Computer systems

College of Saint Benedict & Saint John’s University

great insights of computer science[1]

Bacon, Leibniz, Boole, Turing, Shannon, & Morse
There are only **two nouns** that a computer has to deal with in order to represent “anything”: 0, 1.

[1]The great insights of computer science / CC BY-SA 3.0

- “anything”: there are some things computers cannot do — like determine if a program will ever finish.

great insights of computer science, cont'd

Turing

There are only **five verbs** that a computer has to perform in order to do "anything":

1. move left one location;
2. move right one location;
3. read symbol at current location;
4. print 0 at current location;
5. print 1 at current location.

great insights of computer science, cont'd

Boehm and Jacopini

There are only **three grammar rules** needed to combine these verbs (into more complex ones) that are needed in order for a computer to do ”anything”:

1. *sequence*: first do this, then do that;
2. *selection*: IF such-and-such is the case, THEN do this, ELSE do that;
3. *repetition*: WHILE such-and-such is the case DO this.

a simple language

1. two nouns
2. five verbs
3. three grammar rules

`<`	move left one location
`>`	move right one location
`0`	print `0` at current location
`1`	print `1` at current location
`[`	if current location is `0`, then go to instruction after matching `]`
`]`	go to matching `[` instruction

```
1>1>0>1>0<<<<[0>]1
```

- *sequence*: start at left-most instruction and progress a single instruction to the right
- *selection* and *repetition:* `[…]` provide both — repetition is just fancy selection

a simple language, cont'd

<	move left one location
>	move right one location
0	print 0 at current location
1	print 1 at current location
[	if current location is 0, then go to instruction after matching]
]	go to matching [instruction
^	add one to the 4-bit number ending at the current location

^ replaces the sequence `>0<<<<[0>]1`

```
1>1>0>1^
```

- let's add another instruction to increment a number by one
- we have introduced some *abstraction*

abstraction

abstraction

A mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time.

Abstraction allows program designers to separate categories and concepts related to computing problems from specific instances of implementation.[2]

[2]Abstraction / CC BY-SA 3.0

types of abstraction

data abstraction

The separation of a data type's logical properties from its concrete implementation.

In fact, a data type is a data abstraction.

```
boolean found := false
```

control abstraction

The separation of the behavior of a set of actions from its concrete implementation.

One of the main purposes of programming languages.

```
a := (2 + 3) / 4
```

- Data abstraction — take 8 bits, how do you know how to interpret the 8 bits? You will need to know if it is an: integer, a floating-point value, or something else...all three are abstractions of the bits
- Here is an example — How is an 32-bit number actually stored in our machine? $0x0000010F = 271$
- In the example code, `:=`, `+`, and `/` are all examples of control abstraction
- What vale does `a` hold?
  - You will need to now what data abstraction we are using for the numbers `2`, `3`, and `4`, so that you know their logical properties, like how addition and division of two of them works
- Other examples include decisions, iterations, functions
- Where do classes in Java fit?

abstraction levels

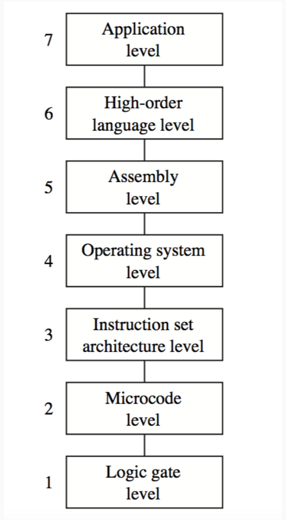
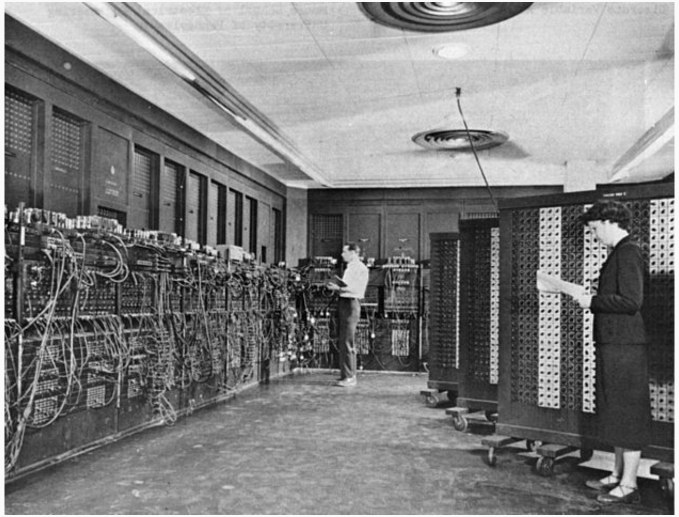


