Lecture 8

Processor Instructions and Addressing

Text: Chapter 6

Assembly Language Statements

```
   [identifier]  OPERATION    operands
```

Examples of statements we have seen so far:

```
          MOV   AX,BX
          ADD   AX,Salary
```

The instruction operations are written using meaningful symbols called **mnemonics** (“MOV”, “ADD”)

The operands will vary depending on the particular instruction.

Categories of Operations (partial list!)

- Arithmetic
- Information movement
- Comparison
- Flow control
- Logical Operations
- Processor Control
- Stack Operations
- String Operations

Categories of Operands

- Register Operands

  The operand is one of the CPU registers, and is identified by its reserved name:

  ```
  MOV     AX,BX
  ```

- Immediate Operands

  The second operand is a constant value or expression whose length is determined by the length of the first operand

  ```
  MOV   AX,1996h
  MOV   AL,19h
  ```

- Direct Memory Operands

  One operand is a register, the other is a memory location (labeled with an identifier).

  ```
  MOV   AX,Salary
  MOV   Count,CX
  ADD   AX,DS:[1998h]
  INC   BYTE PTR [0045h]
  ```

- Indirect Memory Operands

  A register is loaded with the **address** of an operand, and then the register **alone** is used as an operand in an instruction.

```
MOV   BX,OFFSET SALARY
MOV   [BX],1234h
```

- Address Displacement (Indexing)

  Uses the SI and DI (index) registers. The contents of the index register are added to the offset

```
MOV   SI,4
MOV   AL,RateTable[SI]
```

RateTable	02	04	06	08	0A	0C	0E
RateTable	+0	+1	+2	+3	+4	+5	+6

Some introductory instructions

XCHG

Exchange the data values in the two operands. This eliminates the need for a temporary copy.

```
XCHG   AL,AH
XCHG   AX,SALARY
```

LEA

Load Effective (Offset) Address will load a register with the **address** of an operand. The address in that register can later be used to refer to the operand indirectly.

```
LEA   BX,SALARY
MOV   GROSS_PAY,[BX]
```

Important:
How is this different from

```
    MOV   GROSS_PAY,BX
```

INC and DEC

INCrement and **DEC**rement will add one and subtract one, respectively, from either a memory location or a register.

```
INC   AX
DEC   SALARY
```

```
        page   60,132
TITLE   P06MOVE (EXE)  Extended move operations
;---------------------------------------------
        .MODEL SMALL
        .STACK 64
;---------------------------------------------
        .DATA
NAME1   DB    'ABCDEFGHI'
NAME2   DB    'JKLMNOPQR'
;---------------------------------------------
        .CODE
BEGIN   PROC   FAR
        MOV   AX,@data ;Initialize segment
        MOV   DS,AX    ;  registers
        MOV   ES,AX

        MOV   CX,09    ;Init. to move 9 chars
        LEA   SI,NAME1 ;Init. address of NAME1
        LEA   DI,NAME2 ;   and NAME2
B20:
        MOV   AL,[SI]  ;Get character from NAME1,
        MOV   [DI],AL  ;  move it to NAME2
        INC   SI       ;Incr. next char in NAME1
        INC   DI       ;Incr. next pos'n in NAME2
        DEC   CX       ;Decrement loop count
        JNZ   B20      ;Count not zero? Yes, loop

        MOV   AX,4C00H ;Exit to DOS
        INT   21H
BEGIN   ENDP
        END    BEGIN
```

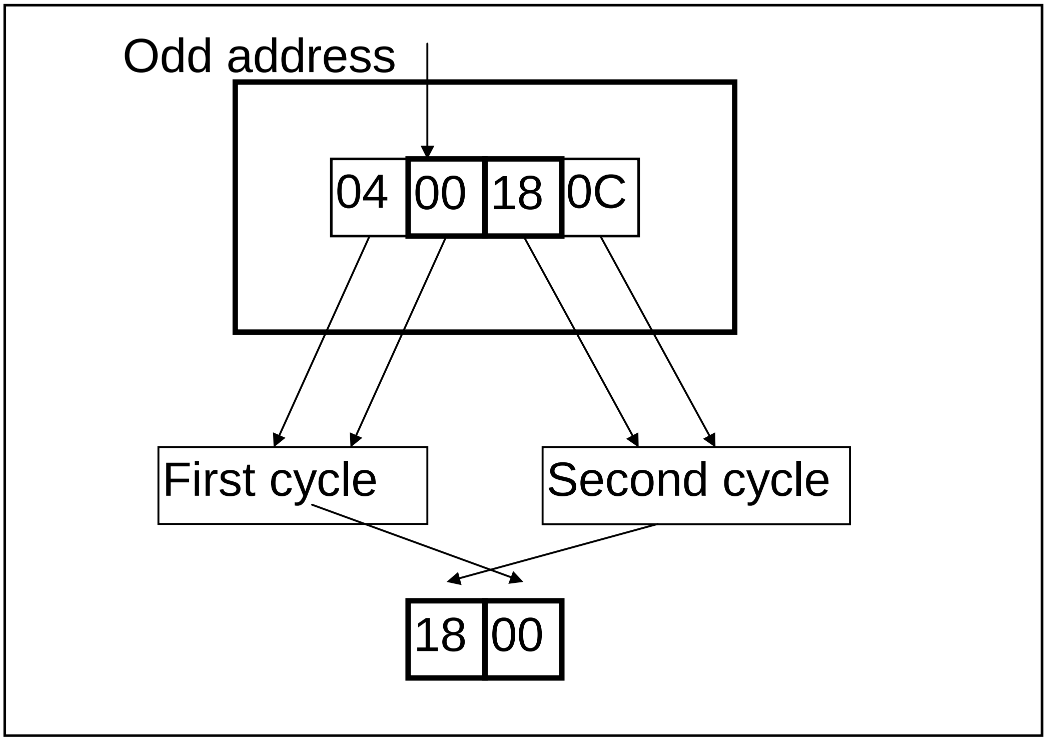
Example of MOV instructions with addressing

