SYMBOLIC INSTRUCTIONS AND ADDRESSING

CHAPTER FIVE

Outline

- Instructions set of the Intel processor
- Data Transfer Instructions
  - MOV, XCHG & LEA
- Arithmetic Instructions
  - INC, DEC, ADD & SUB
- Addressing Modes
- The INT instruction

Instruction sets of Intel processors

- Arithmetic
- ASCII-BCD conversion
- Bit Shifting
- Comparison
- Data Transfer Flag
- Operations
- Input/output
- Logical Operations
- Looping
- Processor Control
- Stack Operations
- String Operations
- Transfer (conditional) operations
- Transfer (unconditional) operations
- Type Conversion

Instruction sets of Intel processors

Arithmetic

- ADC: Add with Carry
- ADD: Add Binary Numbers
- DEC: Decrement by 1
- DIV: Unsigned Divide
- IDIV: Signed(Integer) Divide
- IMUL: Signed(Integer) Multiply
- INC: Increment by 1
- MUL: Unsigned Multiply
- NEG: Negate
- SBB: Subtract with Borrow
- SUB: Subtract Binary Values
- XADD: Exchange and Add

Data Transfer

- LDS: Load Data Segment Register
- LEA: Load Effective Address
- LES: Load Extra Segment Register
- LODS: Load String
- LSS: Load Stack Segment Register
- MOV: Move Data
- MOVS: Move String
- STOS: Store String
- XCHG: Exchange

Instruction sets of Intel processors

LOGICAL OPERATIONS

- AND
- OR
- NOT
- XOR

LOOP OPERATIONS

- LOOP: Loop until Complete
- LOOPE: Loop While Equal
- LOOPZ: Loop While Zero

COMPARISON

- CMP: Compare
- CMPSn: Compare String
- TEST: Test Bits

STACK OPERATIONS

- POP
- PUSH
- POPF
- PUSHF …

Instruction sets of Intel processors

TRANSFER (CONDITIONAL)

- JA
- JAE
- JB
- JBE
- JC
- JCXZ
- JE
- JG
- JGE
- JL …

TRANSFER (UNCONDITIONAL)

- CALL
- INT
- IRET
- JMP
- RET
- RETN/RETF

Data Transfer Instructions: The MOV instruction

- Syntax
  - MOV destination, source
    - It copies the content of the source operand to the destination
    - Source
      - can be memory variable , register or immediate data
    - Destination
      - can be memory variable and register
- Note
  - The size of the source and destination operand has to be equal
  - Moving the content of one memory variable to another is illegal
  - Moving the content of segment register to another segment register is also an illegal move

Data Transfer Instructions: Valid MOV Operations

- **Register Moves**
  - MOV EDX, ECX ;Register-to-register
  - MOV DS, CX ;Register-to-segment register
  - MOV NUM1, DH ;Register-to-memory, direct
  - MOV [DI], DX ;Register-to-memory, indirect
- **Immediate Moves**
  - MOV CX, 4Ch ;Immediate-to-register
  - MOV NUM1, 21 ;Immediate-to-memory, direct
  - MOV NUM2[BX], 24h ;Immediate-to-memory, indirect
- **Direct Memory Moves**
  - MOV CH, NUM1 ;Memory-to-register, direct
  - MOV CX, NUM2 ;Memory-to-register, indirect
- **Segment Register Moves**
  - MOV BX, DS ;Segment register-to-register
  - MOV NUM2, DS ;Segment register-to-memory

Data Transfer Instructions: The XCHG Instruction

- Use
  - Swaps two data items
- Syntax
  - XCHG register/memory , register/memory
- Note
  - The size of the two operands has to be equal.
  - Exchanging the content of one memory variable to another is illegal.
- Ex.

```
X DB 31h
XCHG AX, X
XCHG AX, BX
```

Data Transfer Instructions
The LEA Instruction

- Use
  - To load the offset address of memory variable to registers
  - A common use for LEA is to initialize an offset in BX, DI, or SI for indexing an address in memory
- Syntax
  - LEA reg , mem
- Example

```
msg db "Hello",'$'
.code
    LEA DX, msg        ;loads the address of msg in dx.
```

  - you can also use the mov instruction with the offset keyword to load an offset address
    - MOV DX, OFFSET Msg

Arithmetic Instructions

- **The INC and DEC Instructions**
  - Format
    - INC/DEC register/memory
  - Depending on the result, the operations clear or set the
    - OF (carry into the sign bit, no carry out),
    - SF (plus/minus), and
    - ZF (zero/nonzero) flags
  - Example
    - Given BL = 05H
      - INC BL
      - DEC BL

Arithmetic Instructions

- **The ADD and SUB Instructions**
  - Format
    - ADD/SUB register/memory , register/memory/immediate
  - Valid operations involve
    - Register to/from register
    - Register to/from memory
    - Register - Immediate, and
    - Memory - Immediate
  - Flags affected are AF, CF, OF, PF, SF, and ZF
  - A zero result sets ZF, and a negative result sets SF

Addressing Modes

- Addressing modes refer to the different methods of addressing the operands
- The x86 instructions use eight different operand types: registers, immediate, and six memory addressing schemes.

