Arquitetura do Processador X86
Arquitetura de Computadores I

Prof. MSC. Wagner Guimarães Al-Alam

Universidade Federal do Ceará
Campus de Quixadá

2017-1

Agenda

- Elementos Básicos da Linguagem Assembly
- Exemplo: Somando e Subtraindo Inteiros
- Montando, Linkando e Executando Programas
- Definindo dados
- Constantes Simbólicas
- Programando em 64 bits

Elementos Básicos da Linguagem Assembly

- Constantes inteiras
- Expressões inteiras
- Constantes de caracteres e strings
- Palavras reservadas e identificadores
- Diretivas e instruções
- Labels
- Mneumonicos e operandos
- Comentários
- Exemplos

Constantes inteiras

- Presença opcional do sinal
- Dígitos binários, decimais hexadecimais ou octais
- Caracteres comuns de terminação (base)
  - h - hexadecimal
  - d - decimal
  - b - binário
  - r - real codificado
- Exemplos: 30d, 6Ah, 42, 1101b

Expressões Inteiras

- Operadores e níveis de precedência

Operador	Nome	Nível de Precedência
( )	parêntesis	1
$+$, $-$	positivo, neativo	2
$*$, $/$	multplicação, divisão	3
MOD	módulo	3
$+$, $-$	soma, subtração	4

- Exemplos

Expressão	Valor
$16/5$	3
$-(3+4)*(6-1)$	-35
$-3+4*6-1$	20
$25 mod 3$	1

Constantes de Caracteres e Strings

- Cercando caracteres em aspas simples ou duplas
  - 'A', "x"
  - Caractere ASCII = 1 byte
- Cercando strings com aspas simples ou duplas
  - "ABC"
  - 'xyz'
  - Cada caractere ocupa um byte
- Aspas incorporadas como caractere
  - 'Say "Goodnight," Gracie'

Palavras Reservadas e Identificadores

- Palavras reservadas não podem ser utilizadas como identificadores
  - Mneumônicos de instruções, diretivas, atributos de tipo, operadores, símbolos predefinidos
  - Depende de linguagem, veja no manual do Assembly que for usar (nesta disciplina NASM)
- Identificadores
  - Caracteres do 1 ao 247, incluindo dígitos
  - Caso sensitivo (NASM)
  - Primeiro caractere deve ser uma letra, _ , @, ?, ou $

Diretivas

- Comandos que são reconhecido e atuam sobre o Assembler (não é traduzido para uma instrução de máquina)
  - Não faz parte do conjunto de instruções Intel
  - Usado para declarar código, área de dados, modelo de seleção de memória, declarar procedimentos, etc.
  - Não é caso sensitivo
- Assemblers diferentes possuem diretivas diferentes
  - MASM não é o mesmo que NASM, por exemplo

Instruções

- Convertido em código de máquina pelo assembler
- Executado em tempo real pelo CPU
- Iremos usar o conjunto de instruções do Intel IA-32
- Uma instrução contém:
  - Label (opcional)
  - Mneumônico (requerido)
  - Operando (depende da instrução)
  - Comentário (opcional)

Etiquetas (Labels)

- Atuam como marcadores de local
  - marca o endereço (offset) do código de dado
- Segue as regras para identificadores
- Etiqueta de dados
  - deve ser única
  - exemplo: meuArray (não é seguido por dois pontos :)
- Etiquetas de código
  - alvo das instruções de saltos (jump) e laços (loop)
  - exemplo: L1: (seguido por dois pontos :)

Mneumonicos e Operandos

- Mneumônicos de instruções
  - guiado por memória
  - exemplos: MOV, ADD, SUB, MUL, INC, DEC
- Operandos
  - constantes
  - expressões constantes (podem ser avaliadas em tempo de compilação)
  - registradores
  - memória (etiqueta de dado)

Constantes e expressões contantes são frequentemente chamadas de **valores imediatos**

Comentários

- Comentários são uma boa prática!
  - explicam o propósito do programa
  - quando foram escritos e por quem
  - informação de revisão
  - técnicas de programação complicadas
  - explicações específicas da aplicação
- Comentários de uma linha
  - Começam com ponto e virgula
- Comentários de múltiplas linhas (MASM)
  - começam com a diretiva **COMMENT** e um caractere escolhido pelo programador
  - terminam com o mesmo caractere escolhido pelo programador

