CT215-6
Symbolic Instructions and Addressing

Objectives

- **Assembly language instructions for the Intel 80x86 processor family**
- **requirements for addressing data**
- **Instructions will be discussed:**
  - **MOV, MOVSX, MOVZX, XCHNG, LEA, INC, DEC, AND INT**

Symbolic Instruction Set

- **Arithmetic**
  - ADD, SUB, MUL, DIV, ADC, DEC, INC, ...
- **ASCII-BCD Conversion**
  - AAA, AAD, AAM, AAS, DAA, DAS
- **Bit Shifting**
  - SAL, SAR, SHL, SHR, RCL, ...
- **Comparison**
  - CMP, CMPSn, CMPXCHG, ...

Symbolic Instruction Set

- **Data transfer**
  - MOV, MOVSX, MOVZX, XCHG, …
- **Flag Operations**
  - CLC, CLD, CLI, PUSHF, POPF, …
- **Input/Output**
  - IN, INSN, OUT, OUTSN, …
- **Logical operations**
  - AND, NOT, OR, XOR, ...

Symbolic Instruction Set

- **Looping**
  - LOOP, LOOPE, LOOPZ, LOOPNE, …
- **Processor control**
  - ESC, HLT, LOCK, NOP, WAIT, …
- **Stack operations**
  - POP, POPA, PUSH, PUSHA, …
- **String operations**
  - CMPS, MOVS, REP, SANS, ...

Symbolic Instruction Set

- **Conditional transfer**
  - JE, JZ, JNZ, JNE, JO, …
- **Unconditional transfer**
  - CALL, INT, RET, JMP, RETN, …
- **Type conversion**
  - CBW, CDQ, CWD, CWDE, ...

Instruction Operands

[label]	operation	destination,source

- **Instruction may require one or more than one operand or none**
- **In the case of two operands**
  - Source -- second operand contains value or address of a register or in memory
  - Destination -- first operand contains value in a register or in memory

Examples

```
RETURN

XCHG AX,BX  ;exchange two 16-bit registers

MOV AL,BL   ;move 8-bit register to register

INC AL      ;increment 8-bit register

DEC BX      ;decrement 16-bit register

ADD AX,100H ;add immediate value to 16-bit reg

ADD AX,var1 ;add 16-bit memory to 16-bit reg

SUB BL,CL   ;subtract 8-bit register to reg
```

Basics Operand Types

- **Immediate**
  - constant
- **Register**
  - one of the CPU registers
- **Memory**
  - reference to a location in memory
  - There are six types of memory operands: direct, indirect-offset, register indirect, indexed, base-indexed, base-indexed with displacement

Register Operands

- **Any register of the CPU**

  Examples

```
MOV EAX, EBX
MOV CL, 20H
MOV SI, var1
MOV var1, AX
ADD AX, BX
```

Immediate Operands

- **A constant expression**

Examples

```
MOV AL, 10
MOV EAX, 12345678H
MOV DL, ‘X’
MOV AX, (40*50)
ADD BX, 10H
```

Direct Operands

- **Refer to the contents of memory at a location identified by a label in the data segment**

```
.DATA
Count DB 20
WORDL DW 1000H,2000H
.CODE
MOV AL, Count
MOV BX, WORDL + 2
```

Direct-Offset Operands

- **Access a list of values**

```
.DATA
Array DB 0AH,0Bh,0CH,0DH
.CODE
MOV AL, Array
MOV BL, Array+1
MOV CL, Array+2
MOV DL, Array+3
```

More Example

```
.DATA
Array DW 1000H,2000h,3000H,4000H
.CODE
MOV AX, Array
MOV BX, Array+2
MOV CX, Array+4
MOV DX, Array+6

MOV CX,DS:[38B0H];move word at DS+38B0H
INC BYTE PTR [1B0H];increment byte at
                   offset 1B0H
```

Indirect Operands

- **Uses the content of a register to form the effective memory address**
- **Registers used are: BX, DI, SI and BP**
  - **DS:BX**
  - **DS:DI**
  - **DS:SI**
  - **SS:BP**

Examples

```
.DATA
Dval DB 50
.CODE
MOV BX, OFFSET Dval;load BX with offset
MOV [BX],25         ;move 25 to Dataval
MOV [BX+2],0        ;move 0 to Dataval+2
MOV CL,[BX]         ;2nd operand=DS:BX
MOV BYTE PTR [DI],25;1st operand=DS:DI
ADD [BP],CL         ;1st operand=SS:BP
MOV [BX+DI],0      ;index DS:BX+DI
```