Register Addressing

Immediate Addressing

Memory Addressing

- Direct Memory Addressing
- Direct-Offset Addressing
- Indirect Memory Addressing
- Base Displacement Addressing
- Base-Index Addressing
- Base-Index Displacement Addressing

Addressing Modes

- **Immediate Addressing**
  - the operand is specified in the instruction itself
    - MOV AX, 0245H ;Immediate to register(Word)
    - MOV AL, 0245H ;Invalid move
    - MOV AX, 48H ;Valid Move
- **Register Addressing**
  - operands are specified using registers
    - ADD AX , BX
    - MOV BX, DX
  - is the fastest type of operations, because processing data between registers involves no reference to memory

Addressing Modes

- **Direct Memory Addressing**
  - address of the operand is directly specified in the instruction
  - E.g
    - MOV AX, X ; where X refers to a word space in memory
    - MOV AX, [1592H]
  - DS is the default segment register for addressing data in memory as DS:offset

Addressing Modes

- **Indirect Memory Addressing**
  - allows data to be addressed at any memory location through an offset address held in BP/BX/DI /SI
    - BP is used with SS
    - BX, DI & SI are used with DS
- Example

```
X    DB  50
Y    DW  300
...
LEA BX, X
LEA SI, Y
ADD CL, [BX]        ;second operand DS:BX
MOV CX, [SI]        ;first operand CL=DS:BX , CH=DS:BX+1
ADD [BP], CL        ;first operand SS:BP
```

Addressing Modes

- **Direct-Offset Addressing**
  - Uses arithmetic instruction to modify an address
  - Example
    - X DB 12,15,16…
    - Y DW 163,227,485…

    - MOV CL, x[2] or MOV CL, x+2 ;move the third byte
    - MOV CX, y[4] or MOV CX, Y+ 4 ;move the fifth word

Addressing Modes

- **Base Displacement Addressing**
  - the offset address of the operand is given by the sum of contents of the BX/BP/SI/DI registers and 8-bit/16-bit displacement
  - Ex

```
X   DB   365 dup(?)
…
LEA   BX, X
….
ADD   CL,  [DI +12]          ;DS: (DI + 12)
MOV   [BX +2], 0             ;DS: (BX + 2)
SUB   X[SI],  25             ;DS:X[SI]
MOV   X[DI], DL              ;DS:X[DI]
```

Addressing Modes

- **Base-Index Addressing**
  - the offset address of the operand is computed by summing the contents of base register to the contents of an Index register
  - Example
    - MOV AL , [BX+SI] ;DS:[BX + SI]
- **Base-Index Displacement Addressing**
  - the operands offset is computed by adding the base register contents with Index registers contents and 8/16-bit displacement
  - Example
    - MOV AL , [BX+SI+2] ;DS:[BX + SI + 2]

The Segment Override Prefix

- The processor automatically selects the appropriate segment when addressing:
  - CS:IP for fetching an instruction
  - DS:offset for accessing data in memory, and
  - SS:SP for accessing the stack
- How do we handle data that is subject to another segment register? Such as ES, FS, or GS

The Segment Override Prefix

- Let ES be the other segment and BX contains an offset address within that segment
- Example
  - MOV DX , ES:[BX]
    - The coding of "ES:" indicates an override operator that means "Replace the normal use of the DS segment register with that of ES"
  - MOV ES:[SI+36] , CL
    - moves a byte value from CL into this other segment, at an offset formed by the value in SI plus 36:

Interrupts

- 8086 microprocessors allow program execution to be interrupted by external signals or by special instructions embedded in the program code.
- When the microprocessor is interrupted, it stops executing the current program and calls a procedure which services the interrupt.
- At the end of the interrupt service routine the code execution sequence is returned to the original, interrupted program
- An interrupt can be generated by one of three sources
  - As a result of a processor state violation called an exception
  - By an external device requesting service (Hardware Interrupt)
  - As a result of executing the INT instruction (Software Interrupt)

Int 21h interrupts

- There are 12 BIOS interrupts and 9 DOS interrupts
- Interrupt procedures are called with the INT assembler instruction
- What you need to know?
  - The interrupt type
  - The format of the parameters
- The popular DOS interrupt is INT 21H
  - 02, int 21h
  - 09, int 21h
  - 01, int 21h
  - 3fh, int 21h
- Procedures are called by placing the function number in AH register

02, int 21h

- Usage
  - Lets programs to display a single character or value on the screen
- Pre-condition
  - Store the character to be displayed in DL register
- Steps
  1. Move 02 to AH register
  2. Call the Int 21h function
- Post-condition
  - Null
- Example
  - Mov dl, ‘a’ ; ‘a’ is the character going to be displayed
  - Mov ah,02
  - Int 21h

01, int 21h

- Usage
  - lets programs to accept a single character or value.
- Pre-condition
  - Null
- Steps
  1. Move the 01 to AH register
  2. Call the Int 21h function
- Post-condition
  - The input value from the user is automatically stored in AL register
- Example
  - mov ah,01
  - Int 21h

09, INT 21H

- Usage: used to display a string
- Pre-condition
  1. Store the string going to be displayed in the data segment.
  2. Load the starting address of the data segment in DS register
  3. Load the offset address of the memory variable that contains the string in DX register.
- Steps
  1. Move 09 to AH register
  2. Call the Int 21h function
- Post-condition
  - Null

09, INT 21h Example

- .DATA
  - msg db “Hello”, ‘$’
- .CODE
  - mov ax, @data
  - mov ds, ax
  - mov dx, offset msg ;equivalent to LEA dx, msg
  - mov ah,09
  - Int 21h