Conjunto de Instruções do Pentium (ISA)

Prof. Eduardo Tavares

eagt@cin.ufpe.br

Tipos de Instruções

Transferência de Dados

Aritmética

Manipulação de bit

Laços e Saltos

Subrotina e interrupções

Controle

Strings

Dados numéricos

Byte (8 bits)

Word (16 bits)

Double-Word (32 bits)

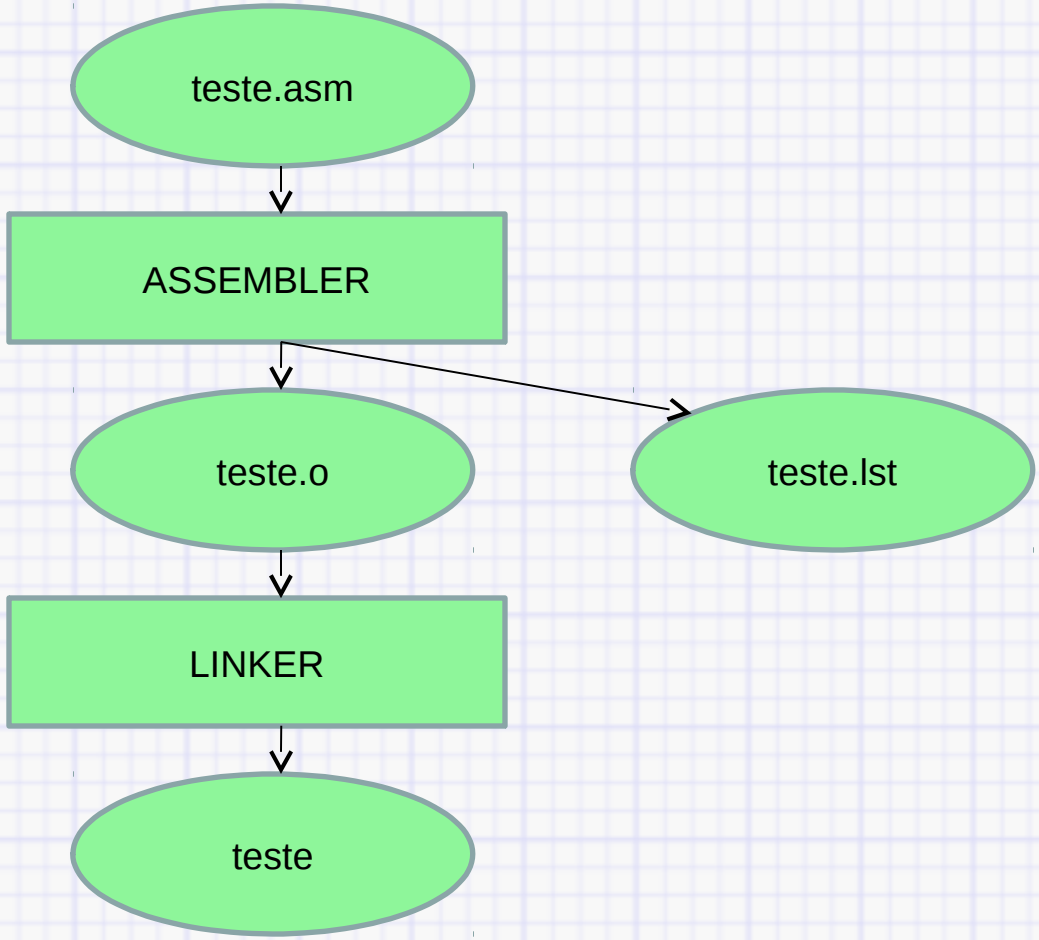
Quadword (64 bits)

...

Template

```
.MODEL SMALL
.DATA

;Dados da aplicação

.CODE
.STARTUP

;Código

.EXIT

; Espaço para possíveis procedimentos

END
```

Montagem



Tipo de instruções

Instruções Monádicas

- **inst *opr dest***

***opr dest* - operando destino**

Instruções Diádicas

- **inst *opr dest*, *opr fonte***

***opr dest* - operando destino**

***opr fonte* - operando fonte**

Transferência de Dados

Instruções

- MOV movimentacao
- MOVSX ext. sinal
- MOVZX ext. c/ zero
- PUSH empilhamento
- PUSHW/PUSHD imedit./ 32 bits
- PUSHA/PUSHAD todos reg/ 32 bits
- PUSHF/PUSHFD 16 bits flags/32 bits
- POP desemp.
- POPA/POPAD todos reg/32 bits
- POPF/POPFD 16 bits flags/32 bits
- IN *input*
- OUT *output*
- XCHG permuta
- XLAT *translate-table*
- LEA carga de end. efetivo
- LDS e similares carga de ponteiro
- BSWAP troca
- LAHF carga de flags em AH
- SAHF arm. de AH nos flags

Transferência de Dados

Sintaxe:

MOV OP_DEST,OP_FONT

Uso:

mov registrador, registrador

mov registrador, imediato

mov memória, imediato

mov registrador, memória

mov memória registrador

EX.: MOV AL,01

MOV AX,[BX+2]

MOV AL,[0400H]

Transferência de Dados

Sintaxe:

MOVSX OP_DEST,OP_FONT

Uso:

movsx reg16, src8

movsx reg32, src8

movsx reg32, src16

src =registrador ou memória

EX.:

AL=36H,BX=C3EEH

MOVSX AX,AL

MOVSX EBX,BX

Transferência de Dados

Sintaxe:

