Today’s topics

**Machine Architecture**
The basic machine
Basic programming

**Upcoming**
Assembler programming

**Reading**
*Great Ideas*, Chapters 8
Computer Architecture

- **Definition of computer architecture**
  - The *programmer’s view of the computer hardware*

- **Hardware – Software Boundary**
  - Not well defined
  - Much hardware is programmed
  - Some hardware instantiates programming steps
  - An imbedded program that cannot be changed could be called hardware

- **Firmware**
  - Sometimes used to describe programming that is seldom changed
  - Typically stored in *read-only* memory (cannot change)
Basic Computer

- Extremely Primitive
  - Cannot understand any Java or English-like commands
  - There is no command to carry out the `while` statement
  - Make up in speed what it gives up in complexity

- Use a *translator* to transform program to machine’s native language
  - Called *compiler*
  - High-level language like Java called the *source* language
  - Target language is called *machine* language
  - Machine language is what the hardware responds to
Machine Language

- **Machine language is the most primitive**
  - Everything represented by numbers
  - At hardware level, numbers are in binary
  - Numbers represent *instructions* (code)
  - **AND** Numbers represent *data*
  - *Context* of use decides whether number is data or instruction

- **In practice, seldom program in machine language**

- **Use a language, very close to machine language called** **Assembler Language**
  - *Symbolic* in nature (as opposed to numeric)
  - Each instruction number has a mnemonic
  - E.g., *12 is ADD*
  - Locations also given names (sometimes *variable name*)
Architectural Features

- **Memory**

- **Central Processing Unit (CPU) seen as set of Registers**
  - IP: Instruction pointer
  - IR: Instruction Register
  - AX: Arithmetic Register/Accumulator
  - CF: Condition Flag
Simple Program

- Show in assembler rather than machine language
  
  ```assembly
  copy   ax, x
  add    ax, y
  copy   z, ax
  ```

- Implements
  
  ```
  z = x + y;
  ```

- Remember, really *ALL NUMBERS*
  
  - Could be:
    
    ```
    20  101
    12  102
    21  103
    ```
  
  - If `copy-into = 20, add = 12, and copy-out = 21` and
    
    `x` is stored in 101, `y` in 102, and `z` in 103
Fetch - Execute Cycle

- Clock systematically leads machine cycle thru steps
- FETCH
  - Get instruction from memory
    - IP register (also called program counter or PC) says *where from*
  - Increment IP (to point to *next* instruction)
- EXECUTE
  - Decode instruction
    - Figure out what is wanted (add?, copy? ...)
    - Extract memory address from instruction
    - If needed, get info from memory
  - Carry out instruction
    - I.e., add info to Accumulator (AX)
More Instructions

- **copy and add**
  - Implicit right to left movement
  - Most instructions involve accumulator (AX)

- **in and out**
  - Like getInt and setInt in Java
  - in goes from keyboard to AX
  - out goes from AX to screen

- **Go through another example -- program to perform:**

```java
{ 
    x = a.getInt();
    y = b.getInt();
    z = (x + y);
    c.setInt(z);
}
```
sum.as

0    in     ax
1    copy   x, ax
2    in     ax
3    copy   y, ax
4    copy   ax, x
5    add    ax, y
6    copy   z, ax
7    copy   ax, z
8    out    ax

20   x   0
21   y   0
23   z   0

Sample I/O:
<23
<16
>39

CompSci 001
More Instructions

- Need to handle Java if and while instructions
- Use \texttt{cmp} instruction
  - Compares values in AX and memory location
  - Sets carry flag (CF) to
    - \texttt{B} below (AX less than memory) or
    - \texttt{NB} not below (AX greater or equal to memory)
- Use \textit{jump} instructions to take advantage of this new info
  - \texttt{jnb} instruction jumps to new location if CF set to NB
  - \texttt{jb} instruction jump to new location if CF set to B
  - \texttt{jmp} always jumps, regardless of CF state
- Can now implement code involving if
Program to write out the larger of two numbers read in:

```
in ax
    copy r, ax
in ax
    copy s, ax
    copy ax, s
cmp ax, r
    jnb there
    copy ax, r
    out ax
    jmp quit
there
    copy ax, s
    out ax
quit
    halt
r    0
s    0
Sample I/O:   <33    <44    >44
```
Tracing

- Tracing is often the only way to figure out assembler programs
  - Number your statements (for reference)
    - Can also use actual memory addresses if known
  - Set up column heading for variables (memory) expected to change
  - Step through the program
    - You play to role of computer
    - Use notes and/or extra columns to keep track of
      - Input and output
      - State of the Condition Flags (CF)
  - Trace with test data
    - Until done or
    - Until program is understood