Linguagem de Montagem e Assembly

André Luiz da Costa Carvalho

Linguagem de Montagem

- Todo programa para ser executado precisar ser convertido de linguagem fonte (alto nível) para um programa equivalente em linguagem de máquina para que o processador possa entender e executar.

Assembly

- Assembly é uma linguagem de baixo nível, chamada frequentemente de “linguagem de montagem”.

- É uma linguagem considerada difícil, principalmente porque o programador precisa conhecer a estrutura da máquina para usá-la.

Assembly

- A linguagem Assembly é atrelada à arquitetura de uma certa CPU, ou seja, ela depende completamente do hardware
- Cada família de processador tem sua própria linguagem assembly (Ex. X86, ARM, SPARC, MIPS)
- Por essa razão Assembly não é uma linguagem portável, ao contrário da maioria das linguagens de alto nível

Assembly (História)

- As primeiras linguagens Assembly surgiram na década de 50, na chamada segunda geração das linguagens de programação.
- A segunda geração visou libertar os programadores de dificuldades como lembrar códigos numéricos e calcular endereços.

Assembly (História)

- Assembly foi muito usada para várias aplicações até os anos 80, quando foi substituída pelas linguagens de alto nível.

- Isso aconteceu principalmente pela necessidade de aumento da produtividade de software.

Assembly (História)

- Atualmente, Assembly é usada para manipulação direta de hardware e para sistemas que necessitem de performance crítica
- *Device drivers*, sistemas embarcados de baixo nível e sistemas de tempo real são exemplos de aplicações que usam Assembly.

Assembly (Assembler)

- A linguagem Assembly é de baixo nível, porém ainda precisa ser transformada na linguagem que a máquina entende

- Quem faz isso é o **Assembler**, um utilitário que traduz o código Assembly para a máquina

Assembly (Assembler)

- Exemplo:
  - Antes -> mov al, 061h (x86/IA-32)
  - Depois -> 10110000 01100001

Assembler - Rodando

- Windows:
  - DEBUG – Aplicação para ver o resultado dos registradores no momento.
- Linux:
  - NASM – Compila para assembler.
- Ambos:
  - gcc –S compila um código C em assembler

Assembly (Fundamentos)

- Byte, Word e Dword são blocos de dados básicos.
- O processador trabalha com o tamanho de dados adequados para executar as instruções.
- Um byte possui 8 bits, um word possui 16 bits ou 2 bytes e um dword possui 32 bits ou 4 bytes.

Assembly (Fundamentos)

- Em Assembly é comum representar os números na forma hexadecimal.
  - É interessante visualizar o número na forma de dados
- A representação hexadecimal facilita o tratamento de números muito grandes e permite saber quais bits estão “ligados” ou “desligados”

Assembly (Fundamentos)

- Um algarismo hexadecimal pode ser representado por quatro algarismos binários

- Logo um byte pode ser representado como dois números hexa, um word como quatro números hexa e um dword como oito números hexa

Assembly (Fundamentos)

Binário	Hexa	Decimal	Tipo
10000000	80	128	byte
1000000000000001	8001	32.769	word
1111111111111111	FFFF	65.535	word
11111111111111111111111111111111	FFFFFFFF	4.294.967.295	dword

Assembly (Registradores)

- Registradores são áreas especiais dentro do processador que são mais rápidas que operandos de memória.

- Usando como base um processador Intel, existem apenas 8 registradores de uso geral.