MOVZX OP_DEST,OP_FONT

Uso:

movsx reg16, src8

movsx reg32, src8

movsx reg32, src16

src =registrador ou memória

EX.: MOVZX AX,AL

Transferência de Dados

Sintaxe:

PUSH OP_FONTE

Uso:

OP_FONTE = registrador,memória,

registrador de segmento, imediato

EX.: PUSH BX

Transferência de Dados

Sintaxe:

PUSHW/PUSHD OP_FONTE (imedit./ 32 bits)

EX.: PUSHW 34ADh

PUSHD EAX

PUSHD pode ser usado para empilhar valor de um reg. de 32 bits

PUSH EAX e PUSHD EAX são equivalentes

Transferência de Dados

Sintaxe:

PUSHA/PUSHAD **(todos reg/ 32 bits)**

16 bits: AX,CX,DX,BX,SP,BP,SI,DI

32 bits: EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI

EX.: PUSHA

PUSHAD

Transferência de Dados

Sintaxe:

PUSHF/PUSHFD **16 bits flags/32 bits**

EX.: PUSHF

PUSHFD

Transferência de Dados

Sintaxe:

POP OP_DEST

Uso:

OP_DEST = registrador (de segmento), memória

SP é incrementado em 2 ou 4 bytes dependendo de OP_DEST

EX.: POP BX

Transferência de Dados

Sintaxe:

POPA/POPAD **todos reg/32 bits**

16 bits: DI,SI,BP,SP,BX,DX,CX,AX

32 bits: EDI,ESI,EBP,ESP,EBX,EDX,ECX,EAX

EX.: POPA

POPAD

Transferência de Dados

Sintaxe:

POPF/POPFD **16 bits flags/32 bits**

EX.: POPF

POPFD

Transferência de Dados

Sintaxe:

IN ACUMULADOR,END/DX ***input***

USO:

ACUMULADOR=AL,AX,EAX

END = IMEDIATO (256)

DX = registrador (65.535)

(Espaço de endereçamento E/S diferente da memória)

EX.: IN AL,60H

IN AL,DX

IN AL, 21H

Transferência de Dados

Sintaxe:

OUT END/DX,ACUMULADOR ***output***

USO:
ACUMULADOR=AL,AX,EAX

EX.: OUT DX,AL
OUT 80H,AX

```
IN AL, 21H ;Read existing bits.
AND AL, 0EFH ;Turn on IRQ 4 (COM1).
OUT 21H, AL ;Write result back to PIC.
```

Transferência de Dados

Sintaxe:

XCHG OP_DEST,OP_FONTE **permuta**

Uso:

Ambos registradores ou 1 é operando na memória

EX.: XCHG AL,AH

XCHG AL, [BX]

Transferência de Dados

Sintaxe:

XLAT ***translate-table***

USO:

O registrador BX deve ter o endereço inicial da lista (usando DS)

O valor de AL é o índice.

EX.: MOV AL,3

MOV BX,0400H

XLAT

Transferência de Dados

Load Effective Address (LEA)

Sintaxe:

LEA OP_DEST,LABEL **carga de end.**

USO:
OP_DEST = registrador
LABEL = operando na memória

EX.: LEA BX,LISTA
LEA AX,[SI]

Obs: diferente de MOV BX,LISTA de MOV AX,[SI]

Transferência de Dados

Sintaxe:

LDS OP_DEST,OP_FONTE **carga de ponteiro**

USO:

OP_DEST – registrador

OP_FONTE – memória

Os primeiros 2 ou 4 bytes são colocados em OP_DEST o restante 2 bytes no registrador DS

Obs: LES,LFS,LGS,LSS - Instruções similares

EX.: LDS BX,[SI]

Transferência de Dados

Sintaxe:

BSWAP OP_DEST **- troca (s*wapping)***

- OP_DEST - registrador de 32 bits

EX.: MOV EAX,12345678H
BSWAP EAX

EAX = 78563412H

Transferência de Dados

Sintaxe:

LAHF **- Carrega AH com flags**

EX.: POPF

LAHF

Transferência de Dados

Sintaxe:

SAHF **- Armazena AH nos flags**

EX.: MOV AH,0FFH

SAHF

Transferência de Dados

MOV AL,[SI]

MOV [SI], AX

Problemas:

MOV [SI],5 (é um byte? Word?...)

Transferência de Dados

MOV AL,[SI]

MOV [SI], AX

Problemas:

MOV [SI],5 (é um byte? Word?...)

Solução:

MOV BYTE PTR [SI],5

MOV WORD PTR [SI], 5

MOV DWORD PTR [SI],5

Transferência de Dados

LEA DX, MESSAGE

MOV DX, OFFSET MESSAGE (equivalente)

E se precisar do segmento?

