Q know system → more reliable code ?

↳ prevent arithmetic errors

↳ cleverly exploit modern processors = faster program !

↳ prevent other vulnerabilites & errors

e.g. buffer overflow, linking

```
#include <stdio.h>

int main()
{
  printf("Hello World\n");
  return 0;
}
```

Ascii-code ?

35 (#), space (32), 60 (<), 62 (>), \n = 10, 34 ("), 34 ("), 92, 110 (\n), 59 (;)

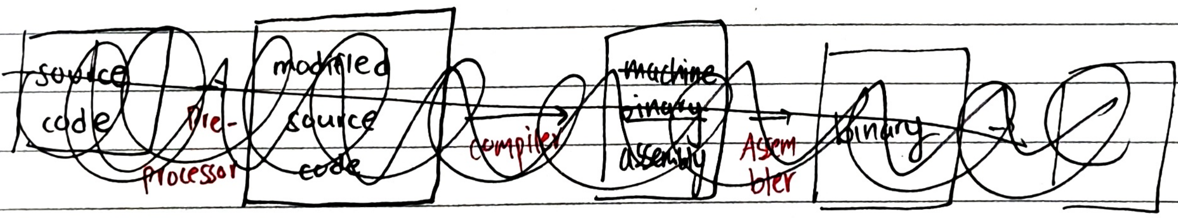
Information = bits + context — everything in computer is bunch of bytes (sequence of 8 0's or 1s.)

- same bit but different value
  - ↳ sometimes integer, character, instruction, etc
- * bits are finite, => sometimes behave in unexpected ways '!

Readable source file → Executable object file

by compiler driver

e.g. gcc -o name file.c



source code → pre-processor → modified source code → compiler → assembly language → Assembler

→ binary (machine) → linker → hello (object file)

Binaries to be linked (printf.o)

Understanding how compilers work
→ more efficient code (if vs switch, for vs while, indexing, pointing) ch 5
→ x86-64 & hierarchical nature of the memory system ch 3 ch 6
→ buffer-overflow vulnerabilities ch 3
→ link-time errors & global variables across diffent source files ch 7

Hardware

- **buses**
  - ↳ transfer data by fixed sizes (4 or 8 bytes)
- **I/O devices** <=> I/O buses
  - ↳ external connection for our computers.
- **Main Memory**
  - ↳ temporary storage device
  - ↳ collection of DRAM (dynamic random access memory)
- **Processor**
  - ↳ interprets instructions in the memory
  - ↳ ALU with word-sized registers

execution of hello

1. ~~memory~~ disk → memory (in main function)
2. memory (instructions) → processor
3. ↳ copy "hello world\n" from memory → registers
4. registers → display device

Caches matter

fills the processor-memory gap { size: processor ××× memory <<< disk
speed: processor >>> memory >>>>>>> disk

- levels of caches.
  - ↳ L1: smaller, faster, data more likely to be retrieved / recently used
  - L2: ~~farther away~~ bigger (larger), bit slower, less likely data
  - L3 (some machines)

* Exploit **locality**: larger data in faster time!
  - ↳ programs more likely to access ~~data~~ local (closer in memory blocks) datas.

(LD)
processor → L1, L2, L3 → memory
SRAM DRAM

→ local disk → servers