CS271 WEEK 1

—Lectures—

Lecture 1

Language Levels

- High: Java, C++, reasonably portable to different arch's
- Low: Intel Asm / Mac Asm (ie.) Architecture specific
- Machine Code: 1's and 0's. Lowest level
-

$$\begin{tikzcd}
\text{High} \arrow[r, bend left, "\text{compiler}"] & \text{low} \arrow[r, bend left, "\text{assembler}"] \arrow[l, bend left, "\text{decompiler}"] & \text{Machine} \arrow[r, "\text{Linker/Loader}"] \arrow[l, bend left, "\text{disassembler}"] & \text{Electronic Hardware exec.}
\end{tikzcd}$$

Assembly Language

- uses mnemonics for instructions "opcodes", such as MOV, JMP.
- Uses naming scheme for variables...
- Assembled into machine code for "local" architecture, or "cross-assembled" into Machine code for another arch.

Lecture 2

Bits & Bytes

- a bit is a <u>bi</u>nary dig<u>it</u> that can either be on (1) or off (0)
- A byte is a group of 8 bits

speeds & sizes

- speeds are in Xbps (i.e. Mbps, Kbps...)
  8 Mbps = 8 000 000 bits/second
- sizes use $K=2^{10}$ $M=2^{20}$ $G=2^{30}$ bytes
- Therefore a 200 GiB = 200 GB Hard drive has $200 \cdot 2^{30}$ bytes
  = 214 748 364 800 Bytes
  = 1 717 986 918 400 bits $\times 8$
- "B" ≃ "Bytes" ; "b" ≃ "bits"

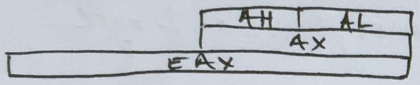
IA-32 Architecture

- what we will use in CS271
- 32 bit architecture. This means 2^{32} addressible memory locations, ≅ 4GiB
- Min memory size ~ 2 byte...
- 4 Processor modes:
  → Sys Mgmt: Full access (No protection of mem)
  → Real address: more locked down than Sys Mgmt; progs still have full access to mem ‖ 1MB
  → Protected Mode: Programs can only access memory in allotted memory segment ‖ 4GB
  → Virtual Mode: (within protected mode): simulates Real Addr within Protected Mode
- Two different processing units: float and int.
  ↳ these are quite different; focusing on int for now.
- IA-32 is CISC – it uses microprograms to implement machine instructions
- IA-32 is byte addressible – each byte of mem has its own addr

- IA-32 is Little Endian
- General Purpose registers
  → 32-bit, "E" for "extended" as the IA-32 is a decendent of 16-bit Machines
  → EAX, EBX, ECX, EDX
- Multi-Purpose registers
  → 32 bit
  → EBP, ESP, ESI, EDI
- Special Purpose Registers
  - EFL, EIP, both 32-bit

E[B,S]P and E[S,D]I can only be broken down into 16 bit sub-registers, i.e $\underset{32}{\text{ESI}} \leftrightarrow \underset{16}{\text{SI}}$

- Sub registers: work mainly for E[A,B,C,D]X



NOTE: Changing Any of the sub registers changes the value of EAX

MASM syntax
- starting program segment
  ↳ use .code, or .data for ex.
- Masm is not case sensitive
- Comments start with a ";"
- Program must include the "TITLE", "INCLUDE" and "END" Directives for CSE271
- Defining a variable must be:
  label type init_val (can be ?) ;comment
- Strings (name BYTE "..." $\underbrace{,0,}$) must be null terminated!!

Literals
- used to get data in your code (i.e. to include numbers/chars...).
- a "radix" (letter at end of a value) is used

- Binary Literal: 1001b
- Decimal Literal: 420
- Hexadecimal Literal: 0x4CF
- Characters Literals: 'z'

Opcodes
- such as MOV, JMP...
- can have either 0, 1, 2 "arguments":
  ↳ ~~[illegible]~~ Opcode
  Opcode ~~dest~~ source
  Opcode dest