MOV AX, SEG MESSAGE

MOV DS, AX

Aritmética

Instruções

- **ADD** adição
- **ADC**
- **INC** incremento
- **SUB** subtração
- **SBB**
- **DEC** decremento
- **CMP** comparação
- **MUL** multiplicação
- **IMUL**
- **DIV** divisão
- **IDIV**

- **NEG** complemento a 2
- **CBW** mudança de módulo
- **CWD**
- **DAA** ajuste BCD
- **DAS**
- **AAA** ajuste ASCII
- **AAS**
- **AAM**
- **AAD**

Aritmética

Sintaxe:

ADD OP_DEST,OP_FONTE - adição (sem vai 1)

USO:

registrador, registrador

registrador, imediato

memória, imediato

registrador, memória

memória, registrador

C	O	Z	S	P	A
M	M	M	M	M	M

EX.: ADD AL,CH

ADD CX,[SI]

ADD DX,4

Instruction	Before add		After add
	source	destination	destination
add AX,DX	DX = AB62H	AX = 1052H	AX = BBB4H
add BL,CH	BL = 76H	CH = 27H	BL = 9DH
add value,10H	—	value = F0H	value = 00H
add DX,count	count = 3746H	DX = C8B9H	DX = FFFFH

Aritmética

Sintaxe:

ADC OP_DEST,OP_DEST - **adição com vai um (dest + src + CF)**

USO:

registrador, registrador

registrador, imediato

memória, imediato

registrador, memória

memória, registrador

C	O	Z	S	P	A
M	M	M	M	M	M

EX.: ADC AL,CH

ADC [BX],DX

ADC DX,4

Aritmética

Sintaxe:

INC OP_DEST **- Incremento**

USO:
OP_DEST = registrador, operando na memória

C	O	Z	S	P	A
–	M	M	M	M	M

EX.: INC AL

INC [BX]

INC [0400H]

Aritmética

Sintaxe:

SUB OP_DEST,OP_DEST **- Subtração**

C	O	Z	S	P	A
M	M	M	M	M	M

EX.:

SUB AL,CH

SUB CX,[0300H]

SUB DX,4

instruction	Before sub		After sub
	source	destination	destination
sub AX,DX	DX = AB62H	AX = 1052H	AX = 64F0H
sub BL,CH	CH = 27H	BL = 76H	BL = 4FH
sub value,10H	—	value = F0H	value = E0H
sub DX,count	count = 3746H	DX = C8B9H	DX = 9173H

Aritmética

Sintaxe:

SBB OP_DEST,OP_FONTE **- Subtração com vem um (dest – (src+cf))**

C	O	Z	S	P	A
M	M	M	M	M	M

EX.: SBB AL,CH

SBB CX,[0300H]

SBB DX,4

Aritmética

Sintaxe:

DEC OP_DEST **- Decremento**

Uso:

OP_DEST = registrador, memória

C	O	Z	S	P	A
–	M	M	M	M	M

EX.: DEC AL

DEC [BX]

DEC [0400H]

Aritmética

Sintaxe:

CMP OP_DEST,OP_FONTE **- Comparação (dest-src)**

EX.: CMP AL,CH

CMP CX,[0300H]

CMP DX,4

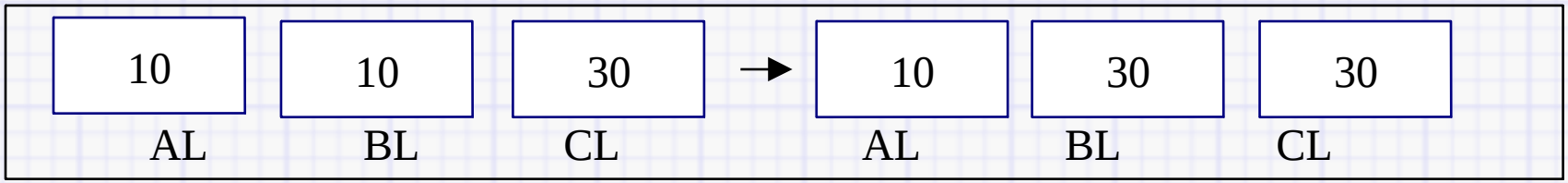
C	O	Z	S	P	A
M	M	M	M	M	M

Aritmética

Sintaxe:

CMPXCHG OP_DEST, OP_FONTE - Compara e permuta

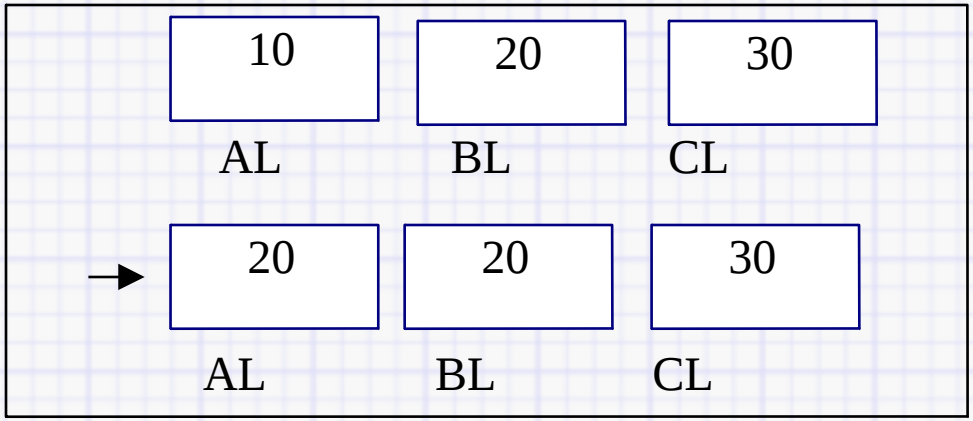
Se OP_DEST = AL, então OP_DEST := OP_FONTE
caso contrário AL := OP_DEST



EX.: CMPXCHG BL,CL

CMPXCHG CX,[0300H]

