

Some background material

Programs and Processes

Program: a file (an executable *image* or set of instructions that can be interpreted).

Process: an instance of a program that is running.

A process is *alive*, a program is just a file.

Compiled vs. Interpreted

Some languages required a translator (called a compiler) that converts a text file in to an executable image (machine language).

Interpreted languages are text files that are read by another program (an interpreter) and executed *on the fly*.

Machine Language

- A sequence of bits.
- Organized in to *words*.
 - different architectures use different length words – typical sizes are 16, 32 or 64 bits.
- The bits are an encoding of some operations that should take place in the CPU.

Instructions

- Each *instruction* does something relatively simple.
 - move some bits around
 - treat some bits as base 2 numbers and apply arithmetic operations.
 - send/read some bits to/from I/O devices.
 - select the group of bits that will make up the next *instruction*.

Example Machine Language

Each instruction is 4 bits long.

There are $2^4=16$ possible unique instructions.

We can organize the bits to make it easy to design a machine that can execute the instructions.

first 2 bits represent an operation.

last 2 bits represent what to operate on.

Machine Language for Life

Operations (2 bits = 4 possible operations):

- 00: Eat
- 01: Play
- 10: Study
- 11: Watch Survivor

Eat Instructions

The last 2 bits specify what we eat:

- 00 Pizza
- 01 Burger
- 10 Macaroni and Cheese
- 11 Softshell Crab

We have 4 different eat instructions.

Play Instructions

The last 2 bits specify what we play:

- 00 Surf the Web
- 01 Quake
- 10 Music
- 11 Pin-the-tail-on-the-donkey

We have 4 different play instructions

Study Instructions

The 3rd bit specifies what we study:

- 0 Comp. Org.
- 1 American History

The 4th bit specifies how hard we study:

- 0 Holding book, but eyes closed
- 1 Cram session

We have 4 different study instructions

Survivor Instructions

4 possible codes: 1100, 1101, 1110, 1111

All four mean the same thing (the last 2 bits don't matter): **WATCH SURVIVOR**

It's OK to waste 2 bits, after all
wasting our time with survivor, right?

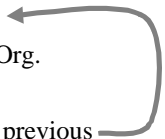
A program

- 0011 Eat soft shell crab.
- 1010 Study history with eyes closed.
- 0100 Surf the web.
- 1101 Watch survivor.
- 1001 Cram for Comp. Org.
- 0101 Play quake

Lose the Survivor Instruction

- Let's make things more interesting:
- *Go back* instruction instead of survivor.
- Last 2 bits determine how far back we go:
 - 00 Repeat previous instruction
 - 01 Go back 1 instruction (before previous)
 - 10 back 2 instructions
 - 11 back 3 instructions

A new plan for tomorrow

- | | |
|------|---------------------------------|
| 0011 | Eat soft shell crab. |
| 1010 | Study history with eyes closed. |
| 0100 | Surf the web. |
| 1001 | Cram for Comp. Org. |
| 0101 | Play quake |
| 1110 | Go back 2 before previous |
- 

Stored-program Computer (VonNeumann Computer)

- Somehow put sequence of instructions in a memory.
- A control unit sequences through the memory (instructions) one at a time.
- It is also possible to put *other stuff* in the memory (data).

Memory: the key to competence

“To be a good programmer you must understand the organization of instructions and data in memory”

- Rudy on Episode 3 of Survivor

Real Instructions and Data

We will spend lots of time on real instruction sets and see where *data* comes in later in the course.

For now – we all need to understand that each piece of data is just a bunch of bits (data looks just like instructions).

Interpreting Data

- It's impossible to tell what a chunk of bits means without any other information.

01001000

- Could be the integer 80
- Could be an instruction.
- Could be the character 'A'
- Could be a floating point number.

Context determines data type

- A computer executes instructions sequentially – it just grabs the bits in the next memory location and assumes it's an instruction.
- Some instructions add integers – in this case the computer assumes the bits in memory represent integer numbers.

Real Programs

- A machine language program is copied in to memory and starts executing at a known location.
- Some of the memory is used to hold data (for example numbers that are added, or text that is read from a keyboard or file).

Operating System

- One of the functions of the operating system (which is itself a running program) is to move programs in to memory and start them running.
- The operating system must manage the memory of the computer (the OS itself must be in memory!).

Types of memory

- RAM: Random Access Memory
- ROM: Read only Memory

- ROM is often used to hold the initial program that a computer should execute when turned on.

Machine Language

In the old days... humans would write programs in machine code (determine every 1 and 0).

The 1s and 0s were put in to memory with a set of switches!

Assembly Language

- An assembler is a program that reads a text file containing symbolic descriptions of instructions (much easier for humans to write).
- The output of an assembler is machine code (the 1s and 0s).

High-level languages

- Many high-level languages translate program text to assembly language (and then run through an assembler).
- C compilers often do this, we will test this out by generating assembly language from C code.