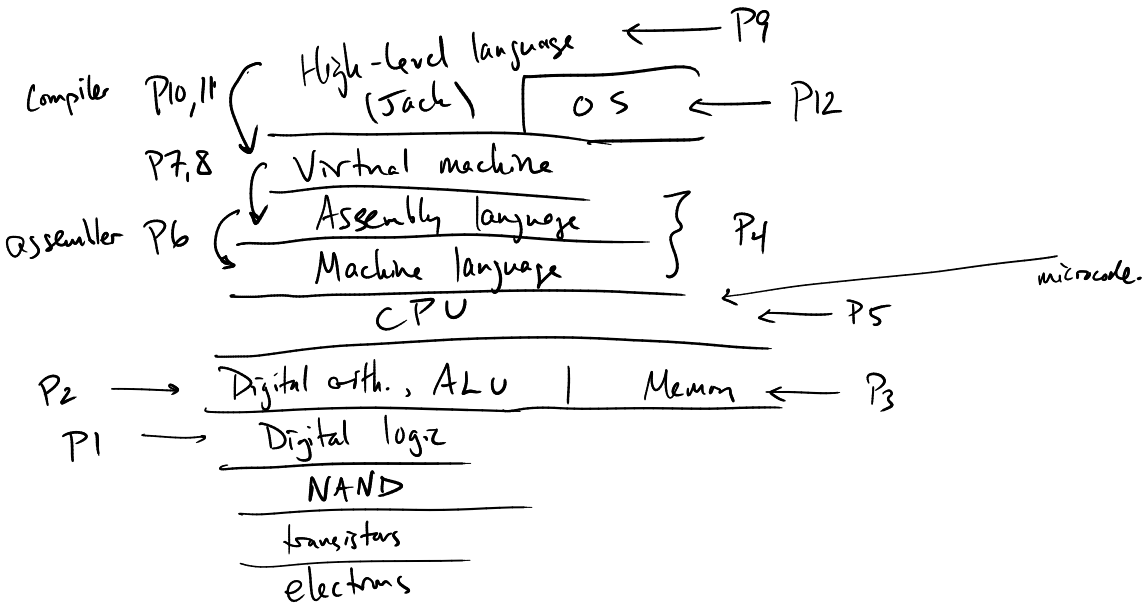


CPU → instruction set = primitive operations the CPU can perform.

Machine language = encoding of instructions as binary strings.

- Pro: binary string = input to a circuit
- Con: hard for humans to read.

Assembly language = encoding of instructions as symbols.

- easier for humans
- has to be translated by an assembler.

1-1 Correspondence Assembly ↔ Machine lang.

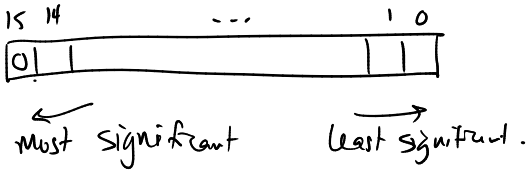
What can a CPU do?

- Load values RAM → CPU register.
- Store values register → other registers or RAM.
- Math + logic (ALU)
- Jump. (esp. conditional jumps).
  ↳ condition based on zr, ng from ALU.
  eg. to check $x > y$, compute $y - x$ and check if result is negative.

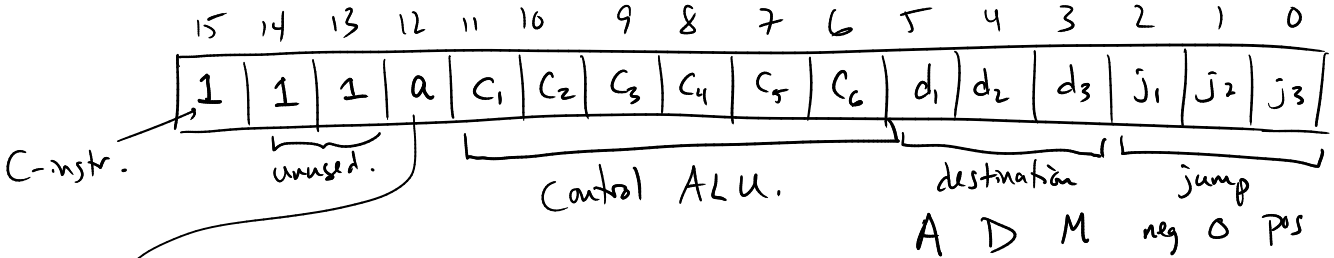
Machine language for the Hack computer.

- 3 registers: A, D, PC
  - A: storing addresses or data.
  - D: storing data
- Instructions are 16 bits. 2 types: A-instructions, C-instructions.

A-instruction: bit 15 is 0.
rest of the 15 bits get loaded into A register.



C-instruction ("computation"): specifies what the CPU should do in 1 cycle.
- D register and {A register value from memory} → ALU.
- ALU does some operation.
- (optionally) store the result of ALU (D, A, memory).
- (optionally) jump.



a bit
0: 2nd ALU input = A reg.
1: 2nd ALU input = RAM[A]
ie. look up value in RAM @ address given by A register.

Assembly language for Hack computer.

A-instr: @value. eg. @24.
(OR @ variable
@ label)

C-instr: dest = op ; jump (optional) → JGT, JNE, JMP, etc.

dest: optional,
A=
or DM=
...

op: some op the ALU can do, using A, D, or M.
eg. D+1
M-D
D&A
~~D+4~~