CMPXCHG EDX,EBX



Aritmética

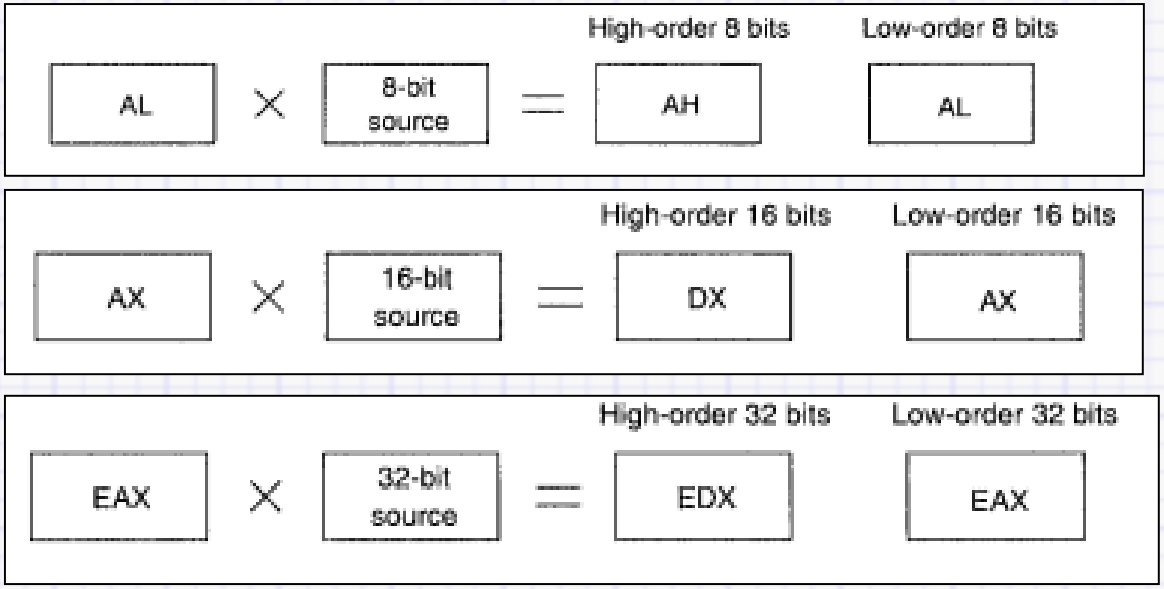
Sintaxe:

MUL OP_FONTE - multiplicação sem sinal

USO:

OP_FONTE = registrador, memória

C	O	Z	S	P	A
M	M	*	*	*	*



EX.: MUL BL

MUL DX

MUL ECX

C e O são "setados" se os registradores que guardam os bits de alta ordem são utilizados

Aritmética

C	O	Z	S	P	A
M	M	*	*	*	*

Sintaxe:

IMUL OP_FONTE - multiplicação sinalizada

USO:

OP_FONTE = registrador, memória (existem outras formas)

EX.: IMUL BL

IMUL DX

IMUL ECX

C e O são “setados” se os registradores que guardam os bits de alta ordem não são a extensão do sinal

Aritmética

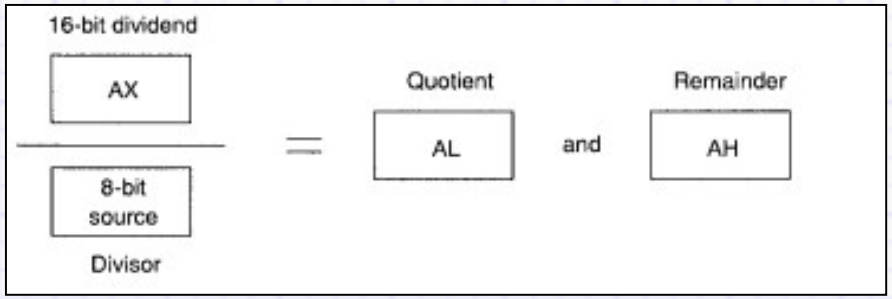
Sintaxe:

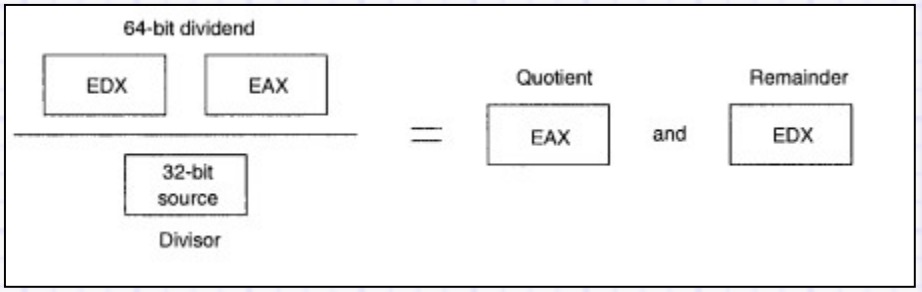
DIV OP_FONTE - divisão (sem sinal)

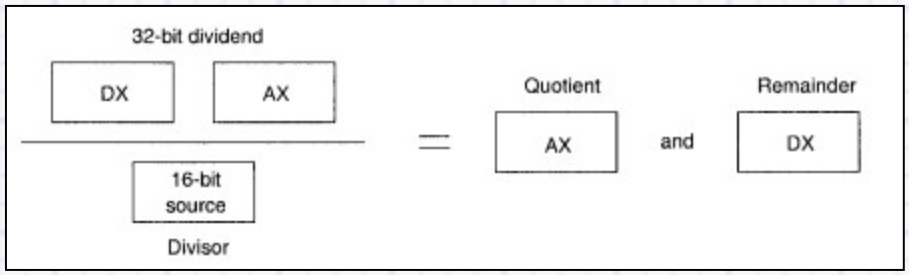
Uso:

