Weeks 2 and 3. CPUs, Binary, ISAs, Register Overview.

CPUs

- Terms
  - Instruction
    - Primitive CPU command
    - Ex: moving data between registers, working with memory, primitive arithmetic operations
  - Machine Code
    - Code that the CPU directly processes. Each instruction is usually encoded by several bytes.
    - Cognitive science professor Douglas Hofstadter has compared machine code to genetic code, saying that "Looking at a program written in machine language is vaguely comparable to looking at a DNA molecule atom by atom."
      - Fun fact: Hofstadters Law was also defined in the same book this quote is sourced from.
        - "Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law."
  - Assembly Language
    - Mnemonic code and some extensions, like macros, that are intended to make a programmer's life easier.
  - CPU Register
    - A CPU register is a quickly accessible location available to a computer's central processing unit (CPU). Registers usually consist of a small amount of fast storage, although some registers have specific hardware functions, and may be read-only or write-only.
    - Each CPU has a fixed set of general purpose registers (GPR2). ≈ 8 in x86, ≈ 16 in x86- 64, and also ≈ 16 in ARM.
  - Von Neumann Architecture has these components:
    - A processing unit that contains an arithmetic logic unit and processor registers

    - A control unit that contains an instruction register and program counter
    - Memory that stores data and instructions
    - External mass storage
    - Input and output mechanisms
- Why don't we write in machine code? Why do we have higher level programming languages?
    - It is much easier for humans to use a high-level PL like C/C++, Java, or Python, but it is easier for a CPU to use a much lower level of abstraction.
    - It is very inconvenient for humans to write in assembly language, due to it being so low-level and difficult to write in without making a huge number of annoying mistakes.
    - A **compiler** converts high level programming language code into CPU readable machine code. More on that in a later week.

Numeral Systems

- We use a decimal numeral system for every day math and most human related activities.
- In digital electronics / engineering, binary is used: 0 for the absence of current in a wire, 1 for the presence of current in a wire.
- Converting Radix
    - Radix = Base
        - Base/radix for decimal numbers = 10
            - 43
                - $(10^1 \cdot 4) + (10^0 \cdot 3) = 43$
                - $(10 \cdot 4) + (1 \cdot 3) = 43$
        - Base/radix for binary numbers = 2
            - 101011
                - $(2^5 \cdot 1) + (2^4 \cdot 0) + (2^3 \cdot 1) + (2^2 \cdot 0) + (2^1 \cdot 1) + (2^0 \cdot 1) = 43$

- $(32 \cdot 1) + (16 \cdot 0) + (8 \cdot 1) + (4 \cdot 0) + (2 \cdot 1) + 1 = 43$

- Fill out this chart on the board, encouraging help from students, start with just a few from each column filled in.

Hexadecimal (Radix = 16)	Binary (Radix = 2)	Decimal (Radix = 10)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

- How does one identify which radix is being used?
  - Decimals are normally written as is: 1234 is just 1234, sometimes they are appended with the “d” suffix: 1234d
  - Binary is either prepended with the “0b” prefix: 0b101011, or the “b” suffix: 101011b
  - Hexadecimal can be done a few different ways, below are they ways to identify 1234ABCD as hexadecimal
    - “0x” prefix: 0x1234ABCD