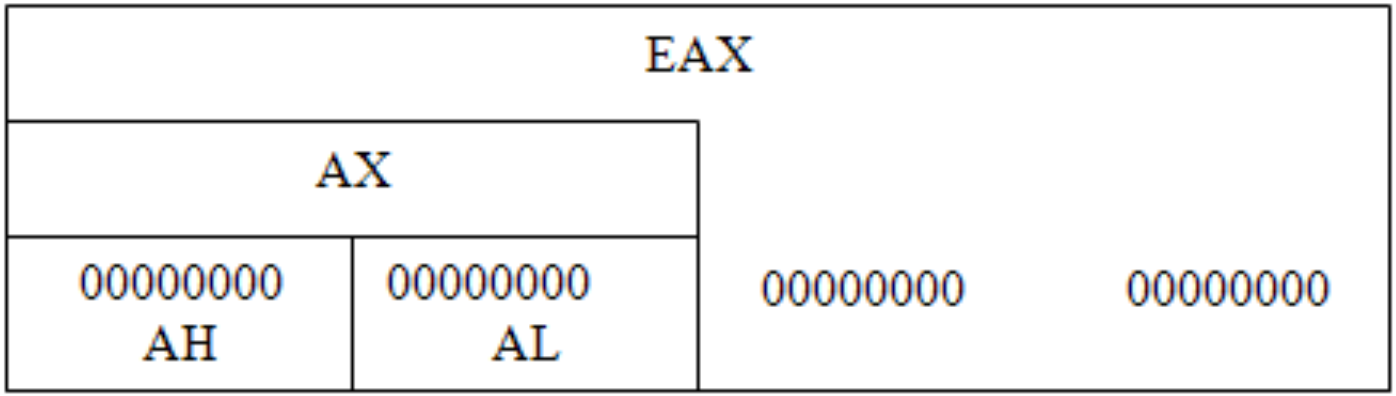
Assembly (Registradores)

- São eles:
  - EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP

- Os registradores ESP e EBP só devem ser usados preferencialmente para trabalhar com a pilha.

Assembly (Registradores)

- Nos registradores de uso geral (Exceto ESI e EDI) é permitido usar três modos de acesso diferentes, ilustrados pela figura abaixo:



Assembly (Registradores)

- EAX
  - Chamado de “Acumulador”, geralmente é usado para operações aritméticas e para guardar resultados
- EBX
  - Chamado de “Base”, geralmente é usado para armazenar dados em geral e para endereços de memória

Assembly (Registradores)

- ECX
  - Chamado de “Contador”, como o nome já diz é usado como contador, principalmente para controlar loops
- EDX
  - Chamado de registrador de dados, é usado geralmente para guardar o endereço de uma variável na memória

Assembly (Registradores)

- ESI e EDI
  - Respectivamente “Source Index” e “Destination Index”, são usados para movimentação de dados, com ESI guardando o endereço fonte de uma variável e EDI guardando o endereço destino.
  - Não podem ser acessados em nível de Byte.

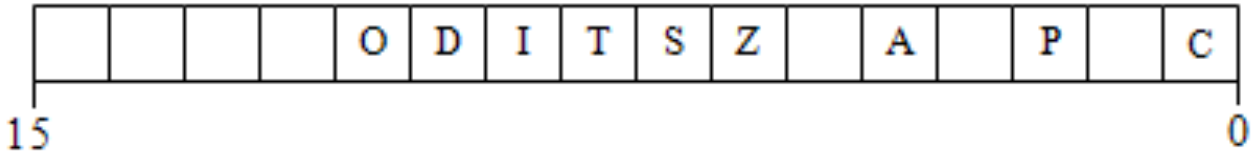
Assembly (Registradores)

- ESP e EBP
  - Respectivamente “Stack Pointer” e “Base Pointer”
  - Só devem ser usados para manipulação da pilha.
  - O Registrador ESP guarda a referência para o topo da pilha, enquanto o registrador EBP é usado para “andar” pela pilha

Assembly (Registradores)

- Entre os registradores que não são de uso geral, existe um registrador muito relevante para o programador, o registrador flags.
- Através do registrador flags podemos saber se dois valores são iguais, se um é maior que outro ou se um valor é negativo, além de outras informações .

Assembly (Registradores)



- O => Overflow
- D => Direction
- I => Interrupt Enable
- T => Trap
- S => Signal

Z => Zero
A => Auxiliar Carry
P => Parity
C => Carry

Constantes e Aritmética

- Assembler aceita, além de números hexadecimais e decimais, expressões matemáticas simples.
- Estas expressões são avaliadas pelo Assembler (construtor assembler) antes de ser convertidas em linguagem de máquina.
  - Nunca em tempo real!
- Strings também podem ser utilizadas.
  - Vetores de bytes.