OP_FONTE = registrador, memória

C	O	Z	S	P	A
*	*	*	*	*	*







EX.: DIV BL

DIV DX

DIV ECX

Aritmética

Sintaxe:

IDIV OP_FONTE - divisão sinalizada

Uso:

OP_FONTE = registrador, memória

C	O	Z	S	P	A
*	*	*	*	*	*

EX.: IDIV BL

IDIV DX

IDIV ECX

Aritmética

C	O	Z	S	P	A
M	M	M	M	M	M

Sintaxe:

NEG OP-DEST - complemento de dois

USO:

OP-DEST = registrador, memória

EX.: NEG AL

NEG [0200H]

NEG DX

Aritmética

C	O	Z	S	P	A
–	–	–	–	–	–

Sintaxe:

CBW - converte b*yte* de AL em palavra extendendo-se o sinal (em AH)

CWD - converte *a* palavra *d*e AX em dupla-palavra estendendo-se o sinal (DX:AX)

CWDE - converte a palavra de AX em dupla-palavra estendendo-se o sinal (EAX)

CDQ - converte *a* dupla-palavra *d*e EAX em quádrupla-palavra estendendo-se o sinal (EDX:EAX)

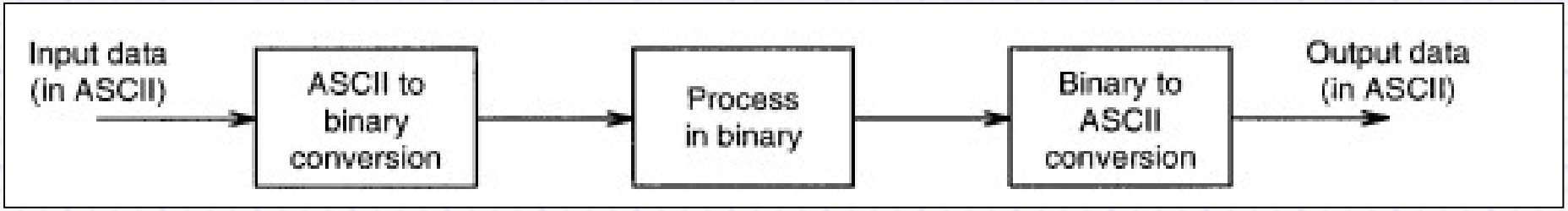
EX.: CBW CWDE

CWD CDQ

ASCII & BCD

ASCII - American Standard Code for Information Interchange

Ex: 1234
31 32 33 34H



Formas de representação números decimais: ASCII e BCD

Unpacked BCD: 01 02 03 04H (Sinal pode ser presentado por 1 byte adicional:80H)
Packed BCD: 12 34H

Não existe suporte para multiplicação e divisão para Packed BCD

ASCII & BCD

Sintaxe:

AAA - ajuste ASCII para adição

```
EX.:    MOV AX,0033H   ; 33h é '3' em ASCII
MOV BL,39H             ; 39h é '9' em ASCII
ADD AL,BL              ;AL=6CH
AAA          ;AX=0102H ; AL = 2 e AH = 1, representando BCD 12
```

Note que o resultado não é um caracter ASCII
Mas é BCD

ASCII & BCD

Sintaxe:

AAS - ajuste ASCII para subtração

AAM - ajuste ASCII para multiplicaçãoo

AAD - ajuste ASCII para divisão

ASCII & BCD

Sintaxe:

DAA - ajuste decimal para adição (BCD compactado)

```
EX.: MOV AL,15H
     MOV BL,25H
     ADD AL,BL          ;AL=3AH
     DAA                ;AL=40H
```

ASCII & BCD

Sintaxe:

DAS - ajuste decimal para subtração (BCD-compactado)

EX.:

```
MOV AL,10H
MOV BL,02H
SUB AL,BL          ;AL=0EH
DAS          ;AL=08H
```

ASCII & BCD

Sintaxe:

XADD OP_DEST, OP_FONTE

OP_FONTE := OP_DEST (ATRIBUIÇÃO DEPOIS DA SOMA)

OP_DEST= OP_FONTE + OP_DEST

Uso:

OP_FONTE = registrador; O_DEST=registrador, memória

EX.:

MOV AX, 2000H

MOV BX, 3000H

XADD AX, BX

Lógica e Manipulação de Bits

Instruções Lógicas

- **NOT**
- **AND**
- **OR**
- **XOR**
- **TEST**

Algumas Instruções Manipulação de bit

- **SHL/SAL - desloc. esquerda**
- **SHR - deslc. direita**
- **SAR**
- **ROL - rotação**
- **ROR**
- **RCL**
- **RCR**

Lógica e Manipulação de Bits

Sintaxe:

NOT OP_DEST - complemento de um

Uso:

OP_DEST = registrador, memória

EX.:

NOT AL

NOT [0200H]

NOT DX

Lógica e Manipulação de Bits

C	O	Z	S	P	A
0	0	M	M	M	*

Sintaxe:

AND OP_DEST,OP_FONTE - operação lógica e

Uso:

Registrador, registrador

Memória, registrador

Registrador, memória

Registrador, Imediato

Memória, Imediato

EX.:

AND AL,CH

AND CX,[SI]

AND DX,FFFFH

Lógica e Manipulação de Bits

C	O	Z	S	P	A
0	0	M	M	M	*

Sintaxe:

OR OP_DEST,OP_FONTE - operação lógica o*u*

Uso:

Registrador, registrador

Memória, registrador

Registrador, memória

Registrador, Imediato

Memória, Imediato

EX.:

