[MAC0211] Laboratório de Programação I
Aula 5
Linguagem de Montagem
(Exemplos de Programas)

Alair Pereira do Lago

DCC–IME–USP

10 de março de 2015

"Hello, world!" para NASM (versão 32 bits) – hello.asm

```
global _start       ; exporta para o ligador (ld) o ponto de entrada

section .text
_start:

    ; sys_write(stdout, mensagem, tamanho)

    mov eax, 4            ; chamada de sistema sys_write
    mov ebx, 1            ; stdout
    mov ecx, mensagem     ; endereço da mensagem
    mov edx, tamanho      ; tamanho da string de mensagem
    int 80h               ; chamada ao núcleo (kernel)

    ; sys_exit(return_code)

    mov eax, 1          ; chamada de sistema sys_exit
    mov ebx, 0          ; retorna 0 (sucesso)
    int 80h             ; chamada ao núcleo (kernel)

section .data
mensagem: db 'Hello, world!',0x0A    ; mensagem e quebra de linha
tamanho:  equ $ - mensagem           ; tamanho da mensagem
```

"Hello, world!" para GAS (versão 32 bits) – hello.S

```
.text

   .global  _start     ; exporta para o ligador (ld) o ponto de entrada

_start:

    # sys_write(stdout, mensagem, tamanho)
    movl    $4, %eax              # chamada de sistema sys_write
    movl    $1, %ebx              # stdout
    movl    $mensagem, %ecx       # endereço da mensagem
    movl    $tamanho, %edx        # tamanho da string de mensagem
    int     $0x80                 # chamada ao núcleo (kernel)

    # sys_exit(codigo_retorno)
    movl    $1, %eax       # chamada de sistema sys_exit
    movl    $0, %ebx     # retorna 0 (sucesso)
    int     $0x80        # chamada ao núcleo (kernel)

.data
mensagem:
    .ascii  "Hello, world!\n"     # mensagem e quebra de linha
    tamanho =    . - mensagem     # tamanho da mensagem
```

"Hello, world!" para NASM (versão 64 bits)

```
global _start          ; exporta para o ligador (ld) o ponto de entrada

section .text
_start:

    ; sys_write(stdout, mensagem, tamanho)

    mov rax, 1             ; chamada de sistema sys_write
    mov rdi, 1             ; stdout
    mov rsi, mensagem      ; endereço da mensagem
    mov rdx, tamanho       ; tamanho da string de mensagem
    syscall                ; chamada ao núcleo (kernel)

    ; sys_exit(return_code)

    mov rax, 60            ; chamada de sistema sys_exit
    mov rdi, 0             ; retorna 0 (sucesso)
    syscall                ; chamada ao núcleo (kernel)

section .data
mensagem: db 'Hello, world!',0x0A     ; mensagem e quebra de linha
tamanho:  equ $ - mensagem            ; tamanho da mensagem
```

Geração do executável

Passo 1 – Geração do código objeto

- Usando NASM, em um computador de 32 bits:

```
$ nasm -f elf32 hello.asm
```

- Usando NASM, em um computador de 64 bits:

```
$ nasm -f elf64 hello.asm
```

- Usando o GAS:

```
$ as -o hello.o hello.S
```

Os comandos acima gerarão um arquivo `hello.o`.

Geração do executável

Passo 2 – Ligação (geração do código de máquina)

```
$ ld -s -o hello hello.o
```

O comando acima gerará o arquivo executável `hello` .

