

[MAC0211] Laboratório de Programação I
Aula 6
Linguagem de Montagem
(Pilha e sub-rotinas)

Alair Pereira do Lago

DCC-IME-USP

12 de março de 2015

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      ; 1º parametro: caminho + nome do arquivo
mov ecx,RDONLY       ; 2º parametro: modo de leitura
mov edx,0            ; 3º 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.

A solução do exercício se encontra no arquivo `maiuscula.asm`

A pilha

“Definição”:

É uma porção da memória que é compartilhada com o sistema operacional. Essa memória tem uma política de acesso do tipo “o último item a entrar será o primeiro a sair”.

É usada para:

- ▶ a comunicação entre programas e subprogramas
- ▶ armazenamento temporário
- ▶ chamadas ao sistema operacional

Mas é preciso ter cuidado...

O uso incorreto da pilha pode “quebrar” um programa ou o sistema todo.

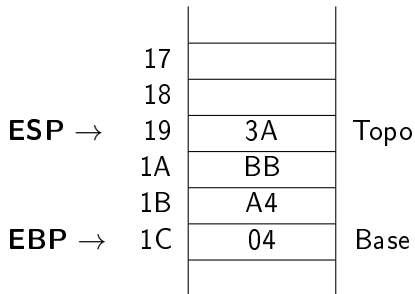
Os apontadores da pilha

Registrador EBP (de *base pointer*)

- ▶ Armazena o endereço da “base” da pilha

Registrador ESP (de *stack pointer*)

- ▶ Armazena o endereço do “topo” da pilha



Operações de manipulação da pilha – **PUSH**

Formato

► **PUSH** *reg/mem/const*

Armazena o valor do operando no topo da pilha. O número de bits que serão armazenados da pilha é definido pelo tamanho do operando.

Obs.: não pode ser aplicada a operandos de 8 bits.

Exemplo

PUSH **EAX**

que equivale a

SUB **ESP,4**

MOV **[ESP],EAX**

Operações de manipulação da pilha – POP

Formato

► **POP** *reg/mem*

Remove o valor do topo da pilha e o armazena no operando. O número de bits que serão retirados da pilha é definido pelo tamanho do operando.

Obs.: não pode ser aplicada a operandos de 8 bits.

Exemplo

```
POP  BX
```

que equivale a

```
MOV  BX,[ESP]
```

```
ADD  ESP,2
```

Operações de manipulação da pilha – **PUSH** e **POP**

Exemplo – ver arquivo `inverte.asm`

- ▶ Programa que lê da entrada padrão uma sequência de caracteres terminada por ENTER e exibe na saída padrão a sequência em ordem inversa.

Operações de manipulação da pilha – **PUSHA** e **POPA**

Formatos

► **PUSHA** ; de *push all*

Empilha o valor de todos os registradores de uso geral da arquitetura 80x86. Os registradores são empilhados na seguinte ordem: `eax`, `ecx`, `edx`, `ebx`, `esp`, `ebp`, `esi` e `edi`.

► **POPA** ; de *pop all*

Desempilha o valor de todos os registradores de uso geral da arquitetura 80x86. Os registradores são desempilhados na ordem inversa à usada no **PUSHA**.

Operações de manipulação da pilha – **PUSHF** e **POPF**

Formatos

- ▶ **PUSHF** ; de *push flags*

Empilha o valor de todas as *flags*.

- ▶ **POPF** ; de *pop flags*

Desempilha o valor de todas as *flags*.

Sub-rotinas (= subprogramas)

- ▶ São usadas para implementar tarefas complexas usando componentes mais simples
- ▶ Melhoram a legibilidade do código e facilitam sua manutenção (por evitar replicações)
- ▶ Podem ser chamadas como se fossem uma instrução presente na linguagem de programação usada
- ▶ Em linguagem de montagem, podem ser implementadas com o auxílio das instruções **CALL** e **RET**

Operações de manipulação da pilha – **CALL** e **RET**

Formatos:

► **CALL** *rot*

Salva na pilha o endereço da instrução seguinte e depois transfere a execução para o endereço especificado pelo rótulo.

Equivale a (se pudéssemos manipular o registrador EIP):

```
PUSH EIP
```

```
JMP rot
```

► **RET**

Recupera da pilha o endereço da instrução a ser executada na sequência e depois transfere a execução para esse endereço.

Equivale a (se pudéssemos usar um registrador para especificar o destino de um salto):

```
POP EBX
```

```
JMP EBX
```

Implementação de sub-rotinas

Exemplo de programa

```
_start:
    ...
    call abre_arq
    ...                ; faz operacoes de manipulacao do arquivo
    ...                ; faz outras operacoes quaisquer
    call fecha_arq
    ...

abre_arq:
    ...
    mov  eax,5        ; chamada ao sistema (open)
    int  80h
    ret

fecha_arq:
    ...
    mov  eax,6        ; chamada ao sistema (close)
    int  80h
    ret
```

Implementação de funções

Considerações gerais:

- ▶ funções são implementadas como sub-rotinas (com CALL e RET)
- ▶ a passagem de parâmetros é feita via pilha
- ▶ a pilha também é usada para armazenar as variáveis locais da função
- ▶ o valor de retorno da função pode ser devolvido na pilha ou em EAX
- ▶ a função não deve “estragar” o valor dos registradores

Responsabilidade do chamador:

- ▶ empilhar parâmetros
- ▶ chamar função
- ▶ liberar espaço dos parâmetros

Implementação de funções

Responsabilidade da função chamada:

- ▶ salvar BP do chamador
- ▶ salvar todos os registradores que vão ser afetados
- ▶ alocar espaço para variáveis locais
- ▶ realizar trabalho usando argumentos e variáveis locais
- ▶ setar valor de retorno (em espaço próprio ou em EAX)
- ▶ desalocar o espaço das variáveis locais
- ▶ restaurar registradores afetados
- ▶ restaurar BP
- ▶ retornar

Implementação de funções

Exemplo – ver arquivo `funcao.asm`

Implementação de uma função que possui o seguinte protótipo

```
int FUNC (int A, int B, int C)
```

e que tem como saída o valor de $(A^2 + B^2)/C^2$.

Exercícios

1. Faça uma função que receba como parâmetro de entrada um número inteiro i e uma *string* s (= endereço de um vetor de caracteres), transforme i em *string* e o armazene em s .
2. Faça uma função que receba como parâmetro de entrada um número inteiro i e uma *string* s (= endereço de um vetor de caracteres), e armazena em s a representação em hexadecimal do número.

Dica: você pode usar operações lógicas e rotações (SHR e SHL) para obter bits específicos de um número. Por exemplo,

SHR EAX,4

rotaciona EAX 4 bits à direita (o que equivale a dividir o número por 2^4)

Bibliografia e materiais recomendados

- ▶ Slides de uma aula da universidade de Princeton sobre funções em linguagem de montagem
<http://www.cs.princeton.edu/courses/archive/spr11/cos217/lectures/15AssemblyFunctions.pdf>
- ▶ Capítulos 3, 4, 6 e 7 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>
- ▶ Notas das aulas de MAC0211 de 2010, feitas pelo Prof. Kon
<http://www.ime.usp.br/~kon/MAC211>

Cenas dos próximos capítulos...

- ▶ Ainda sobre linguagem de montagem
 - ▶ Interface entre programas em linguagem de montagem e em C
 - ▶ Depuração