OR AL,00H

OR CX,[SI]

OR DX,FFFFH

Lógica e Manipulação de Bits

C	O	Z	S	P	A
0	0	M	M	M	*

Sintaxe:

XOR OP_DEST,OP_FONTE - operação lógica *ou exclusivo*

Uso:

Registrador, registrador

Memória, registrador

Registrador, memória

Registrador, Imediato

Memória, Imediato

EX.:

XOR AL,CH

XOR CX,[SI]

XOR DX,DX

Lógica e Manipulação de Bits

C	O	Z	S	P	A
0	0	M	M	M	*

Sintaxe:

TEST OP_DEST,OP_FONTE **- operação lógica e**

Uso:

Registrador, registrador

Memória, registrador

Imediato, registrador

Imediato, memória

EX.:

TEST AL,CH

TEST CX,[SI]

TEST DX,FFFFH

Lógica e Manipulação de Bits

Sintaxe:

SETcc OP_DEST - atribui 01 a operando destino se a condição testada for verdadeira. Caso contrário, atribui-se 00.

USO:
OP_DEST = registrador, memória

EX.:

SETZ AL

SETLE AX

SETNC DX

Lógica e Manipulação de Bits

C	O	Z	S	P	A
*	*	M	*	*	*

Sintaxe:

BSF/BSR OP_DEST,OP_FONTE - procura pelo 1°bit igual a um no operando fonte a partir do LSB ou MSB. A posição encontrada será informada em operando destino.

Uso:

OP_DEST = REG. de 16 ou 32 bits

OP_FONTE = deve ser REG. ou mem de 16 ou 32 bits

EX.: BSF EAX,EBX

Lógica e Manipulação de Bits

C	O	Z	S	P	A
M	–	–	–	–	–

Sintaxe:

(BIT TEST; BIT TEST AND COMPLEMENT/SET/RESET) -> Guarda o bit como estava no Carry Flag

BT/BTC,BTS,BTR OP_DEST,OP_FONTE - testa, complementa, s*eta* ou *reseta* um bit do operando destino especificado no opernado fonte.

Uso:

OP_DEST = deve ser REG. de 16 ou 32 bits

OP_FONTE = deve ser REG. ou mem de 16 ou 32 bits

EX.:

BT AX,BX

BTC AX,15

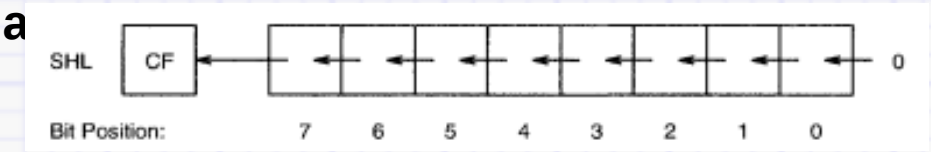
BTS AX,1

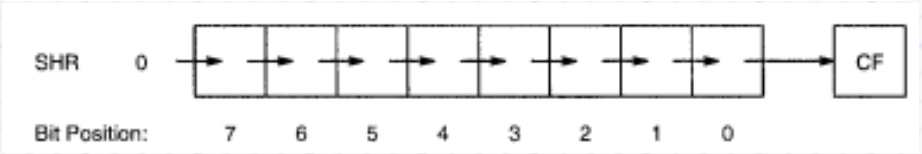
BTR AX,0

Lógica e Manipulação de Bits

Algumas Instruções Manipulação de bit

- **SHL/SAL - desloc. esquerda**
- **SHR - deslc. direita**
- **SAR**
- **ROL - rotação**
- **ROR**
- **RCL**
- **RCR**





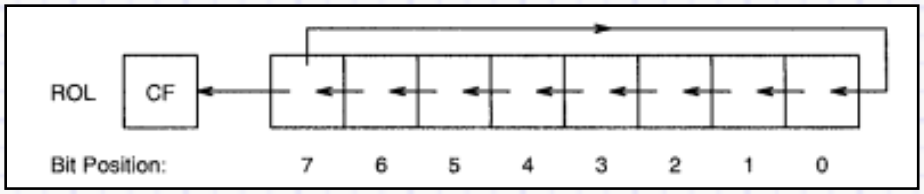
Instruction	Before shift AL or AX	After shift AL or AX	After shift CF
`shl AL,1`	1010 1110	0101 1100	1
`shr AL,1`	1010 1110	0101 0111	0
`mov CL,3` `shl AL,CL`	0110 1101	0110 1000	1
`mov CL,5` `shr AX,CL`	1011 1101 0101 1001	0000 0101 1110 1010	1

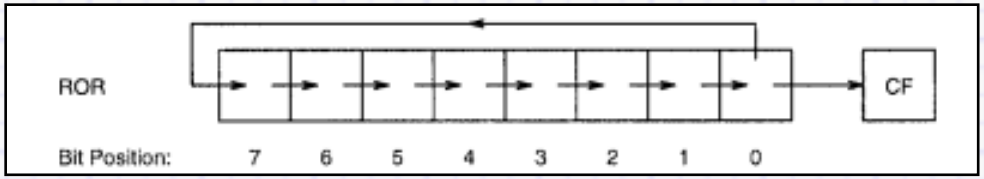
Lógica e Manipulação de Bits

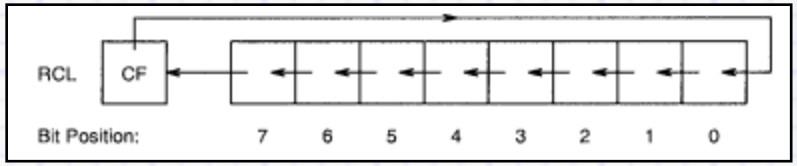
Algumas Instruções Manipulação de bit

- **SHL/SAL - desloc. esquerda**
- **SHR - deslc. direita**
- **SAR**
- **ROL - rotação**
- **ROR**
- **RCL**
- **RCR**