Memória

- Além de registradores, programas assembly podem acessar a memória do computador.
  - Variáveis.
- Declaração: <nome> <tamanho> <valor>
  - Tamanho: BYTE, WORD, DWORD, QWORD.
- Exemplo:
  - Media BYTE 5
  - NUMERO DWORD 100000
  - Nome BYTE “Andre”

Comentários

- Comentários em Assembler são feitos com “;”

Código assembler padrão

```
.data
;Declarações de Variáveis

.code
;codigo fonte
```

Assembly (Instruções)

- Movimentação de dados:
  - mov destino, fonte (Sintaxe Intel)
  - No máximo um operador de memória.

```
.data
var1 BYTE 10h
.code
mov al,var1                    ; AL = 10h
mov al,[var1]                  ; AL = 10h
```

Assembly (Instruções)

Intel	AT&T
mov eax, 1	movl $1, %eax
mov ebx, 0ffh	movl $0xff, %ebx
mov eax, [ebx]	movl (%ebx), %eax
mov eax, [ebx+3]	movl 3(%ebx), %eax

Assembly (Instruções)

- Dados de tamanhos diferentes
  - MOVZX – adiciona zeros na parte faltante
  - movzx eax,bx
- Trocar valores:
  - XCHG <val1> <val2>
  - Ao menos 1 deve ser um registro

Assembly (Instruções)

- Incremento/Decremento
  - INC/DEC

```
inc ah
dec bx
```

Assembly (Instruções)

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
    inc myWord
    dec myWord
    inc myDword

    mov ax,00FFh
    inc ax
    mov ax,00FFh
    inc al
```

Assembly (Instruções)

```
.data
myWord  WORD 1000h
myDword DWORD 10000000h
.code
    inc myWord    ; 1001h
    dec myWord    ; 1000h
    inc myDword   ; 10000001h

    mov ax,00FFh
    inc ax ; AX = 0100h
    mov ax,00FFh
    inc al ; AX = 0000h
```

Assembly (Instruções)

- Exercício

```
.data
myByte BYTE 0FFh
.code
    mov al,myByte           ; AL =
    mov ah,[myByte+1]       ; AH =
    dec ah                  ; AH =
    inc al                  ; AL =
    dec ax                  ; AX =
```

Assembly (Instruções)

- Exercício

```
.data
myByte BYTE 0FFh
.code
    mov al,myByte              ; AL = FFh
    mov ah,[myByte+1]          ; AH = 00h
    dec ah                     ; AH = FFh
    inc al                     ; AL = 00h
    dec ax                     ; AX = FEFF
```

Assembly (Instruções)

- Instrução de soma:
  - add destino, fonte (Sintaxe Intel)

Exemplo: add eax,[ebx+ecx]

- Instrução de subtração:
  - sub destino, fonte (Sintaxe Intel)

Exemplo: sub eax,ebx

Assembly (Instruções)

```
mov eax,10000h ; EAX = 10000h
add eax,40000h ; EAX = 50000h
sub eax,20000h ; EAX = 30000h
```

Assembly (Instruções)

```
.data
var1 DWORD 10000h
var2 DWORD 20000h
.code                          ; ---EAX---
   mov eax,var1                ; 00010000h
   add eax,var2                ; 00030000h
   add ax,0FFFFh               ; 0003FFFFh
   add eax,1                   ; 00040000h
   sub ax,1                    ; 0004FFFFh
```

Assembly (Instruções)

NEG (negate) Instruction

Inverte o sinal de um número