Figure P.1

history of abstraction


U.S. Army Photo / Public Domain

- Circa 1946 — World War II
- LG1 & ISA3
- Six women: Kay McNulty, Marlyn Wescoff, Ruth Lichterman, Betty Jean Jennings, and Fran Bilas, programmed the first large-scale general purpose machine, ENIAC, to compute ballistics trajectories. Programming was done by reorganizing wiring of plugboards.

history of abstraction, cont'd

SSEM Manchester museum close up / CC BY 3.0

- Circa 1948
- LG1 & ISA3
- Stored program in memory. Programs were entered in binary form by stepping through each word of memory in turn, and using a set of 32 switches known as the input device to set the value of each bit of each word to either 0 or 1.

history of abstraction, cont’d

Commodore Grace M. Hopper, USN / Public Domain

- Circa 1951
- ASM5 & HOL6
- assembly and high-order languages were developed around the same time
- Assembly programs are machine-specific, 1-to-1 mapping between assembly instructions and machine instructions
- some early programming languages still in use today:
  - FORTRAN (1954)
  - LISP (1958)
  - COBOL (1959)
  - ALGOL (1958)

history of abstraction, cont'd

- Circa 1956
- OS4
- operating system is a piece of software that runs other pieces of software
- allow multi-tasking and multi-user environments
- gives software an interface to hardware — it manages hardware resources

hardware abstraction

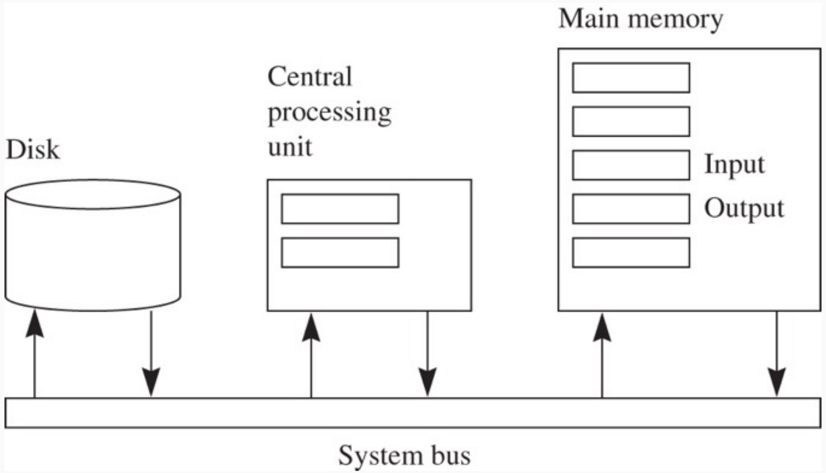
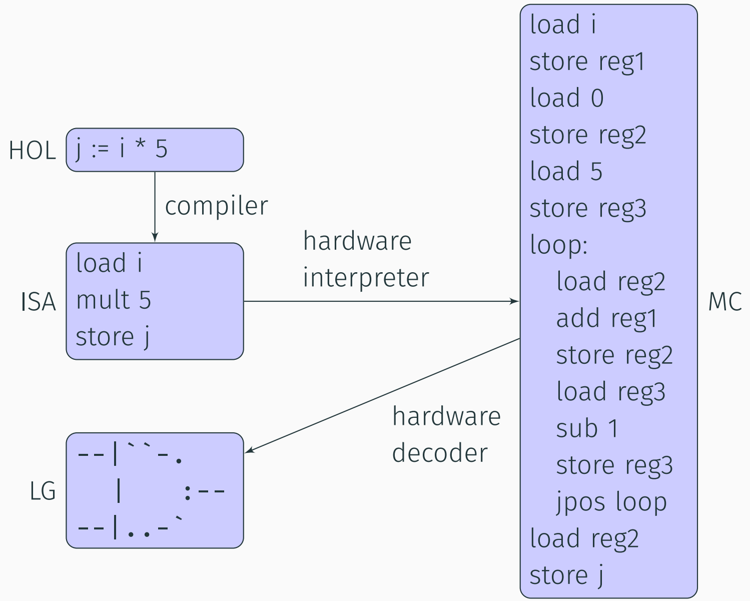