Instruction	Before execution AL or AX	After execution AL or AX	After execution CF
`rol AL,1`	1010 1110	0101 1101	1
`ror AL,1`	1010 1110	0101 0111	0
`mov CL,3` `rol AL,CL`	0110 1101	0110 1011	1
`mov CL,5` `ror AX,CL`	1011 1101 0101 1001	1100 1101 1110 1010	1



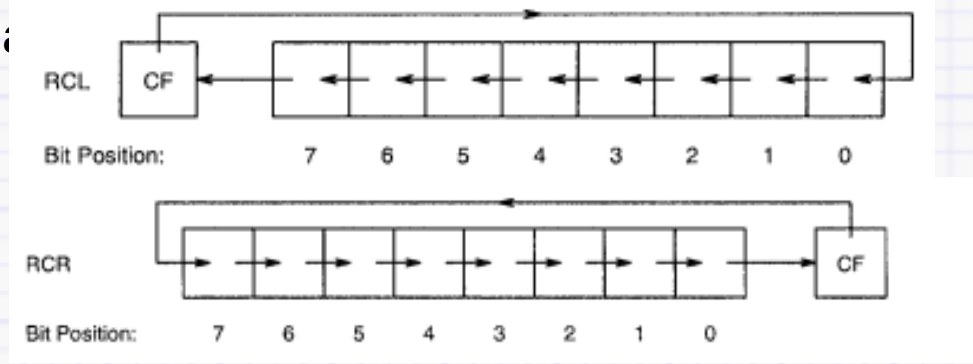




Lógica e Manipulação de Bits

Algumas Instruções Manipulação de bit

- **SHL/SAL - desloc. esquerda**
- **SHR - deslc. direita**
- **SAR**
- **ROL - rotação**
- **ROR**
- **RCL**
- **RCR**



Instruction	Before execution		After execution	
	AL or AX	CF	AL or AX	CF
`rcl AL,1`	1010 1110	0	0101 1100	1
`rcr AL,1`	1010 1110	1	1101 0111	0
`mov CL,3` `rcl AL,CL`	0110 1101	1	0110 1101	1
`mov CL,5` `rcr AX,CL`	1011 1101 0101 1001	0	1001 0101 1110 1010	1

Laços, Desvios, Subrotinas e Interrupções

Laços e Desvios

- **JMP -desvio incondicional**
- **Jcc - desvio condicional**
- **JCXZ**
- **LOOP - laço**
- **LOOPE**
- **LOOPNE**

Subrotinas e Interrupções

- **CALL - subrotinas**
- **INT - interrupções**
- **INTO**
- **RET - retorno**
- **IRET**

Laços, Desvios, Subrotinas e Interrupções

Laços e Desvios

- **Desvios fora do segmento requer ajuste de CS**
- **Às vezes necessário informar ao montador (assembler), se o jump vai ser "perto" (*intrasegment jump* – IP + deslocamento*)* ou "longo" (*intersegment jump – CS:segmento_alvo*;IP:deslocamento)**

Sintaxe:

JMP END_ALVO - **desvio incondicional**

EX.:

JMP FAR PTR XYZ

JMP [SI] ⟶ DS:0600
SI:0300

Memória

06300	34
06301	12

IP: | 12 | 34 |

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

JCC END_ALVO - **desvio condicional (*intrasegment jump*)**

EX.:

```
JCC MENOR_RES
...
MENOR_RES:
...
```

```
JNZ MENOR_RES
JMP FAR PTR XYZ
...
MENOR_RES:
...
```

Algumas formas:

jz – jump if equal (ZF=1)
jnz – jump if not zero (ZF=0)
jc – jump if carry (CF=1)
jnc – jump if not carry (CF=0)
jcxz – jump if cx=0
jecxz – jump if ecx=0

je – jump if equal (ZF=1)
jg – jump if greater (((SF xor OF) or ZF)=0)
jl – jump if less ((SF xor OF)=1)
jge – jump if greater or equal ((SF xor OF)=0)
jle – jump if less or equal (((SF xor OF) OR ZF)=1)
jne – jump if not equal (ZF=0)

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

LOOP/LOOPE/LOOPNE/LOOPZ/LOOPNZ OP_DEST – laço (*intrasegment jump*)

Descrementa o registrador contador (CX/ECX)

USO:

OP_DEST = deslocamento

EX:

```
MOV CX, 10
BACK: ...
        ...
      LOOP BACK
```

```
MOV CX, 5
OUTER: PUSH CX
            MOV CX,10
INNER:...
            ...
          LOOP INNER
            POP CX
            LOOP OUTER
```

Laços, Desvios, Subrotinas e Interrupções

- **CLC - Limpa CF**
- **STC - Faz CF=1**
- **CMC - Complementa CF**
- **CLD - Limpa DF**
- **STD - Faz DF=1**
- **CLI - Limpa IF**
- **STI - Faz IF=1**

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

CALL nome_procedimento – chamada de procedimento (empilha o endereço da próxima instrução – e segmento - imediatamente seguinte a chamada do procedimento)

EX:

CALL XYZ (empilha o endereço para ADD AX)

ADD AX

CALL FAR PTR XYZ (empilha o endereço da instr. e o segmento de código)

ADD AX

Obs: O procedimento precisa executar a instrução RET para finalizar sua execução

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

INT TIPO – interrupção de software

Empilha o registrador de flags,CS,IP

Depois do empilhamento do registrador de flags limpa trace flag e interrupt-enable flag

USO:

TIPO = um valor imediato de 8 bits (0 a 255)

EX:

INT 21H

Obs: A ISR precisa executar a instrução IRET para finalizar sua execução

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

INTO – gera interrupção 4 (INT 4)se flag de overflow estiver “setada”

EX:

INTO

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

HLT – Parada

Para o sistema. O processador fica neste estado até ocorrer um reset ou uma interrupção NMI ou INTR.

