Lec-4-vonNeumann

What we are looking for

-- A general design/organization

-- Some concept of generality and completeness

-- A completely abstract view of machines
  a definition
  a completely (?) general framework

-- An introduction to a common, standard ISA

-- An introduction to the LC3

Getting in tune with the current scene, listen to

Dave Patterson:

Computer Architecture is Back:
Parallel Computing Landscape

http://www.youtube.com/watch?v=On-k-E5HpcQ

Von Neumann

description (Turing)

$\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_n\}$

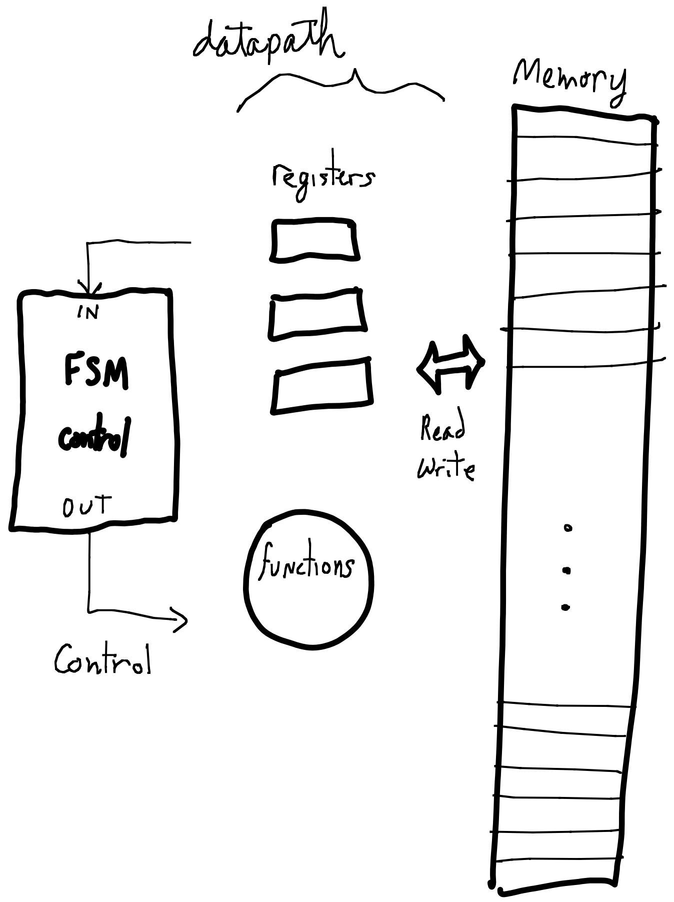
State	Input	Output	Move	Next State
A	σ_1			
	σ_2			
	σ_3			
⋮				
	σ_n			
B	σ_1			
	σ_2			
⋮				

$$\begin{tikzcd} A \arrow[r, bend left, "IN = \sigma_3 / out = \sigma_1 / move = L"] & B \end{tikzcd}$$

32-bit symbols

2^{32} symbols in $\Sigma \rightarrow$ (4G)

VS.

