CMSC 131

Microprocessor

- Manages arithmetic, logic, and control operations of the computer.

Machine Language Instruction

Each family processor has its own set of instructions

- Displaying information
- Keyboard operation
- Performing various tasks

Processor only understands machine language instructions which are string of 1s and 0s

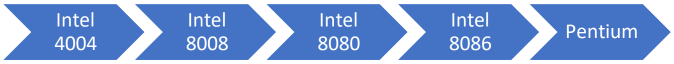
Software Development

- Too obscure and complex

Assembly Language

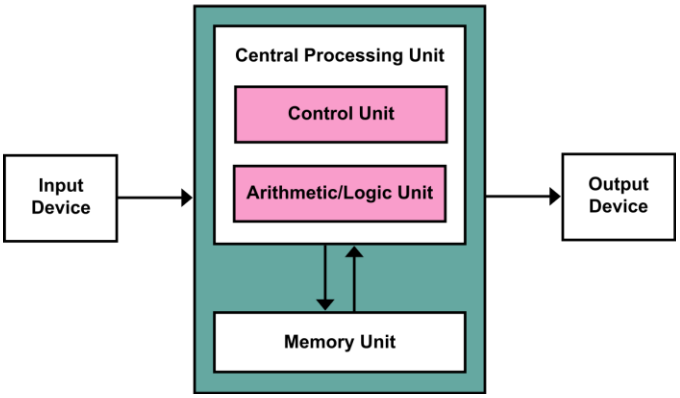
- Designed for a specific family of processors that represents various instructions in symbolic code and a more understandable form

Evolution of Microprocessors



- **Intel 4004 (1971)**
  - 4-bit microprocessor
  - 4 kB main memory
  - 45 instructions
  - First programmable device used for calculators
- **Intel 8008 (1972)**
  - 8-bit version of 4004
  - 16 kB main memory
  - 48 instructions
- **Intel 8080 (1973)**
  - 8-bit microprocessor
  - 64 kB main memory
  - 500,000 instructions / second
  - 10x faster than Intel 8008
- **Intel 8086/8088 (1978)**
  - 16-bit microprocessor
  - 1 MB main memory
  - 400 nanosecond clock cycle time
- **Pentium (1993)**
  - 32-bit microprocessor
  - 31-bit and 64-bit data bus
  - 4 GB main memory
  - Pro (1995), II (1997), III (1999), IV (2002)

Von Neumann Architecture



1. Data and instruction are stored in a single set of read-write memory
2. Contents of the memory are addressable by memory address, without regard to the type of data obtained
3. Execution occurs in a sequential fashion unless explicitly altered from one instruction to another

Computer System Components

- **Memory**
  - Stores instructions and data
- **Input / Output**
  - Peripherals for input and output instructions and data
- **Arithmetic Logic Unit**
  - Perform arithmetic operations and logical instructions

Myths on Assembly Language

- Assembly language is hard to learn
- Assembly is hard to read and understand
- Assembly is hard to write
- Assembly is hard to maintain
- Improved compiler technology has eliminated the need for assembly language

Advantages of Assembly Language

- **Speed**
  - Assembly language programs are generating the fastest programs around
- **Space**
  - Assembly language programs are often the smallest
- **Capability**

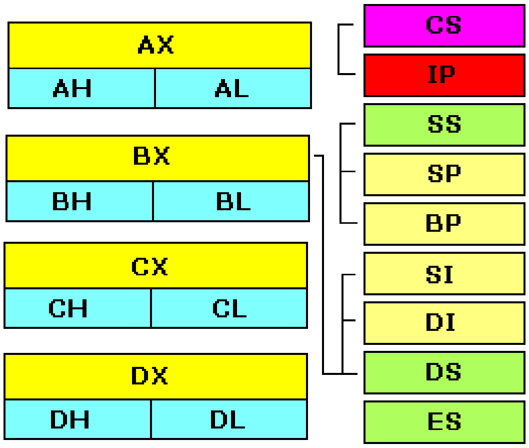
- You can do things in assembly language which are difficult or impossible in HLLs

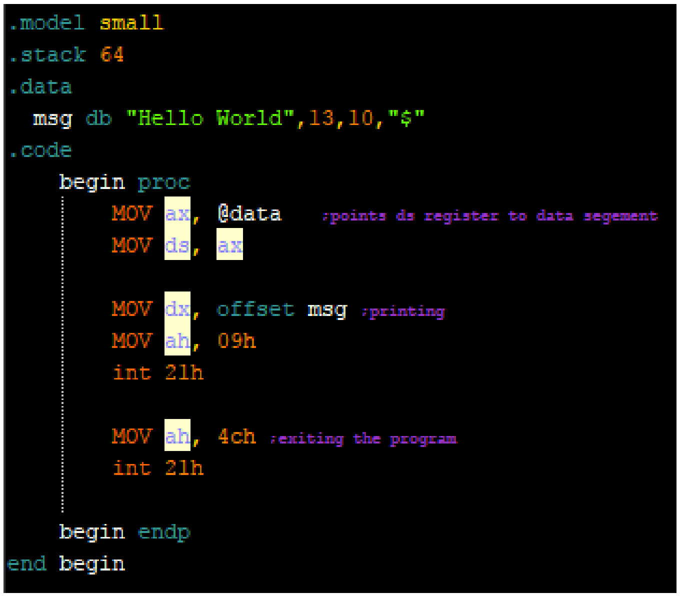
- **Knowledge**
    - Your knowledge of assembly language will help you write better programs, even when using HLLs

