    push ebx

    push eax

    push msg

    call printf

    add esp, 12

     ;Incremento o ebx em 1

    inc ebx

    jmp for_each

    end_for_each:

                    xor eax, eax

                    ret
```

4. Escreva um trecho de código que carregue o valor 5 num registrador de uso geral. Em seguida, incremente seu valor até 100, usando um laço para isso.

```
%include "io.inc"
section .data
   formatout: db "%d", 10,0
section .text
global CMAIN
extern printf
CMAIN:
        mov ebx, 5; Move 5 pro contador
        while:
                cmp ebx, 101; Comparo o contador com 101, se igual para a execução.
                je end_while; Pulo que para a execução.
           push ebx
           push formatout
           call printf
           add esp, 8

                inc ebx; Incrementa o contador.
                jmp while; Volta pro começo do laço while.
```

```
end_while:
        xor eax, eax
        ret
```

5. O que são interrupções?

Equivalente a uma chamada de procedimento (call), interrupção é um sinal de um dispositivo que tipicamente resulta em uma troca de contextos, isto é, o processador para de fazer o que está fazendo para atender o dispositivo que pediu a interrupção.

Computadores digitais geralmente oferecem uma maneira de iniciar rotinas de software em resposta a eventos eletrônicos assíncronos. Esses eventos são sinalizados para o processador através de pedidos de interrupção (IRQs). O processamento da interrupção compõe uma troca de contexto para uma rotina de software especificamente escrita para tratar a interrupção. Essa rotina é chamada rotina de serviço de interrupção, ou tratador de interrupção (interrupt handler). Os endereços dessas rotinas são chamados vetores de interrupção e são armazenados geralmente em uma tabela na memória RAM, permitindo sua modificação caso seja necessário. As Interrupções foram concebidas para evitar o desperdício de tempo computacional em loops de software (chamados polling loops) esperando eventos que serão disparados por dispositivos. Ao invés de ficarem parados esperando o evento acontecer, os processadores tornaram-se capazes de realizar outras tarefas enquanto os eventos estão pendentes. A interrupção avisa ao processador quando o evento ocorreu, permitindo dessa forma uma acomodação eficiente para dispositivos mais lentos.

6. Quais são os registradores de uso geral para 32 bits, 16 bits e 8 bits?

Usado em operações aritméticas juntamente com EAX (EDX recebe o resto da divisão e o produto da multiplicação), acesso de portas de entrada e saída, entre outros.
Eles são registradores de 32 bits, e podem ser divididos em registradores menores (de 16 bits): AX, BX, CX e DX, que por sua vez podem ser divididos em registradores menores (de 8 bits): AH, AL, BH, BL, CH, CL e DH, DL.

7. Qual o significado dos termos H e L dos registradores de 8 bits?

H e L: Utilizado para representar respectivamente seus 8 bits mais e menos significativos.

8. Quais são os componentes principais de um processador moderno?

A Unidade lógica e aritmética (ULA) - é a responsável por executar efetivamente as instruções dos programas, como instruções lógicas, matemáticas, desvio, etc.
A Unidade de controle (UC) - é responsável pela tarefa de controle das ações a serem realizadas pelo computador, comandando todos os outros componentes.
Os registradores - são pequenas memórias velozes contidas na RAM, que armazenam comandos ou valores que são utilizados no controle e processamento de cada instrução.
A MMU (em inglês: Memory Management Unit) é um dispositivo de hardware que transforma endereços virtuais em endereços físicos e dá suporte para o sistema operacional administrar a alocação da memória principal do computador entre os diversos programas em execução no computador.
E Unidade de Ponto Flutuante - Nos processadores atuais são implementadas unidades de cálculo de números reais. Tais unidades são mais complexas que ULAs e

trabalham com operandos reais, também chamados de ponto flutuante, com tamanhos típicos variando entre 32, 64 e 128 bits.

9. Quais são as estruturas de controle da programação estruturada e o que diz o princípio da programação estruturada?

Ela faz uso de três estruturas de controle para a construção de um programa: Sequência, Seleção e Repetição.
O princípio da programação estruturada formaliza a ideia de dividir em blocos
Força também o programador a saber exatamente o estado do programa antes e depois de cada bloco.

10. Escreva um trecho de código, em Assembly, que represente o While e o do While.