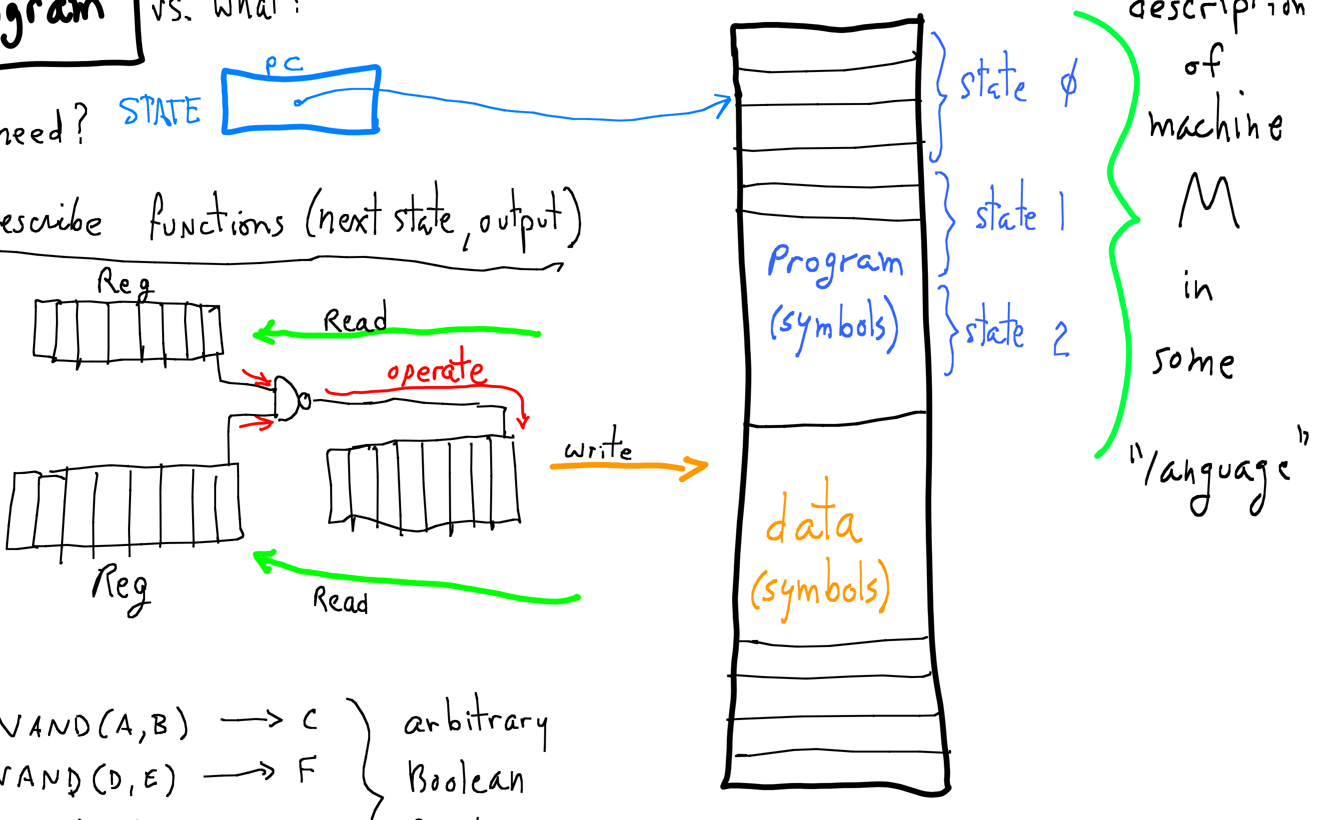


Stored Program

vs. what?

what do we need? STATE

And? describe functions (next state, output)



NAND(A,B) → C
NAND(D,E) → F
NAND(C,F)
etc.

} arbitrary Boolean functions

We need a language rich enough to describe any function, then we can describe any machine, and simulate it.

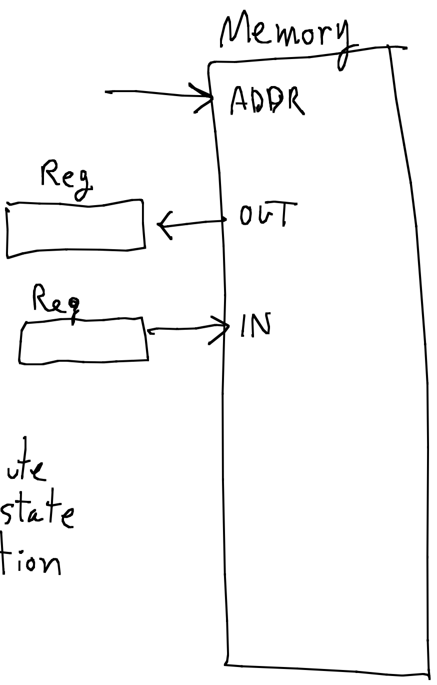
What else ?

NB--It's not obvious what capabilities we need. Can we find a model that could tell us that?

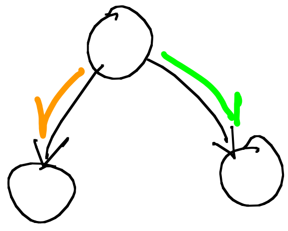
Memory can be thought of as an array, the address is the array index:
 Memory[address] <== in
 out <== Memory[address]

(Convenience / peformance): other functions, AND, OR, NOT
ADD, SUB,
MUL, DIV

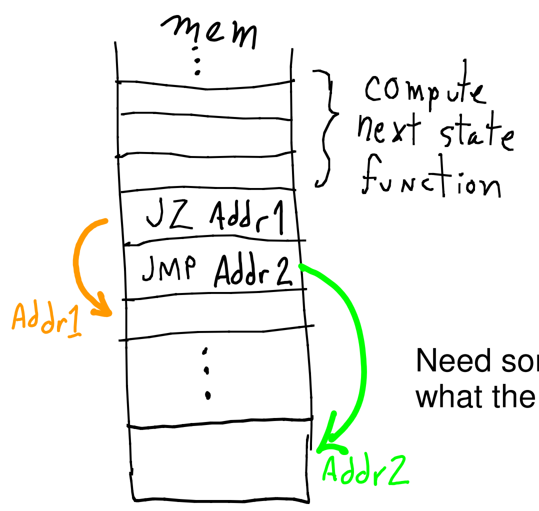
(Read / Write / STACK): Load Reg, Address
Store Reg, Address
push Reg
pop Reg