Exemplos de Formatos de Instruções

- Nenhum operando
  - stc ; set Carry flag
- Um operando
  - inc eax ; register
  - inc myByte ; memory
- Dois operandos
  - add ebx,ecx ; register, register
  - sub myByte,25 ; memory, constant
  - add eax,36 * 25 ; register, constant-expression

Exemplo: Somando e Subtraindo Inteiros - MASM

```
1  ; Dialeto MASM
2  ; AddTwo.asm - adds two 32-bit integers
3
4  .386
5  .model flat,stdcall
6  .stack 4096
7  ExitProcess PROTO, dwExitCode:DWORD
8  .code
9  main PROC
10       mov eax,5       ; move 5 to the EAX register
11
12       add eax,6       ; add 6 to the EAX register
13       INVOKE ExitProcess,0
14 main ENDP
15 END main
```

Exemplo: Somando e Subtraindo Inteiros - NASM

```
1  %include "io.inc" ;NASM macro library - biblioteca multiplataforma de macro de I/O
2
3  section .text
4  global CMAIN
5  CMAIN:
6          mov ebp, esp; for correct debugging
7          mov eax,5       ; move 5 ao registrador EAX
8          add eax,6       ; soma 6 ao registrador EAX
9          xor eax, eax ;modo mais eficiente de atribuir 0 ao registrador EAX
10         ret ;retorna ao SO o valor de EAX (convencao)
```

Exemplo de Saída

Mostrando registradores e flags no debugger

EAX=00030000 EBX=7FFDF000 ECX=00000101 EDX=FFFFFFFF
ESI=00000000 EDI=00000000 EBP=0012FFF0 ESP=0012FFC4
EIP=00401024 EFL=00000206 CF=0 SF=0 ZF=0 OF=0

Sugestão de Padrão de Código (1/2)

- Algumas abordagens sobre Capitalização (uso de CAPS)
  - não usar CAPS em nada
  - usar CAPS em tudo
  - usar CAPS em palavras reservadas, incluindo mnemônicos de instruções e nomes de registradores
  - usar CAPS somente em diretivas e operadores
- Outras sugestões
  - descrever nomes de identificadores
  - espaço cercando operadores aritméticos
  - linhas brancas entre procedimentos

Sugestão de Padrão de Código (2/2)

- Indentação e espaçamento
  - código e etiqueta de dados - sem indentação
  - instruções executáveis - indentar com 4 - 5 espaços
  - comentários: lado direito da página, alinhado verticalmente
  - 1 - 3 espaços entre instruções e seus operandos
    - ex: mov ax,bx
  - 1-2 linhas em branco entre procedimentos

Template de Programa MASM

```
1  ;Program Template (Template.asm)
2  ;Program Description:
3  ;Author:
4  ;Creation Date:
5  ;Revisions:
6  ;Date:
7  ;Modified by:
8
9  .386
10 .model flat,stdcall
11 .stack 4096
12 ExitProcess PROTO, dwExitCode:DWORD
13 .data
14 ; declare variables here
15 .code
16 main PROC
17       ; write your code here
18       INVOKE ExitProcess,0
19 main ENDP
20 ; (insert additional procedures here)
21 END main
```

Hello World - NASM

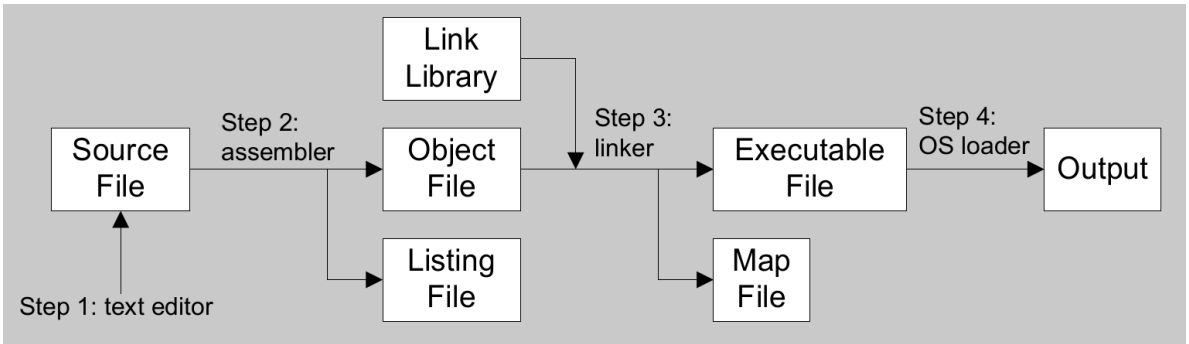
```
1 %include "io.inc"
2
3 section .text
4 global CMAIN
5 CMAIN:
6         ;write your code here
7         xor eax, eax
8         ret
```