Laços, Desvios, Subrotinas e Interrupções

Sintaxe:

LOCK **- Instrução prefixo**

Bloqueia o barramento

EX.: LOCK XCHG

Sintaxe:

NOP **- No *operation***

Não executa nada

Strings

Instruções para tratamento de Strings

DS:SI – Aponta para o primeiro elemento da String fonte

ES:DI – Aponta para o primeiro elemento da String destino

Auxílio da manipulação utilizando **repeat** (instrução de prefixo)

Flag de direção (DF – bit 10 - indica como SI e DI são ajustados durante a execução de uma instrução

Strings

Sintaxe:

REP **- Instrução prefixo**

Repete enquanto CX/ECX for diferente de zero. Deve ser usada antes das instruções MOVS, STOS, INS e OUTS. CX/ECX é decrementado.

EX.: REP MOSVB

```
while (ECX ≠ 0)
    execute the string instruction;
    ECX := ECX–1;
end while
```

Strings

Sintaxe:

REPE/REPZ **- Instrução prefixo**

Repete enquanto CX/ECX for diferente de zero e ZF = 1. Deve ser usada antes das instruções CMPS e SCAS.

EX.: REPE CMPSB

```
while (ECX ≠ 0)
    execute the string instruction;
    ECX := ECX–1;
    if (ZF = 0)
    then
        exit loop
    end if
end while
```

Strings

Sintaxe:

REPNE/REPNZ - Instruções prefixo

Repete enquanto CX/ECX for diferente de zero e ZF = 0. Deve ser usada antes das instruções CMPS e SCAS.

EX.: REPNE SCASW

```
while (ECX ≠ 0)
    execute the string instruction;
    ECX := ECX–1;
    if (ZF = 1)
    then
        exit loop
    end if
end while
```

Strings

Sintaxe:

MOVS/ MOVSB/ MOVSW/ MOVSD - move

Move o elemento apontado por SI no segmento de dados para área apontada por DI no segmento de extra. DI e SI serão incrementados ou decrementados de 1, 2, 0u 4 dependendo do flag DF e do tipo do dado.

EX.:

```
MOV SI,0400
MOV AX,139F
MOV DS,AX
MOV DI,0410
MOV ES,AX
CLD
MOV CX, 6
REP MOVSB
```

```
MOV DX, 410
MOV AH, 09
INT 21H
```

Strings

Sintaxe:

CMPS/ CMPSB/ CMPSW/ CMPSD - compara

Compara o elemento apontado por DI no segmento extra com o elemento apontado por SI no segmento de dados. DI e SI serão incrementados ou decrementados de 1, 2, ou 4 dependendo do flag DF e do tipo do dado.

EX.:

```
MOV SI,0400
MOV AX,139F
MOV DS,AX
MOV DI,0410
MOV ES,AX
```

```
CLD
MOV CX, 6
REPZ CMPSB
```

Strings

Sintaxe:

SCAS/ SCASB/ SCASW/ SCASD - s*can*

Compara o elemento apontado por DI no segmento extra com o conteúdo do acumulador (AL,AX ou EAX). DI será incrementado ou decrementado de 1, 2, 0u 4 dependendo do flag DF e do tipo do dado.

EX.:

```
MOV DI,0402
MOV AX,139F
MOV ES,AX
MOV AL,62 ; ´b´ em ASCII
MOV CX,3
STD

REPNE SCASB
```

Strings

Sintaxe:

LODS/ LODSB/ LODSW/ LODSD - carrega

Carrega o elemento apontado por SI no segmento de dados no acumulador (AL,AX ou EAX). SI será incrementado ou decrementado de 1, 2, 0u 4 dependendo do flag DF e do tipo do dado.

EX.:

```
MOV SI,402
MOV AX,139F
MOV DS,AX
    STD
    LODSB
```

Strings

Sintaxe:

STOS/ STOSB/ STOSW/ STOSD - armazena

Armazena na área apontada por DI no segmento de extra o conteúdo do acumulador (AL,AX ou EAX). DI será incrementado ou decrementado de 1, 2, 0u 4 dependendo do flag DF e do tipo do dado.

EX.:

```
MOV DI,402
MOV AX,139F
MOV ES,AX
MOV AL,65 ; ´e´ em no padrão ASCII
    STD
    STOSB
```

Strings

Sintaxe:

INS – memoria,DX **input**

Memoria = ES:DI

OUTS - DX,memoria - output

Memoria = DS:SI

DI e SI são incrementados ou decrementados em 1,2,4 dependendo de DF

EX.:

INS DWORD PTR ES:[DI],DX (INSD)

INS BYTE PTR ES:[DI],DX (INSB)

OUTS DX,BYTE PTR DS:[SI], (OUTB)

Obs: Os operandos servem para indicar o tamanho do dado