OFFSET Operator

- **OFFSET operator returns the 16-bit offset of a variable**

```
.DATA
WordA DW 1234H
.CODE
MOV BX,OFFSET WordA ;BX = 0000
```

Register Addressing

- Never mix registers of different sizes

```
MOV AL, BX
```

- No segment to segment register move

```
MOV  DS, SS
```

- CS register may not be used as a destination

```
MOV  CS, AX
```

- If source is a memory location, destination can’t be a segment register

```
MOV DS, WORDA
```

PTR Operator

```
.DATA
Var1 DB 22H
Var2 DW 2672H
.CODE
MOV  AH, BYTE PTR Var2  ;Move first byte
MOV  BYTE PTR VAR2, 05  ;Store 5 in first
                         byte of Var2
MOV  AX, WORD PTR Var1  ;Move word at offset
                         Var1
```

MOV Instruction

[label]	**MOV**	destination,source

MOV instruction transfers (copies) a word or a byte from the address of the second operand to the address of the first operand

MOV instruction

MOV	**Move (Byte or Word)**:
	MOV *dest, src*
	Transfers a byte or a word from the source operand to the destination operand.
	Instruction Operands:
	MOV mem, accum
	MOV accum, mem
	MOV reg, reg
	MOV reg, mem
	MOV mem, reg
	MOV reg, immed
	MOV mem, immed
	MOV seg-reg, reg16
	MOV seg-reg, mem16
	MOV reg16, seg-reg
	MOV mem16, seg-reg
		(dest)←(src)	AF –
			CF –
			DF –
			IF –
			OF –
			PF –
			SF –
			TF –
			ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Examples

Register Moves

```
MOV BX,AX   ;register to register
MOV DS,BX   ;register to segment register
MOV Var1,CX ;register to memory direct
MOV [DI],CX ;register to memory indirect
```

Immediate Moves

```
MOV AX,004CH   ;immediate to register
MOV Var1,2C    ;immediate to memory direct
MOV Var1[BX],2C;immediate to memory indirect
```

Examples

Direct Memory Moves

```
MOV BX,Var1     ;memory to register direct
MOV CX,Var1[CX];memory to register indirect
```

Segment Register Moves

```
MOV AX,DS   ;segment register to register
MOV Var1,DS ;segment register to memory
```

MOVE and FILL Instruction

[label]	**MOVSX/MOVZX**	destination,source

MOVSX/MOVZX instruction transfers (copies) a word or a byte from the second operand to a double word or a word of the first operand

- **MOVSX --** move and fill the signed bit
- **MOVZX --** move and fill the zero bits

Examples

```
MOVSX BX,10110000B ;CX=11111111 10110000
MOVZX BX,10110000B ;CX=00000000 10110000

MOV   BL,22H
MOVZX AX,BL  ;AX=0022H

.data
Var16 DW 1234H
.code
MOVZX EDX,VAR16   ;EDX=00001234H
MOVSX EDX,VAR16   ;EDX=FFFF1234H
```

XCHG Instruction

[label]	**XCHG**	reg/mem,reg/mem

XCHNG instruction exchanges the contents of two registers, or the contents of a register and a variable.

```
XCHG register,register
XCHG register,memory
XCHG memory,register
```

XCHG Instruction

XCHG	**Exchange**: XCHG *dest, src* Switches the contents of the source and destination operands (bytes or words). When used in conjunction with the LOCK prefix, XCHG can test and set a semaphore that controls access to a resource shared by multiple processors. **Instruction Operands**: XCHG accum, reg XCHG mem, reg XCHG reg, reg	(temp) ← (dest) (dest) ← (src) (src) ← (temp)	AF – CF – DF – IF – OF – PF – SF – TF – ZF –

NOTE: The three symbols used in the Flags Affected column are defined as follows:
– the contents of the flag remain unchanged after the instruction is executed
? the contents of the flag is undefined after the instruction is executed
✓the flag is updated after the instruction is executed

Examples

```
XCHG AX,BX ;exchange two 16-bit registers
XCHG AH,AL ;exchange two 8-bit registers
XCHG VAR1,BX ;exchange 16-bit memory with
             register
XCHG EAX,EBX ;exchange two 32-bit
             registers
```

```
        PAGE 60, 123
TITLE   EXACHANGE TWO VARIABLES
;---------------------------------------
        .MODEL SMALL
        .STACK 100H         ;define stack
        .DATA               ;define data
Value1  DB      0AH
Value2  DB      14H
;---------------------------------------
        .CODE               ;define code segment
main    PROC    FAR
        MOV     AX,@data    ;set address of data
        MOV     DS,AX       ;  segment in DS

        MOV     AL,Value1 ;load AL
        XCHG    Value2,AL ;exchange AL with Value2
        MOV     Value1,AL   ;store AL into Value1

        MOV     AX,4C00H  ;end processing
        INT     21H
main    ENDP              ;end of procedure
        END     main      ;end of program
```