Montando, Linkando e Executando Programas

- Ciclo Montagem-Link-Executar
- Listando Arquivo
- Mapear Arquivo

Ciclo Montagem-Link-Executar

- O diagrama que segue descreve os passos da criação do programa fonte até a execução do programa compilado.
- Se o código fonte é modificado, os passos 2 a 4 devem ser repetidos.



Listando Arquivo

- Use para ver como o programa é compilado
- Contém
  - código fonte
  - endereços
  - código objeto (linguagem de máquina)
  - nomes de segmentos
  - símbolos (variáveis, procedimentos, e constantes)

Definindo Dados

- Tipos de dados intrísecos
- Definição de estado de dados
- Definindo dados BYTE e SBYTE
- Definindo dados WORD e SWORD
- Definindo dados DWORD e SDWORD
- Definindo dado QWORD
- Definindo dado TBYTE
- Defining dado número Real
- Ordem Little Endian
- Somando variáveis ao programa AddSub
- Declarando dados não inicializados

Tipos de dados intrísecos - (1/2)

- BYTE, SBYTE
  - 8-bit unsigned integer; 8-bit signed integer
- WORD, SWORD
  - 16-bit unsigned & signed integer
- DWORD, SDWORD
  - 32-bit unsigned & signed integer
- QWORD
  - 64-bit integer
- TBYTE
  - 80-bit integer

Tipos de dados intrísecos - (2/2)

- REAL4
  - 4-byte IEEE short real
- REAL8
  - 8-byte IEEE long real
- REAL10
  - 10-byte IEEE extended real

Tipos de dados intrísecos em NASM

Usados na seção .data

Diretiva	Propósito	Espaço de Armazenamento
DB	Define Byte	aloca 1 byte
DW	Define Word	aloca 2 bytes
DD	Define Doubleword	aloca 4 bytes
DQ	Define Quadword	aloca 8 bytes
DT	Define Ten Bytes	aloca 10 bytes

Exemplos

```
choice DB 'y'
number DW 12345
neg_number DW -12345
big_number DQ 123456789
real_number1 DD 1.234
real_number2 DQ 123.456
```

Definindo Estado de dado

- Um comando de definição de dados define um armazenamento em memória para uma variável.
- Pode opcionalmente ser associada a um nome (label)
- Sintaxe: [name] directive initializer [,initializer] . . . value1 BYTE 10
- Todos inicializadores se tornam dados binários.

Definindo dados BYTE e SBYTE

Cada uma das seguintes linhas define um simples byte de armazenamento:

```
1 value1 BYTE 'A' ; character constant
2 value2 BYTE 0 ; smallest unsigned byte
3 value3 BYTE 255 ; largest unsigned byte
4 value4 SBYTE -128 ; smallest signed byte
5 value5 SBYTE +127 ; largest signed byte
6 value6 BYTE ? ; uninitialized byte
```

- MASM não previne você de inicializar um BYTE com um valor negativo, mas isso é considerada uma má prática.
- Se você declara uma variável SBYTE, o debugger Microsoft irá automaticamente mostra seu valor em decimal com sinal.