```
%include "io.inc"
section .data:

    formatout: db "%d", 10,0

section .text

global CMAIN

CMAIN:

    mov ebx, 20; Move 20 pro contador

    while:

    push ebx

    push formatout

    call printf; Chama função printa na tela.

    add esp, 8; Limpa pilha


    cmp ebx, 0; Compara contador com 0

    je end_while; se sim, finaliza o programa


    dec ebx; Decrementa o contador.

    jmp while; Volta pro começo do laço while.

    end_while:

    xor eax, eax

    ret
```

```
%include "io.inc"
section .data

    formatout: db "%d", 10,0

section .text

global CMAIN

extern printf

CMAIN:

        mov ebx, 20; Move 20 pro contador

        dowhile:

    push ebx; Puxo ebx

    push formatout; formato para saída

    call printf; Imprimo ebx

    add esp, 8; Limpo a pilha

        dec ebx; decremento ebx em 1


    cmp ebx, -10; se igual a -10

    je end_dowhile; finaliza

    jmp dowhile; ou volta pro começo do laço

        end_dowhile:

                xor eax, eax

                ret
```

11. Crie um programa em Assembly que rode um laço para números no intervalo de 0 a 100, use o ebx como contador. Se o valor que estiver no contado for multiplo de 2, o programa deve exibir na tela a palavra Buzz, se o número for multiplo de 3, o programa deve exibir na tela a palavra Fizz, se o número for múltiplo de ambos, 2 e 3, o programa deve exibir na tela a palavra BuzzFizz.

```
%include "io.inc"
section .data
        buzz: db "Buzz", 10, 0
        fizz: db "Fizz", 10, 0
        buzzfizz: db "Buzzfizz", 10, 0
        msg: db "%d", 10,0
section .text
```

```
global CMAIN
extern printf
CMAIN:
        mov ebx, 1; Move 1 pro contador
        while:
                cmp ebx, 101; Comparo o contador com 101, se maior para a
execução.
                je end_while; Pulo que para a execução.

                mov edx, 0; movo 0 pro registrador que vai receber o quociente da
divisão. movo 0 pro registrador que vai receber o quociente da divisão.
                mov ecx, 2; movo 2 pra ecx, pra operação.
                mov eax, ebx; Movo o valor pra eax, pra operação.
                div ecx ;edx recebe o resultado de ecx/eax

                cmp edx, 0; Comparo quociente com 0, se for, número divisivel por 2.
                je printBuzz; Pulo pro bloco de printa buzz.
                jmp print_fizz; Se não, pro printa fizz.

                printBuzz:
                        mov edx, 0; movo 0 pro registrador que vai receber o quociente
da divisão. movo 0 pro registrador que vai receber o quociente da divisão.
                        mov ecx, 3; movo 3 pra ecx, pra operação.
                        mov eax, ebx; Movo o valor pra eax, pra operação.
                        div ecx;edx recebe o resultado de ecx/eax
                        cmp edx, 0; Comparo quociente com 0, se for, número divisivel
por 3.

                        je print_buzzfizz; Sendo divisivel por ambos, printa buzzfizz na
tela.
                    jmp fimbuzz; Senão, printa buzz somente.

                fimbuzz:
                        push buzz; Puxa o buzz
                        call printf; Chama função printa na tela
                        add esp, 4; Limpa pilha.
                        jmp endprint; Pula pra parte do contador; Pula pra parte do
contador

                        print_buzzfizz:
                                push buzzfizz; Puxa o buzzfizz
                                call printf; Chama função printa na tela.
                                add esp, 4; Limpa pilha
                                jmp endprint; Pula pra parte do contador; Pula pra parte
do contador

                print_fizz:
                        mov edx, 0; movo 0 pro registrador que vai receber o quociente
da divisão. movo 0 pro registrador que vai receber o quociente da divisão.
                        mov ecx, 3; movo 3 pra ecx, pra operação.
                        mov eax, ebx; Movo o valor pra eax, pra operação.
                        div ecx;edx recebe o resultado de ecx/eax
                        cmp edx, 0; Comparo quociente com 0, se for, número divisivel
por 3.
```

```
                cmp edx, 0; Divisivel por 3
                je fimfizz; Se sim, pula pro fimfizz, pra printar o fizz de vez.
                jmp printnumero; Senão, printa o número, não divisivel por 2
nem por 3.

                fimfizz:
                        push fizz; Puxo o fizz
                        call printf; Chama função printa na tela.
                        add esp, 4; Limpa pilha
        jmp endprint; Pula pra parte do contador

                printnumero:
                        push ebx
                        push msg; Puxo o conteúdo de msg.
                        call printf; Chama função printa na tela.
                        add esp, 8; Limpa pilha
                jmp endprint; Pula pra parte do contador

        endprint:
                inc ebx; Incrementa o contador.
                jmp while; Volta pro começo do laço while.

end_while:
        xor eax, eax
        ret
```