LEA Instruction

[label]	**LEA**	register,memory

LEA instruction -- Load Effective Address loads a register with the offset of a data label.

```
LEA SI,ARRAY
MOV SI,OFFSET ARRAY
```

Examples

```
Assume Array is located at 01B0H
.data
Array DW 1234H,5678H
.code
MOV BX,2
LEA SI,Array[BX] ;SI=01B2H
MOV AX,[SI]      ;AX=5678H
```

INC and DEC Instructions

[label]	**INC/DEC**	register/memory

- **Increment or decrement by one**

```
INC AX
DEC Var1
```

Extended Move Operations

- **Move a block of bytes from one memory area to another memory area**

```
       PAGE 60,132
TITLE  A06MOVE (EXE) Extended move operation
;-----------------------------------------------------------------
       .MODEL SMALL
       .STACK 64
       .DATA
Headg1 DB 'InterTech'
Headg2 DB 'LaserCorp', '$'
;-----------------------------------------------------------------
        .code
Main    PROC    FAR
        MOV     AX,@data  ;initialize segment
        MOV     DS,AX     ;  register
        MOV     ES,AX

        MOV     CX,09     ;initialize to move 9 characters
        LEA     SI,Headg1 ;initialize address of Headg1
        LEA     DI,Headg2 ;        and Headg2
A20:
        MOV     AL,[SI]   ;get a character from Headg1
        MOV     [DI],AL   ;  move it to Headg2
        INC     SI        ;increment next char in Headg1
        INC     DI        ;increment next position in Headg2
        DEC     CX        ;decrement count for loop
        JNZ     A20       ;count not zero? Yes, loop finished

        MOV     AH,09H    ;request display
        LEA     DX,Headg2 ;   of Headg2
        INT     21H

        MOV     AX,4C00H  ;end processing
        INT     21H
Main    ENDP              ;end of procedure
        END     Main
```

Stack Operations

The Stack -- a special storage area in memory used by a program to temporary store data and pass parameter between functions.

A stack is a Last-In-First-out (LIFO) data structure.

Only two operations can be used to add or remove items from the stack: PUSH and POP

SS - - contains the initial address of the stack.

SP contains the offset of the top of the stack.

SP points to the top of the stack.

Stack Operations

- Assume the stack was declared to be of size 16 bytes

```
.STACK    16
```

- Then, the initial value of SP is 16 (i.e. 10H). Initially SP points 1 byte past the top of the stack.
- Stack operations Syntax

Stack Operations

```
PUSH    Source
```

- Source can be a **16-bit** general register, a segment register, or a memory location. Source type must be a word.

```
POP    Destination
```

- Destination can be a **16-bit** general register, a segment register other than CS, or a memory location. Destination type must be a word.

Example

```
.STACK 16

MOV     AX, 4A9FH
MOV     BX, 372CH
PUSH    AX
PUSH    BX
POP     DX
POP     CX
```

COM Programs

Program Size

COM

- maximum 64K (including 256-byte PSP)
- no 512-byte header record when stored on disk

Segments

- data, stack, and code in one (64k) segment
- stack segment in a COM program is automatically generated

Initialization for COM Program

- **All four segment registers are automatically initialized with PSP address**
- **Addressing begins at address 100H after .CODE directive, need the directive:**

```
ORG 100H
```

Converting EXE to COM

Source program must be coded according to COM requirements.

- only CODE segment directive needed
- ASSUME directive
- ORG 100H ;start right after of PSP

COMMAND assemble and link a COM

```
TASM          prog1.asm prog1.obj
TLINK  /t     prog1.obj prog1.com
```

```
                PAGE  60, 132
TITLE           PROG1.COM
CODESG          SEGMENT PARA 'Code'
                ASSUME  CS:CODESG, DS:CODESG, SS:CODESG
                ORG   100H
                JMP   MAIN
FLDD            DW    175
FLDE            DW    150
MAIN            PROC  NEAR
                MOV   AX, FLDD
                …
                MOV   AX, 04C00H
                INT   21H
MAIN            ENDP
CODESG          ENDS
                END
```