Definindo Array de BYTE

Exemplo que usa inicializadores múltiplos:

```
1 list1 BYTE 10,20,30,40
2 list2 BYTE 10,20,30,40
3       BYTE 50,60,70,80
4       BYTE 81,82,83,84
5 list3 BYTE ?,32,41h,00100010b
6 list4 BYTE 0Ah,20h,'A',22h
```

Definindo STRINGS - (1/3)

- Uma string é implementada como um array de caracteres
  - Por conveniência, isso é usualmente cercado por aspas
  - Frequentemente será encerrado por null (0)
- Examples:

```
1 str1 BYTE "Enter your name",0
2 str2 BYTE 'Error: halting program',0
3 str3 BYTE 'A','E','I','O','U'
4 greeting BYTE "Welcome to the Encryption Demo program "
5          BYTE "created by Kip Irvine.",0
```

Definindo STRINGS - (2/3)

- Para continuar uma simples string em múltiplas linhas, termine cada linha com uma vírgula:

```
1 menu BYTE "Checking Account",0dh,0ah,0dh,0ah,
2       "1. Create a new account",0dh,0ah,
3       "2. Open an existing account",0dh,0ah,
4       "3. Credit the account",0dh,0ah,
5       "4. Debit the account",0dh,0ah,
6       "5. Exit",0ah,0ah,
7       "Choice> ",0
```

Definindo STRINGS - (3/3)

- Sequência de terminação de linha (end-of-line):
  - 0Dh = carriage return
  - 0Ah = line feed

```
1 str1 BYTE "Enter your name:
2 ",0Dh,0Ah
3 BYTE "Enter your address: ",0
4 newLine BYTE 0Dh,0Ah,0
```

S

ugestão: Definir todas as strings usadas em seu programa na mesma área do segmento de dados.

Usando o Operador DUP

- Use DUP para alocar (criar espaço para) um array de string. Sintaxe: counter DUP (argument)
- Counter e argument devem ser constantes ou expressões constantes

```
1 var1 BYTE 20 DUP(0) ; 20 bytes, all equal to zero
2 var2 BYTE 20 DUP(?) ; 20 bytes, uninitialized
3 var3 BYTE 4 DUP("STACK") ; 20 bytes: "STACKSTACKSTACKSTACK"
4 var4 BYTE 10,3 DUP(0),20 ; 5 bytes
```

NASM

Em NASM o comando equivalente ao DUP é o TIMES:
zerobuf: times 64 db 0

Definindo dados WORD e SWORD

- Define armazenamento para inteiros de 16-bit
  - ou caracteres double
  - valor simples ou múltiplos valores

```
1 word1 WORD   65535       ;largest unsigned value
2 word2 SWORD  -32768      ;smallest signed value
3 word3 WORD   ?           ;uninitialized, unsigned
4 word4 WORD   "AB"        ;double characters
5 myList WORD  1,2,3,4,5   ;array of words
6 array WORD   5 DUP(?)    ;uninitialized array
```

Definindo dados WORD e SWORD

Definições de armazenamento para inteiros com sinal e sem sinal de 32 bits:

```
1 val1 DWORD  12345678h      ;unsigned
2 val2 SDWORD -2147483648    ;signed
3 val3 DWORD  20 DUP(?)      ;unsigned array
4 val4 SDWORD -3,-2,-1,0,1   ;signed array
```

Definindo Dado QWORD, TBYTE, Real

Definição de armazenamento para quadwords, valores tenbyte, e números reais:

```
1 quad1 QWORD 1234567812345678h
2 val1 TBYTE 100000000123456789Ah
3 rVal1 REAL4 -2.1
4 rVal2 REAL8 3.2E-260
5 rVal3 REAL10 4.6E+4096
6 ShortArray REAL4 20 DUP(0.0)
```

Ordem Little Endian

- Todos os tipos de dados maiores de byte armazenam seus bytes individuais na ordem reversa. O byte menos significativo ocorre no primeiro endereço de memória.
- Exemplo:
  - val1 DWORD 12345678h

0000:	78
0001:	56
0002:	34
0003:	12

Adicionando Variáveis ao AddSub - MASM

```
1  TITLE Add and Subtract, Version 2
2  (AddSub2.asm)
3  ; This program adds and subtracts 32-bit unsigned
4  ; integers and stores the sum in a variable.
5  INCLUDE Irvine32.inc
6  .data
7  val1 DWORD 10000h
8  val2 DWORD 40000h
9  val3 DWORD 20000h
10 finalVal DWORD ?
11 .code
12 main PROC
13         mov eax,val1  ; start with 10000h
14         add eax,val2  ; add 40000h
15         sub eax,val3  ; subtract 20000h
16         mov finalVal,eax ; store the result (30000h)
17         call DumpRegs ; display the registers
18         exit
19 main ENDP
20 END main
```