Registers

- Small, high speed memory locations inside the processor
- An excellent place to store variables
- Used to store temporary results and certain control information
- Take zero memory cycles to access

8086 Registers





```
.model small
.stack 64
.data
  msg db "Hello World",13,10,"$"
.code
    begin proc
        MOV ax, @data    ;points ds register to data segement
        MOV ds, ax

        MOV dx, offset msg ;printing
        MOV ah, 09h
        int 21h

        MOV ah, 4ch ;exiting the program
        int 21h

    begin endp
end begin
```

- Data used by the CPU is usually stored in registers before it is processed
- Most instructions in assembly often deal with register values

8086 Registers

- **Basic Groups**
    1. General Purpose Registers
    2. Pointer Registers
    3. Index Registers
    4. Flag Registers

1. General Purpose Registers

- May appear as operands of arithmetic, logical operations

- **Accumulator Register (AX)**
    - Where arithmetic and logical operations take place
- **Base Register (BX)**
    - Used to hold indirect addresses
- **Count Register (CX)**
    - Used to count iterations in a loop or specify members
- **Data Registers (DX)**
    - Holds overflow from certain arithmetic operations
    - Holds I/O addresses when accessing data

2. Segment Registers

- Used to point segments of memory

- **Code Segment (CS)**
    - Points to the beginning of the code segment where instructions are stored
- **Data Segment (DS)**
    - Points to the beginning of the segment memory that contains data for the program
- **Stack Segment (SS)**
    - Points to the area of memory that is used for all stack operations
- **Extra Segment (ES)**
    - Used by the programmer to set aside a portion of memory for some other use

PROGRAM SKELETON

1. Select a memory model
2. Define stack size
3. Declare variables
4. Write code
5. Mark the end of the source file

3. Pointer Registers

- **Instruction Pointer**
    - Used together with CS (pointing to where the code segment is)
    - Tells where in the code segment the next instruction to be executed is
    - You should not change the value of this register
- **Stack Pointer**
    - Maintains program stack
- **Base Pointer**
    - Used to address parameters passed to the subroutines on the stack

4. Index Registers

- SI & DI – Source and Destination index
    - May be used to access memory
    - Implicitly used for all string instructions

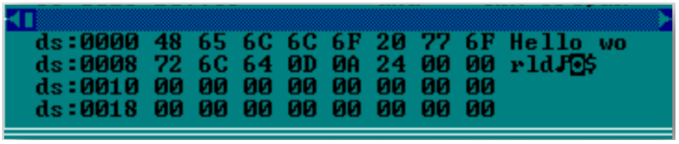
5. Flag Registers

- Often referred to as the "Program Status Register"
- Used by the processor as a series of 16 individual bits which keep track of certain conditions that happen when the PC is running
- Considered set when they are equal to 1
- Cleared when they are equal to 0

MEMORY

- Before 8086, programmers were only able to use 16 – bit values to access memory
- Memory [0] to Memory [65535]
- About 64K of memory
- Later, 8086 introduced *segment addressing.*

- **Segment Addressing**
    - It is used to access memory
        - Segment value
        - Offset value
    - Notation
        - SEGMENT: OFFSET



```
ds:0000 48 65 6C 6C 6F 20 77 6F Hello wo
ds:0008 72 6C 64 0D 0A 24 00 00 rld♪◙$
ds:0010 00 00 00 00 00 00 00 00
ds:0018 00 00 00 00 00 00 00 00
```

- Column 1: the memory location
- Column 2: what is stored in those memory location (in hexadecimal)
- Column 3: contains the equivalent of those hexadecimal values in ASCII
- So FFFF:FFF0 means
    - Segment FFFFh
    - FFF0 bytes from the beginning of the segment

DATA TRANSFER INSTRUCTIONS

MOV destination, source

- reg, reg
- mem, reg
- reg, mem
- mem, immed
- reg, immed

Example:

- **MOV ax, word1**
    - "Move word1 to ax"
    - Contents of the register are replaced by the contents of the memory location word1
- **XCHG ah, bl**
    - Swaps the contents of ah and bl
- **Illegal: MOV word1, word2**
    - Can't have both operators be memory locations

MOV INSTRUCTION

- Copy the contents of a source to a destination

EXCHANGE (XCHG)