Desmontadores (*disassemblers*)

- Um desmontador é um programa que recebe como entrada um executável ou código objeto qualquer e gera como saída um programa em linguagem de montagem
- O desmontador associado ao `nasm` é o `ndisasm`

  Exemplo de uso:

  ```
  $ ndisasm hello.o
  ```

NASM: estrutura de um programa

Seções

- `.text` – onde fica o código-fonte; é uma seção só para leitura
- `.data` – onde fica os dados/variáveis inicializados
- `.bss` – onde fica os dados/variáveis não inicializados

NASM: declaração de variáveis

- Variável é um nome simbólico para um dado atualizável pelo programa
- Cada variável possui um tipo e recebe um endereço de memória
- Usa-se pseudo-instruções para definir o tipo da variável
- O montador atribui o endereço de memória
- Nomes de variáveis, constantes e rótulos devem:
  - conter somente letras, números ou os caracteres '_', '$', '#', '@', '~', '.' e '?'
  - iniciar por letra, '_', '?' ou '.' (sendo que o uso do ponto denota um rótulo local [1])
- NASM é sensível a letras e maísculas e minúsculas nos nomes dos identificadores

[1] Veja mais sobre rótulos locais na Seção 3.9 do documento em `http://www.nasm.us/xdoc/2.10.07/html/nasmdoc3.html`.

NASM: declaração de variáveis inicializadas

A declaração de variáveis inicializadas é feita na seção `.data` .

"Tipos" de variáveis inicializadas:

Pseudo-Instrução	**Entende-se por**
DB	define byte (1 byte)
DW	define word (2 bytes consecutivos)
DD	define doubleword (4 bytes consecutivos)
DQ	define quadword (8 bytes consecutivos)
DT	define ten bytes (10 bytes consecutivos)

NASM: declaração de variáveis inicializadas

Exemplos

```
nome      tipo   valor inicial
NUM1:     DB     64
NUM2:     DB     0150h          ; ilegal, valor > que 1 byte
NUM3:     DW     0150h
NUM5:     DW     1234h          ; byte 34h, endereço NUM5 |
                                ; byte 12h endereço NUM5+1
VET:      DB     10h,20h,30h    ; vetor de 3 bytes
LETRAS:   DB     "abC"          ; vetor de caracteres ASCII
MSG:      DB     'Ola!',0Ah,0Dh ; mistura letras e números
```

NASM: declaração de variáveis não inicializadas

A declaração de variáveis não inicializadas é feita na seção `.bss` .

"Tipos" de variáveis não inicializadas:

Pseudo-Instrução	**Entende-se por**
RESB	reserve byte (1 bits)
RESW	reserve word (2 bytes consecutivos)
RESD	reserve doubleword (4 bytes consecutivos)
RESQ	reserve quadword (8 bytes consecutivos)
REST	reserve ten bytes (10 bytes consecutivos)

Exemplos

```
nome    tipo    quantidade
BUFFER: RESB    64          ; reserva 64 × 1 bytes
NUM:    RESW    1           ; reserva 1 × 2 bytes
VET:    RESQ    10          ; reserva 10 × 8 bytes
```

Acessando a memória por meio de variáveis

```
section .data

VAR1:    DW    0F17H

section .text

MOV      EAX,VAR1        ; copia em EAX o endereço de
                         ; memória associado a VAR1

MOV      EAX,[VAR1]      ; copia em EAX o valor de VAR1,
                           ou seja, 0F17H
```

Declaração de constantes

- Constante é um nome simbólico para um valor constante usado com frequência no programa.
  É definido por meio da pseudo-instrução **EQU**.

Exemplos

nome	**EQU**	**valor**	
LF:	EQU	0Ah	; LF = caracter Line Feed
CR:	EQU	0Dh	; CR = caracter Carriage Return
LINHA:	EQU	'Digite seu nome'	

```
MSG:    DB      LINHA,LF,CR
```

Algumas diferenças entre as sintaxes da AT&T e Intel

	AT&T	**Intel**
Ordem dos operandos	MOV *origem*, *dest*	MOV*dest*, *origem*
Declaração de variáveis	var1: .int *valor*	var2: DB *valor*
Declaração de constantes	const1 = *valor*	const2: EQU *valor*
Uso dos registradores	MOV %eax,%ebx	MOV ebx,eax
Uso das variáveis	MOV $var1,%eax	MOV eax,var2
Uso das constantes	MOV $const1,%eax	MOV eax,const2
Uso de imediatos	MOV $57,%eax	MOV eax,57
Num. hexadecimais	MOV $0xFF,%eax	MOV eax,0FFh
Tam. das operações	MOVB [ebx],%al	MOV al, byte [ebx]
Delimitador de comentários	# comentário AT&T	; comentário Intel

