Computers

1) Classes

1.1) Personal
- desktop, laptop, tablet, (smart phones)
- ~1% of all computers sold (10 bn in 2008)
- cost: $20 - 2,00

1.2) Embedded
- integrated into a larger device or system.
  - automotive (airbags, ABS)
  - appliances (stove, microwave)
  - airplanes
- ~99% of all CPUs
- cost: microchip PIC12: $0.41

1.3) Servers
- provides service to many users.
  - cloud computing (Amazon EC2, MS Azure)
  - mainframes (IBM system z) - high reliability/uptime
  - supercomputers - weather modelling
    - protein folding
    - simulation
- < 1% of all CPUs sold
- cost ~$2000 / chip.

2) Structure
- Definition: a computer is a "programmable device that can store, retrieve, and process data.

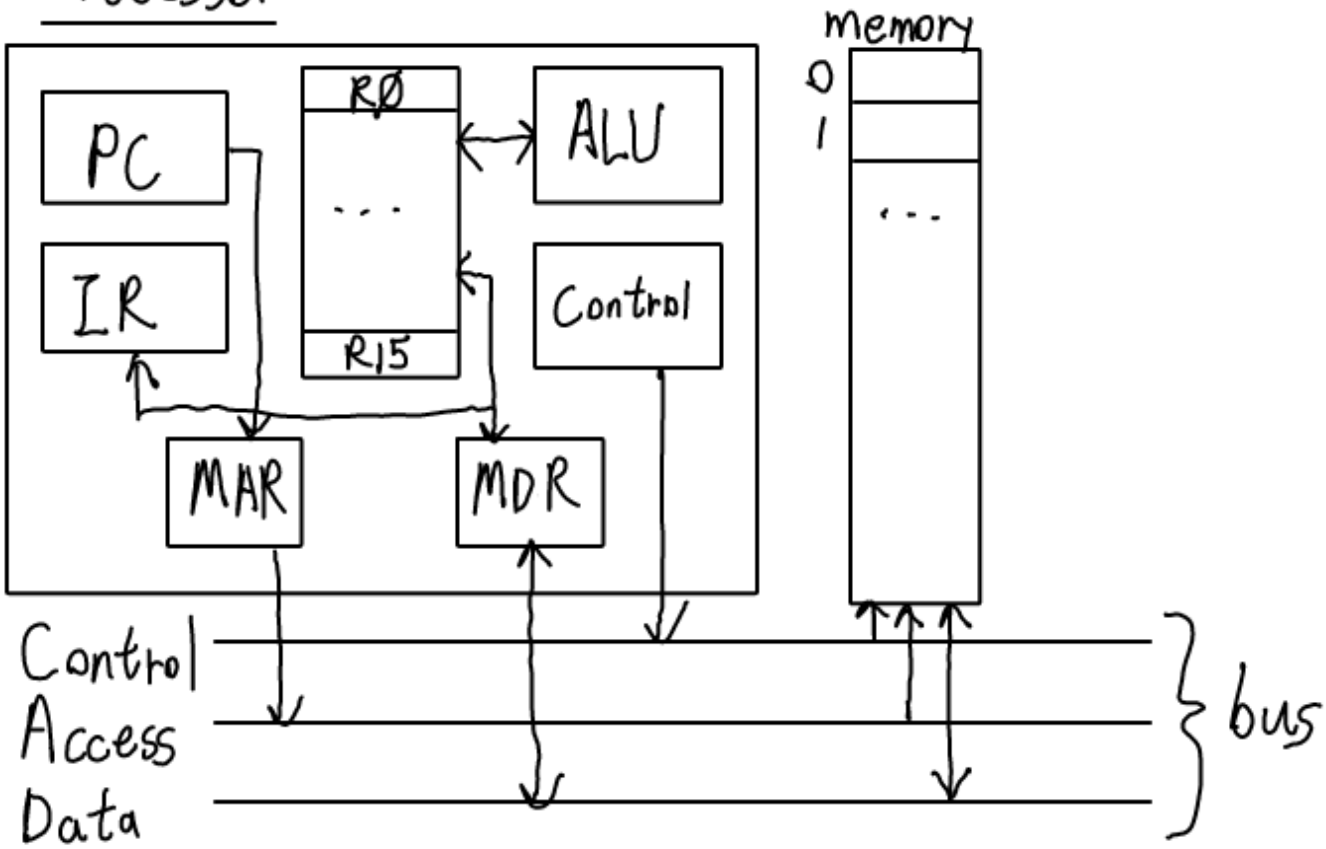
Computers of all classes can be decomposed into five types of functional units.