Some notes about addressing

Alignment

While a word may be any two bytes of memory, a word brought through the bus to or from the CPU must start with an even numbered byte.

Thus, loading a word whose first byte is at an odd address involves moving **two** words across the bus.



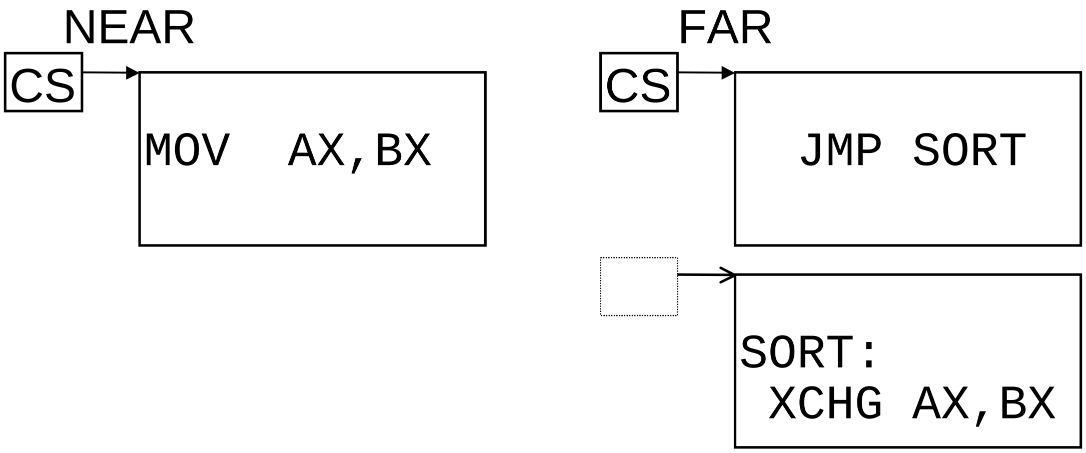
Processors with 32-bit data busses (80386 and higher) prefer addresses evenly divisible by four.

This does not cause errors; it only affects program performance. It can be resolved with the assembler's **ALIGN** directive.

Near and Far Addresses

A NEAR address is within the same segment and thus requires only an offset.

A FAR address is one which is in a different segment, so in addition to the offset, the segment address (in a segment register) is required.



Segment Override

Code usually comes from the code segment (CS:IP), and data from the data segment (DS:offset). An alternative segment can be given explicitly:

```
    MOV   AX,ES:[BX]
```

Here, the data will come from the Extra Segment.

Exercises - Lecture 8

Write a code segment that will add the first, third and fifth word of the array of words called "TheList" as declared in the data segment below. Do it by putting the address of "TheList" in the SI register, and writing ADD instructions that accumulate the answer in the AX register. For each instruction, you need to calculate the proper offset to be used with the [SI] register.

```
DATASG    SEGMENT    PARA
THELIST   DW         3,4,2,7,6,8,9
DATASG    ENDS

CODESG    SEGMENT    PARA
MAIN      PROC       FAR
          MOV        AX,DATASG
          MOV        DS,AX
          ________________________________
          ________________________________
          ________________________________
          ________________________________
          ________________________________
          ________________________________
          MOV        AX,4C00h
          INT        21h
MAIN      ENDP
CODESG    ENDS
          END        MAIN
```

Using the same data segment as above, write a code segment that will increase the value in the first word by one (use INC) and decrease the value in the last word by one (use DEC).

```
CODESG    SEGMENT    PARA
MAIN      PROC       FAR
          MOV        AX,DATASG
          MOV        DS,AX
          ________________________________
          ________________________________
          ________________________________
          ________________________________

          MOV        AX,4C00h
          INT        21h
MAIN      ENDP
CODESG    ENDS
```