Figure 1.9

- Ask students what the different parts of the picture are
- The only instructions that a CPU can perform are hard-wired into its circuits (LG1) — example: `add` is hard-wired, `multiply` is written in MC2
- MC2 requires an interpreter to be hard-wired into CPU to translate MC2 to LG1
- CPU can only compute on values in *registers* — modern machine has on the order of tens of registers
- Show demonstration with SimHYMN
- *memory-mapped i/o* — the input and output devices are treated as special locations in a process' address space — this is handled by the OS

hardware abstraction, cont'd



- With the introduction of MC, there is not a 1-to-1 mapping between IS3 and capabilities of CPU
- Now, ISA defines capabilities and MC implements them if LG doesn't

software abstraction

algorithm
: a set of *instructions* that, when carried out in the proper sequence, solves a problem in a finite amount of time

program
: an algorithm written in a language that is understandable by a computer, i.e., that can be executed on a computer

- what is an instruction
  - this is where people get hung up, *instruction* is not necessarily an instruction that a machine can understand
  - but, if the algorithm is for a computer, the instruction should be something that the computer is capable of

algorithms

```
while i is greater than or equal to 0
  print i
  subtract 1 from i
```

```
print the numbers from i to 0
```

- I consider both of these algorithms

system performance

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

- Note the prefix indicate powers of 10

- $\frac{\text{instructions}}{\text{program}}$ is # of ISA 3 instructions in the program
- $\frac{\text{cycles}}{\text{instruction}}$ is average # of MC2 instructions per ISA3 instruction
- $\frac{\text{time}}{\text{cycle}}$ is $\frac{1}{\text{CPU frequency}}$ – also called the *period*

system performance

$$\frac{\text{time}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{time}}{\text{cycle}}$$

try it yourself

Suppose your CPU is rated at 2.5 GHz and you execute a program task on your app that requires the execution of 16 million ISA3 instructions. If each ISA3 instruction executes an average of 3.7 MC2 instructions, what is the execution time of the program task in seconds?

- Note the prefix indicate powers of 10
- $\frac{\text{instructions}}{\text{program}}$ is # of ISA 3 instructions in the program
- $\frac{\text{cycles}}{\text{instruction}}$ is average # of MC2 instructions per ISA3 instruction
- $\frac{\text{time}}{\text{cycle}}$ is $\frac{1}{\text{CPU frequency}}$ – also called the *period*
- $(16 \times 10^6) \times (3.7) \times (1/(2.5 \times 10^9)) = 23.7\text{ms} = 23.7 \times 10^{-3} \approx 0.024\text{s}$

bandwidth

$$\text{information} = \frac{\text{information}}{\text{time}} \times \text{time}$$

bandwidth

$$\text{information} = \frac{\text{information}}{\text{time}} \times \text{time}$$

try it yourself

One of the largest (but not the largest, but very close to it), telecommunications companies in the U.S., offers an internet product called Continuum that promises up to 100 Mb/s for the low price of $45/month. A smaller and slightly less popular telecomm company offers a competing product offering up to 25 Mb/s for $30/month. If I have two devices capable of streaming Netflix HD, which product should I choose? Construct an argument to explain to the customer service representative for the company whose product you do not choose, why.

- Netflix HD transfers about 3 GB/hour
- $3\text{GB/hour} = 3 * 1000 * 1000 * 1000 * 8 = 24,000,000,000\text{b/hour} = 24,000\text{Mb/hour} \approx 6.6\text{Mb/s}$