① Input: mouse, punch card, touchscreen, voice, camera } I/O
② Output: printer, spark plugs, screens
③ Storage: data and instructions (binary)
- memory is organized into a linear array of bytes.

④ ALU - Arithmetic Logic Unit
- performs operations on data stored in registers.
- add, multiply, AND, NOT, ...

⑤ Control Unit:
- intepret instructions, fetch operands, controls ALU

Processor



PC = Program Counter
stores memory address of next instruction

IR = Instruction Register
Stores the instruction read from memory

MAR = Memory Access Register
outputs address to memory

MDR = Memory Data Register
holds data/instructions from memory or going to memory

3) Instruction Execution

1. instruction fetch (IF)
   - copy PC contents to MAR
     and assert R/$\overline{W}$ control signal
   - wait for response from memory
     and copy MDR contents to IR.
   - increment PC
2. instruction decode
   - interpret bits in IR
3. operand fetch (OF)
   - read data from registers
     and/or extract constants from IR.
4. execute (EX)
   - use ALU to read memory (load)
     or write memory (store)
5. writeback (WB)
   - write result to a register

e.g. execute Load R2, LOC ← memory address label

1. always same as above
2. recognize "Load"
3. extract LOC from IR
4. copy LOC to MAR and assert R/$\overline{W}$ control signal
5. copy MDR contents to R2.

HMK: Add R4,R2,R3 (R4 ← [R2]+[R3])
Store R4, LOC

4) Design Paradigm (2., 3.3)

CISC: Complex Instruction-Set Computer

- machine instructions can perform complex operations
  e.g. (x86) movsb copies an array of bytes
- instructions are variable length.
- operands come from registers or memory
  e.g. (M68K) ADD DØ, LOC (mem[LOC] ← [DØ]+[mem[LOC]])
- complex addressing modes
  e.g. (M68K) ADD DØ, (AØ)+
- Smaller object code
- direct support of HLL constructs.
  - ease of assembly language programming
- hardware is difficult to pipeline (speed up)

RISC: Reduced Instruction-Set Computer

- fewer, simpler instructions
- load/store architecture
  - only load or store instructions access memory
  - ALU operands only come from registers

e.g. (ARM)
```
ldr r1, LOC
add r1, rØ, r1
ldr r2, =LOC
str r1, [r2]
```

- object code is larger (by ~30%)
- hardwire easier to pipeline.
- fixed length instructions

5) Register Transfer Notation (2.3.1)
(no standard)

- expresses the semantics of instruction execution as data transfers and control flow (logic)
- memory locations are assigned labels
  e.g. LOC, A
- registers are named R0, R1, PC, IR
- [x] denotes contents of x.
  e.g. [LOC]: contents of memory at LOC
  [R0]: contents of register R0
  [[R0]]: contents of memory at the location specified by contents of R0

',' denotes parallel

';' denotes sequential

e.g. ADD R4, R2, R3

$R4 \leftarrow [R2] + [R3]$

instruction fetch

$MAR \leftarrow [PC]$, $R/\overline{W} \leftarrow 1$, $PC \leftarrow [PC] + 4$;

$IR \leftarrow [MDR]$