Adicionando Variáveis ao AddSub - NASM

```
1  %include "io.inc"
2
3  section .data
4  val1 dd 1000h
5  val2 dd 4000h
6  val3 dd 2000h
7
8  section .bss
9  finalVal: RESW 1
10
11 section .text
12 global CMAIN
13 CMAIN:
14         mov ebp, esp; for correct debugging
15         mov eax,[val1] ; start with 1000h
16         add eax,[val2] ; add 4000h
17         sub eax,[val3] ; subtract 2000h
18         mov [finalVal],eax ; store the result (3000h)
19
20         mov edx, 3000h;permite validar a resposta
21
22         xor eax, eax
23         ret
```

Declarando Dados Não Inicializados - MASM

- Use a diretiva .data? para declarar um segmento de dados não inicializado:
  - .data?
- Neste segmento, declare variáveis com o inicializador "?":
  - smallArray DWORD 10 DUP(?)

Vantagem

O arquivo executável do programa tem seu tamanho reduzido.

Reserva de Dados Não Inicializados - NASM

Diretiva	**Propósito**
RESB	Reserve um Byte
RESW	Reserve um Word
RESD	Reserve um Doubleword
RESQ	Reserve um Quadword
REST	Reserve um Ten Bytes

Constantes Simbólicas

- Diretiva sinal de igual
- Calculando o tamanho de Arrays e Strings
- Diretiva EQU
- Diretiva TEXTEQU

Diretiva sinal de igual

- name = expression
  - expression é um inteiro de 32-bit (expression or constant)
  - pode ser redefinido
  - name é chamado de constante simbólica
- é uma boa prática de programação usar símbolos

```
1 COUNT = 500
2 .
3 .
4 mov ax,COUNT
```

Calculando o tamanho de Arrays e Strings

- contador da localização corrente: $
  - subtrair o endereço da lista
  - a diferença é o número de bytes

```
1 list BYTE 10,20,30,40
2 ListSize = ($ - list)
```

Calculando o Tamanho de um Array de Palavras

Divide o número total de bytes por 2 (o tamanho de uma palavra)

```
1 list WORD 1000h,2000h,3000h,4000h
2 ListSize = ($ - list) / 2
```

Calculando o Tamanho de um Array de Doubleword

Divide o número total de bytes por 4 (o tamanho de um doubleword)

```
1 list DWORD 1,2,3,4
2 ListSize = ($ - list) / 4
```

Diretiva EQU

- Define um símbolo com um inteiro ou expressão de texto.
- Não pode ser redefindo

```
1 PI EQU <3.1416>
2 pressKey EQU <"Press any key to continue...",0>
3 .data
4 prompt BYTE pressKey
```

Diretiva TEXTEQU

- Define um símbolo com um inteiro ou expressão de texto.
- Chamado macro de texto
- Pode ser redefinido

```
1  continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">
2  rowSize = 5
3  .data
4  prompt1 BYTE continueMsg
5  count TEXTEQU %(rowSize * 2)   ; evaluates the expression
6  setupAL TEXTEQU <mov al,count>
7
8
9  .code
10 setupAL   ; generates: "mov al,10"
```

Programação em 64 bits

- MASM suporta programação em 64-bit, contudo as seguintes diretivas não são permitidas:
  - INVOKE, ADDR, .model, .386, .stack

Versão 64-Bit de AddTwoSum

```
1  ; AddTwoSum_64.asm - Chapter 3 example.
2  ExitProcess PROTO
3  .data
4  sum DWORD 0
5  .code
6  main PROC
7  mov eax,5
8  add eax,6
9  mov sum,eax
10
11 mov ecx,0
12 call ExitProcess
13 main ENDP
14 END
```

Considerações do Slide Anterior

- As seguintes linhas não são necessárias:
  .386
  .model flat,stdcall
  .stack 4096
- INVOKE não é suportado.
- Instrução CALL não pode receber argumentos
- Use registradores 64-bit quando possível