Algumas operações úteis – Abertura de arquivos

```
; declaracao de constantes e variaveis
RDONLY: equ   0   ; modo de abertura -- somente leitura
WRONLY: equ   1   ; modo de abertura -- somente escrita
RDWR:   equ   2   ; modo de abertura -- leitura e escrita
arquivo:  db    "MAC211.txt"    ; nome do arquivo a ser lido
buffer:   resb 256              ; um buffer com 256 bytes

; abertura do arquivo
;int open (const char *pathname, int flags, mode_t mode);
mov ebx,arquivo     ; 1° parametro: caminho + nome do arquivo
mov ecx,RDONLY      ; 2° paramentro: modo de leitura
mov edx,0           ; 3° parametro: permissoes de acesso,
                    ;   so e' relevante na criacao de arquivos
mov eax,5           ; numero da chamada ao sistema (open)
int 80h             ; chamada ao nucleo do SO

; apos a execucao da interrupcao, em caso de sucesso,
; o descritor do arquivo estara' em EAX
```

Algumas operações úteis – Leitura de arquivos

```
; [continuacao do programa anterior]

; leitura do arquivo
; int read(int fd, void *buf, size_t count);
mov ebx,eax         ; 1º parametro: descritor do arquivo
mov ecx,buffer      ; 2º parametro: ponteiro para o buffer
mov edx,256         ; 3º argumento: qtde de bytes a ser lida
mov eax,3           ; numero da chamada ao sistema (read)
int 80h             ; chamada ao nucleo do SO

; apos a execucao da interrupcao, a qtde de bytes lida
; do arquivo estara' em EAX
```

Algumas operações úteis – Escrita em arquivos

```
; [continuacao do programa anterior]

;int write(int fd, const void *buf, size_t count);
mov edx,eax         ; 3º parametro: tamanho da mensagem
mov ebx,1           ; 1º parametro: stdout
mov ecx,buffer      ; 2º parametro: ponteiro para a msg
mov eax,4           ; numero da chamada ao sistema (write)
int 80h             ; chamada ao nucleo do SO

; apos a execucao da interrupcao, a qtde de bytes escrita
; estara' em EAX
```

Exercício

Usando a sintaxe Intel, faça um programa em linguagem de montagem que leia um texto da entrada padrão, passe-o para letras maiúsculas e mostre o resultado na saída padrão.
Caracteres que não são letras minúsculas devem permanecer inalterados.

Mais Exercícios (lição de casa)

1. Faça um programa em linguagem de montagem que leia um texto da entrada padrão, inverta-o e mostre o resultado na saída padrão.
2. Faça um programa em linguagem de montagem que leia um arquivo texto e conte o número de caracteres e o número de palavras presentes no arquivo. Considere que o separador de palavras é o caracter de espaço (‘ ’). Obs.: para imprimir os resultados das contagens na saída padrão, você precisará converter um número em *string*.

Bibliografia e materiais recomendados

- Capítulos 3, 4 e 6 do livro *Linux Assembly Language Programming*, de B. Neveln
- Livro *The Art of Assembly Language Programming*, de R. Hyde
  `http://cs.smith.edu/~thiebaut/ArtOfAssembly/artofasm.html`
- *The Netwide Assembler* – NASM
  `http://www.nasm.us/`
- *GNU Assembler* – GAS
  `http://sourceware.org/binutils/docs-2.23/as/index.html`
- *Linux assemblers: A comparison of GAS and NASM*
  `http://www.ibm.com/developerworks/linux/library/l-gas-nasm/index.html`
- Tabela de chamadas ao sistema no Linux
  `http://www.ime.usp.br/~kon/MAC211/syscalls.html`

Cenas dos próximos capítulos...

- Ainda sobre linguagem de montagem
  - Uso da pilha de dados
  - Subrotinas