(Conditional Branching)



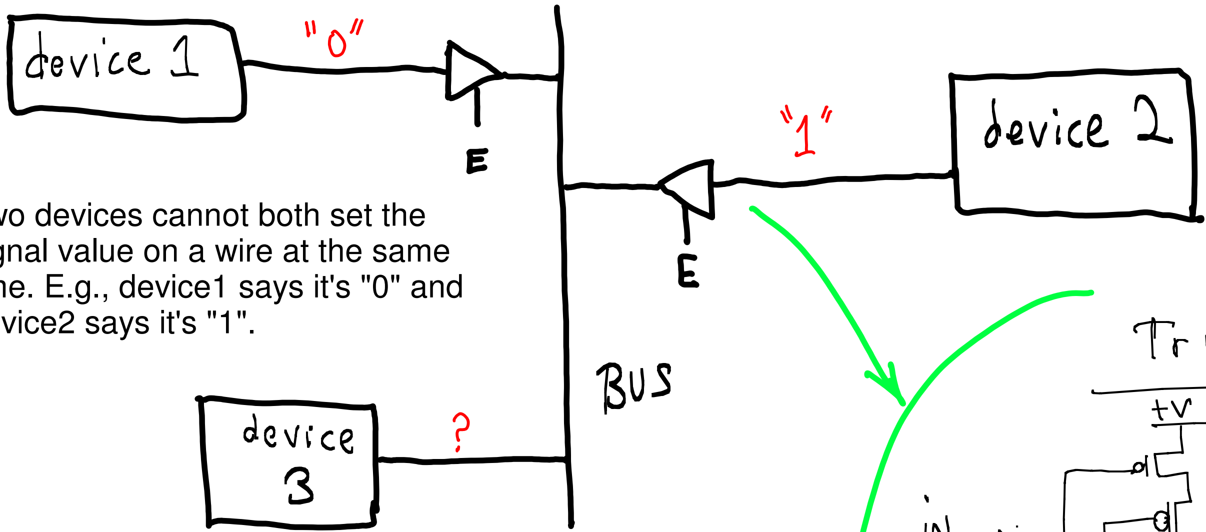
recreate a branched graph in linear memory. Execution follow path through graph.



Need some way for the machine to "know" what the outcome was.

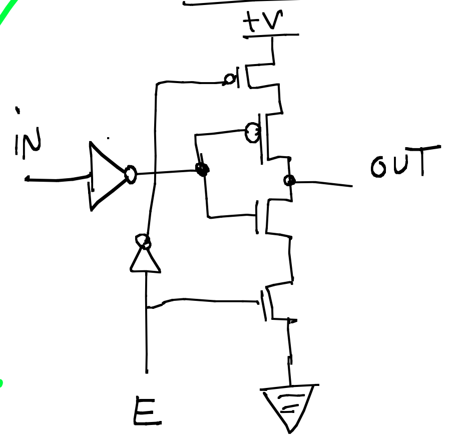
Sharing wires

Controller "enables" one of the tri-states at a time. Bus is shared between devices.



Two devices cannot both set the signal value on a wire at the same time. E.g., device1 says it's "0" and device2 says it's "1".