- “h” suffix: 1234ABCDh
- “$” prefix: $1234ABCD

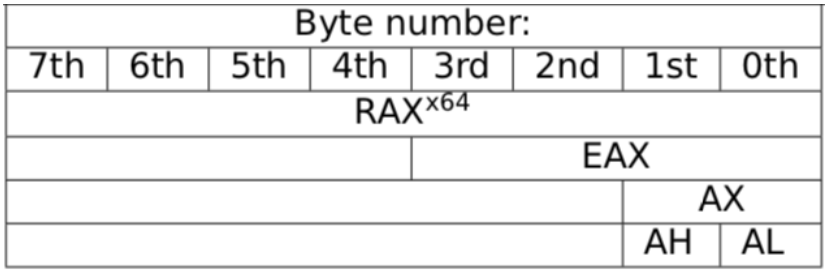
ISAs

- Instruction Set Architecture
  - An abstract model of a computer
  - Is the interface between **software** and **hardware**
    - Software that has been written for an ISA can run on different implementations of the same ISA
  - In general, ISAs define:
    - The supported data types
    - What state there is (such as the main memory and registers) and their semantics (such as the memory consistency and addressing modes)
    - The instruction set (the set of machine instructions that comprises a computer's machine language)
    - The input/output model.
  - ISAs allow us to have the same instruction set shared between vastly different processor designs.
    - Intel Pentium and AMD Athlon both support x86 ISA, but have different internal designs.
    - The concept of an architecture, distinct from the design of a specific machine, was developed by Fred Brooks at IBM during the design phase of System/360.
  - ISAs are typically classified by complexity, and the two most popular complexity variants include CISC and RISC
- The book covers MIPS, x86, and ARM.
  - CISC
    - Complex instruction set computer
    - Contains many specialized instructions, some of which may rarely or never used in practical programs

  - x86
- RISC
  - Reduced instruction set computer
  - Simplifies the processor by efficiently implementing only the instructions that are frequently used in programs
  - Less common operations are implemented as subroutines, having their resulting additional processor execution time offset by infrequent use
  - ARM, MIPS
- Course will focus on x86/x86-64

General Purpose Registers

This section can be glossed over in order to properly introduce the Tools Intro, come back to these notes when necessary.



- `Bit - atomic`
- `Byte - 8 bits`
- `WORD - 2 bytes`
- `DWORD - 4 bytes`
- `QWORD - 8 bytes`

- x86 stores data in 8 General-Purpose Registers (GPR), 6 Segment Registers, 1 Flag Register, and an Instruction Pointer.
- GPR (16bit)
  1. Accumulator register (AX). Used in arithmetic operations
  2. Counter register (CX). Used in shift/rotate instructions and loops.
  3. Data register (DX). Used in arithmetic operations and I/O operations.

4. Base register (BX). Used as a pointer to data (located in segment register DS, when in segmented mode).
5. Stack Pointer register (SP). Pointer to the top of the stack.
6. Stack Base Pointer register (BP). Used to point to the base of the stack.
7. Source Index register (SI). Used as a pointer to a source in stream operations.
8. Destination Index register (DI). Used as a pointer to a destination in stream operations.

- These are listed in the same order that is used in a push-to-stack operation, which will be covered later.
- In 16-bit mode, the register is identified by its two-letter abbreviation from the list above. In 32-bit mode, this two-letter abbreviation is prepended with an 'E' (extended). For example, 'EAX' is the accumulator register as a 32-bit value. The same follows for 64-bit, and the ‘R’ prefix.
  - In 64-bit processors operations that output to a 32-bit subregister are automatically zero-extended to the entire 64-bit register. Operations that output to 8-bit or 16-bit subregisters are *not* zero-extended (this is compatible x86 behavior).

- Segment Register
  - Stack Segment (SS). Pointer to the stack.
  - Code Segment (CS). Pointer to the code.
  - Data Segment (DS). Pointer to the data.
  - Extra Segment (ES). Pointer to extra data ('E' stands for 'Extra').
  - F Segment (FS). Pointer to more extra data ('F' comes after 'E').
  - G Segment (GS). Pointer to still more extra data ('G' comes after 'F')
  - Segment Registers are not typically used with applications running on modern operating systems, instead pointing to the same location and the application utilizing paging instead. (with the exception of FS and GS which point to thread specific data)
- FLAGS Register
  - Also: EFLAGS and RFLAGS (notice a pattern?)

- All FLAGS registers contain the condition codes, flag bits that let the results of one machine-language instruction affect another instruction.
- Arithmetic and logical instructions set some or all of the flags, and conditional jump instructions take variable action based on the value of certain flags.
- For example, jz (Jump if Zero), jc (Jump if Carry), and jo (Jump if Overflow) depend on specific flags. Other conditional jumps test combinations of several flags.

- Instruction Pointer (IP, EIP, RIP)
  - The IP register contains the address of the next instruction to be executed if no branching is done.
  - IP can only be read through the stack after a call instruction.