```
.data
valB BYTE -1
valW WORD +32767
.code
   mov al,valB              ; AL = -1
   neg al                   ; AL = +1
   neg valW                 ; valW = -32767
```

Assembly (Instruções)

- Instruções de operações lógicas:
  - and/or/xor destino, fonte (Sintaxe Intel)

    Exemplo: and ax,bx

  - and/or/xor fonte, destino (Sintaxe AT&T)

    Exemplo: andw %bx,%ax

Assembly (Instruções)

- Instrução de comparação:
  - cmp operando1, operando2 (Intel)

    Exemplo: cmp 08h, eax

  - cmp operando1, operando2 (AT&T)

    Exemplo: cmp $0x8, %eax

Assembly (Pilha)

- Todos os programas fazem uso da pilha em tempo de execução, porém nas linguagens de alto nível não é preciso se preocupar com o funcionamento da pilha
- Já em Assembly, o programador precisa saber trabalhar com a pilha, pois ela é uma ferramenta importante

Assembly (Pilha)

- A pilha é uma área de dados existente na memória em tempo de execução, na qual seu programa pode armazenar dados temporariamente
- O processador é rápido no acesso à pilha, tanto para escrever quanto para ler

Assembly (Pilha)

- As principais funcionalidades da pilha são:
  - Preservar valores de registradores em funções
  - Preservar dados da memória
  - Transferir dados sem usar registradores
  - Reverter a ordem de dados
  - Chamar outras funções e depois retornar
  - Passar parâmetros para funções

Assembly (Instruções)

- Instruções de manipulação da pilha:
  - push eax (Sintaxe Intel)
  - push %eax (Sintaxe AT&T)
  - pop eax (Sintaxe Intel)
  - pop %eax (Sintaxe AT&T)

Assembly (Pilha)

```
push ax
push bx
push cx
push dx
push ds
push es
push di
push si
```

```
push a
push es, ds
pop a
pop es, ds
```

Pilha

- Trocando o valor de dois registradores via pilha:

Push eax

Push ebx

Pop eax

Pop ebx

Assembly (Instruções)

- Instruções de jump
  - (“Pulo” incondicional):
  - Marca-se um label no código com “:” no final.
  - Jump label fará o código voltar para aquele lugar.

```
Volta:
        add eax,10
        sub ebx,10
        jump volta
```

Assembly (Instruções)

- LOOP – repetição por contagem.
  - Registrador ECX guarda a contagem das iterações.
  - A cada rodada ele é decrementado.
  - Repetição para ao chegar em zero.

```
    mov ax,6
    mov ecx,4
L1:
    inc ax
    loop L1
```

Assembly (Instruções)

- E repetições aninhadas?
  - A repetição mais externa deve guardar o valor de ecx antes de entrar na interna

```
L1:
    push ecx  ; guardando contador
    mov ecx,20    ; setando o interno
L2: .
    .
    loop L2    ; loop interno
    pop ecx; voltando ao valor original
    loop L1
```

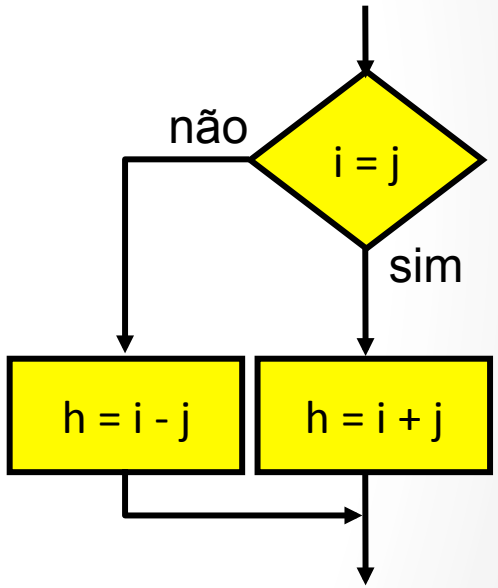
Assembly (Instruções)

- Comparação.
  - cmp val1 val2
  - Flag zero = 1 se forem iguais.
  - Val1 > val2 Carry Flag = 1 Zero Flag = 0
  - Val1< val2 Carry Flag =0 Zero Flag = 0
- Por que?

Assembly (Instruções)

- Instruções de Jump Condicional

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0



Assembly (Instruções)

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

Assembly (Instruções)

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

Assembly (Instruções)

- Exemplos:

```
cmp eax,ebx
jne  Diferente
```

- Compara 2 registradores e põe o resultado na variável maior:

```
    mov maior,bx
    cmp ax,bx
    jna Next
    mov maior,ax
Next:
```