- MOV and XCHG cannot perform memory to memory moves
- This provides an efficient means to swap the operands
    - No temporary storage is needed
    - Sorting often requires this type of operation
    - This works only with general registers

XCHG destination source

- reg, reg
- reg, mem
- mem, reg

ARITHMETIC INSTRUCTIONS

- operands must be same size

- source can be a general register, memory location or constant

- **ADD:** ADD dest, source
- **SUBTRACT:** SUB dest, source
- **INCREMENT:** INC dest
- **DECREMENT:** DEC dest
- **NEGATE:** NEG dest

ASSEMBLY PROGRAMMING

- Models for Assembly Programming
  1. Tiny
  2. Small
  3. Medium
  4. Compact
  5. Large
  6. Huge

- **TINY**
  - Code and data group combined into a single group called DGROUP
- **SMALL**
  - Code is in a single segment; code and data both smaller than 64k
- **MEDIUM**
  - Code uses multiple segment, one per module; code can be larger than 64k but data must be smaller than 64k
- **COMPACT**
  - Code is in a single segment; data can be more than 64k but code can't.
- **LARGE**
  - Code uses multiple segments; code and data can be more than 64k but arrays can't.
- **HUGE**
  - Code uses multiple segments; code, data, and arrays can be more than 64k

VARIABLES

- A memory location

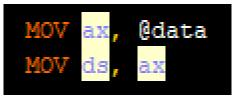
Declaration:

name db **value** (db – define byte)

name dw **value** (dw – define word)

- **Name**
  - Should start with a letter followed by a combination of letters and digits
- **Value**
  - Can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "?" symbol for variables that are not initialized

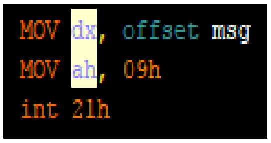
DATA PART



```
MOV ax, @data
MOV ds, ax
```

these lines initialize the ds register to point the portion of memory where we store our data

PRINTING



```
MOV dx, offset msg
MOV ah, 09h
int 21h
```

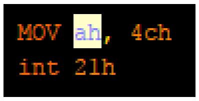
1. Stores the address of the variable <u>msg</u> in memory into the register dx
2. Assign service number
3. Call DOS interrupt 21h

INTERRUPTS

- Built-in subroutines
- Usually used to handle inputs and outputs to an assembly program

- **int 21h**
  - Screen inputs and outputs
  - Speaker beeping
  - Ending program
- **Service 09h**
  - Prints contents of memory pointed to by dx character until it sees a "$"
  - When int 21h is invoked, "Hello World" is printed letter by letter into the screen
- **13, 10**
  - Equivalent to "\n" in C or a new line
  - 10 is the ASCII character for new line with same column position
  - 13 is the ASCII character for carriage return (Moves the cursor at the start of the current line of the screen)

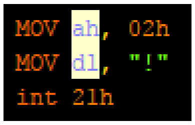
EXITING THE PROGRAM



```
MOV ah, 4ch
int 21h
```

without these two lines, the program will continue running which can cause the computer to crash

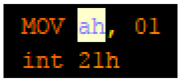
PRINTING A SINGLE CHARACTER



```
MOV ah, 02h
MOV dl, "!"
int 21h
```

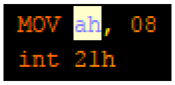
to print a single character to the screen, use service 02h

READING INPUT FROM THE USER



```
MOV ah, 01
int 21h
```

With echo – prints the input



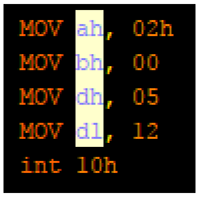
```
MOV ah, 08
int 21h
```

Without echo – does not print the input

SETTING THE CURSOR

- Screen is composed of 80 x 24 characters
- int 10h, service 02h
- BIOS interrupt that set the cursor
- Input: DH and DL

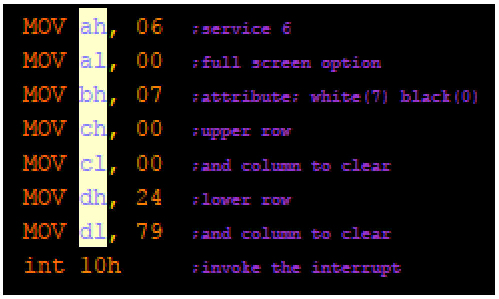
Example:



```
MOV ah, 02h
MOV bh, 00
MOV dh, 05
MOV dl, 12
int 10h
```

To set the cursor to row 5 column 12

CLEARING THE SCREEN



```
MOV ah, 06    ;service 6
MOV al, 00    ;full screen option
MOV bh, 07    ;attribute: white(7) black(0)
MOV ch, 00    ;upper row
MOV cl, 00    ;and column to clear
MOV dh, 24    ;lower row
MOV dl, 79    ;and column to clear
int 10h       ;invoke the interrupt
```