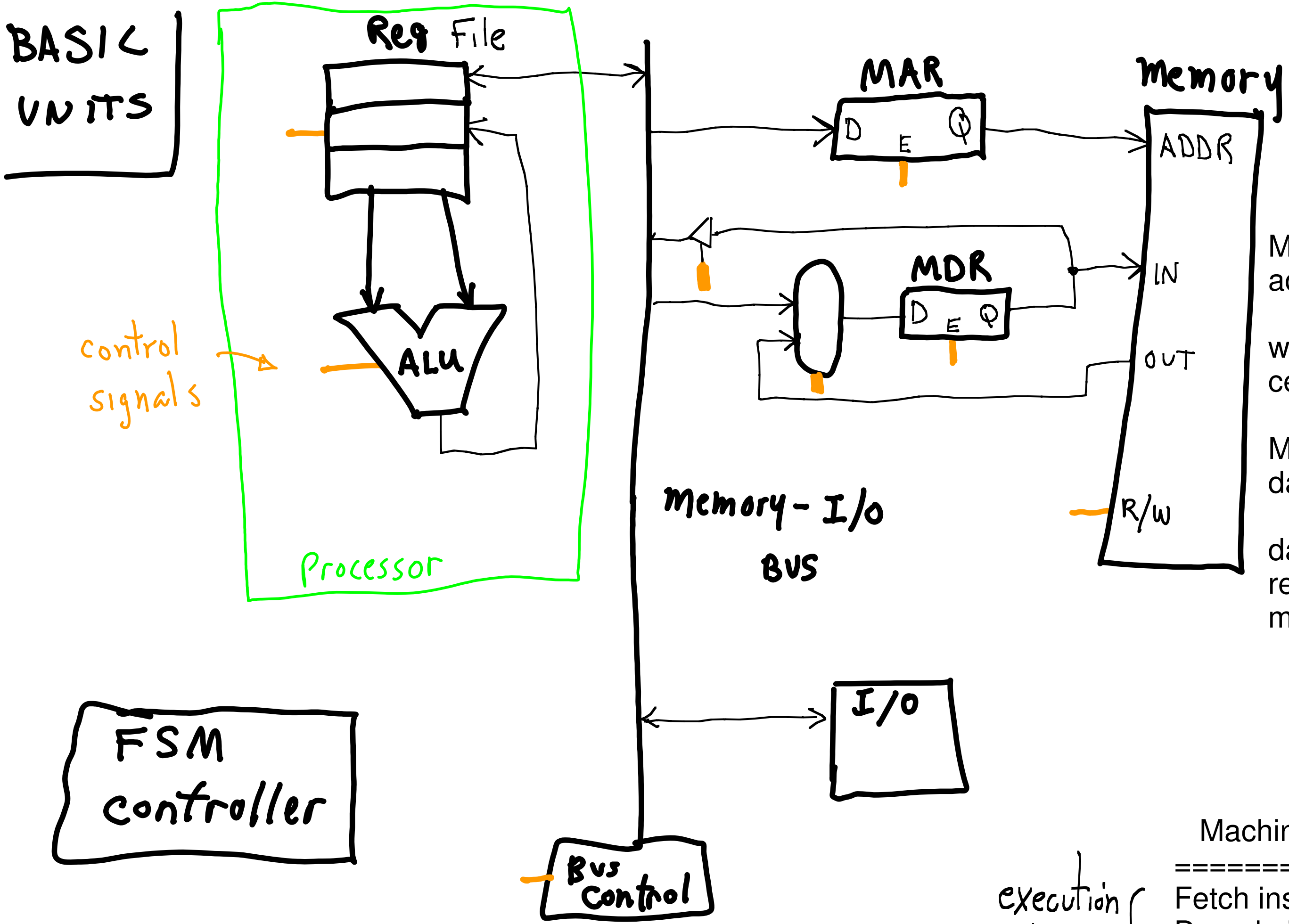
Tristate Buffer



Verilog wire/reg states

x	z	1	0
unknown	high-Z	3v	0v

BASIC UNITS



MAR, memory address register:

which memory cell is accessed.

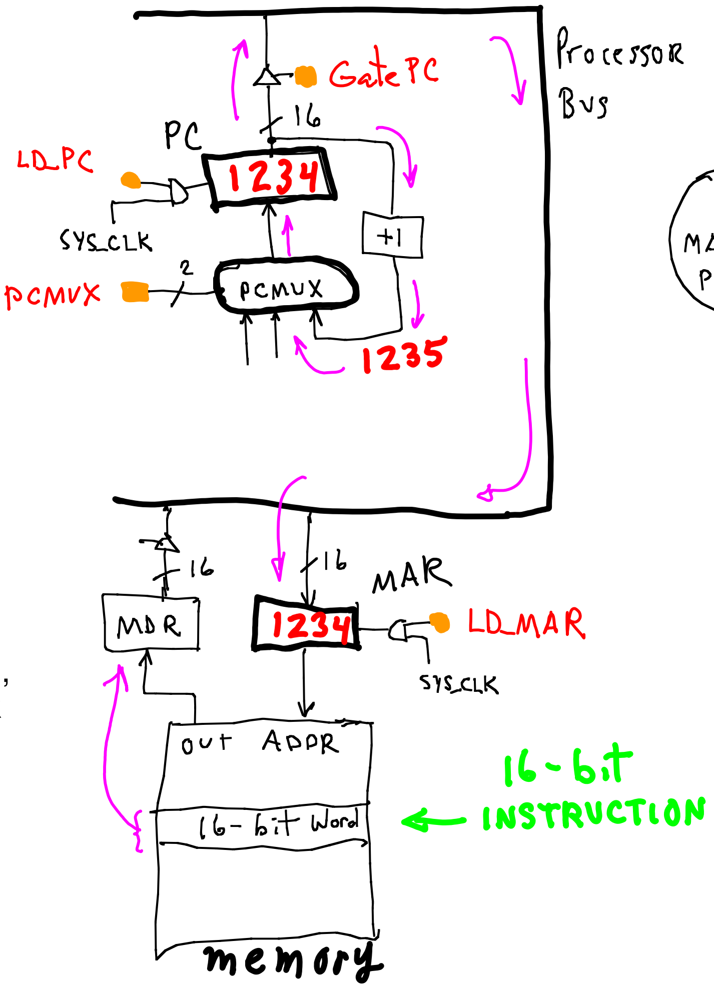
MDR, memory data register:

data sent to, or received from memory.

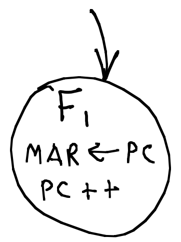
Instruction Execution

LC-3

Fetch-1



execution phases	Machine Cycle
	Fetch instruction
	Decode instruction
	Evaluate address
	Fetch operands
	Execute instruction
	Store result



FSM Controller State

Fetch, F1

● GatePC	= 1	control signals
● LD_PC	= 1	
● PCMUX	= 10	
● LD_MAR	= 1	

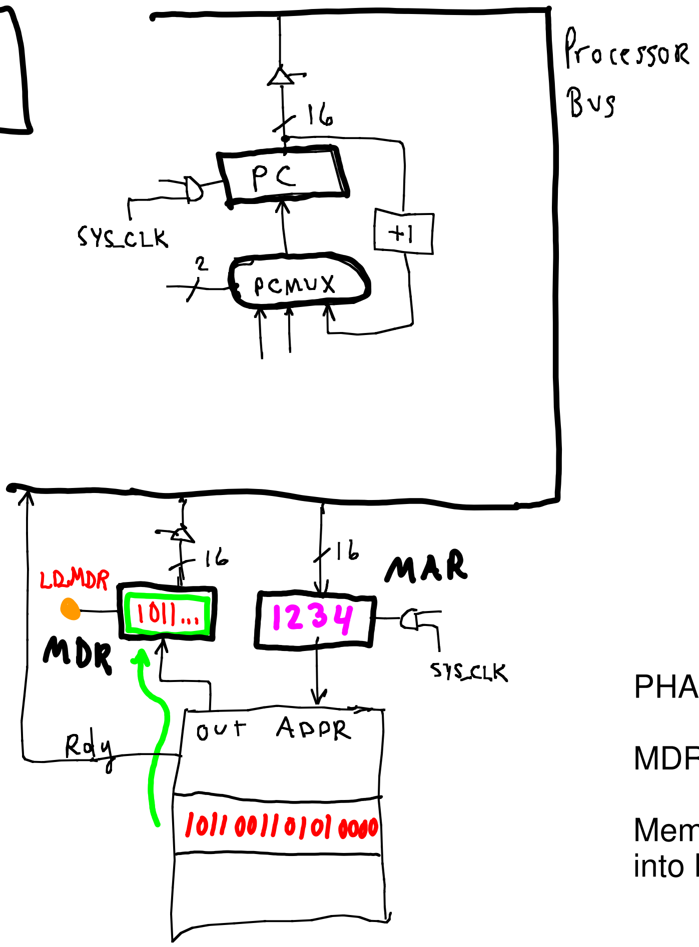
PHASE:

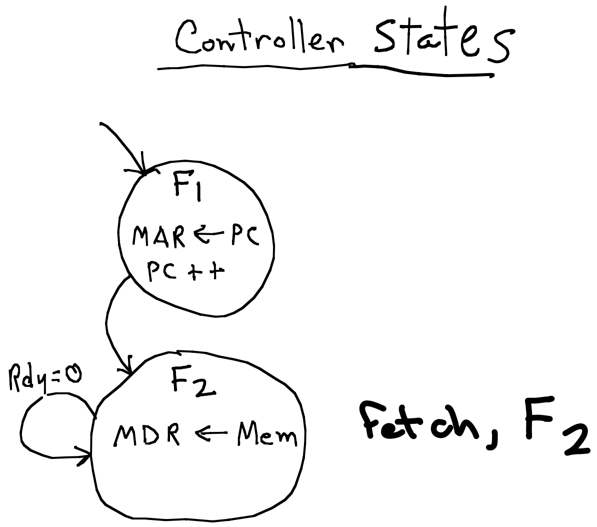
Fetch instruction
-- 1st step, state F-1

PC <== PC+1
MAR <== PC

(Happens in parallel, finalized when clock pulse arrives.)

LC3 fetch, F2





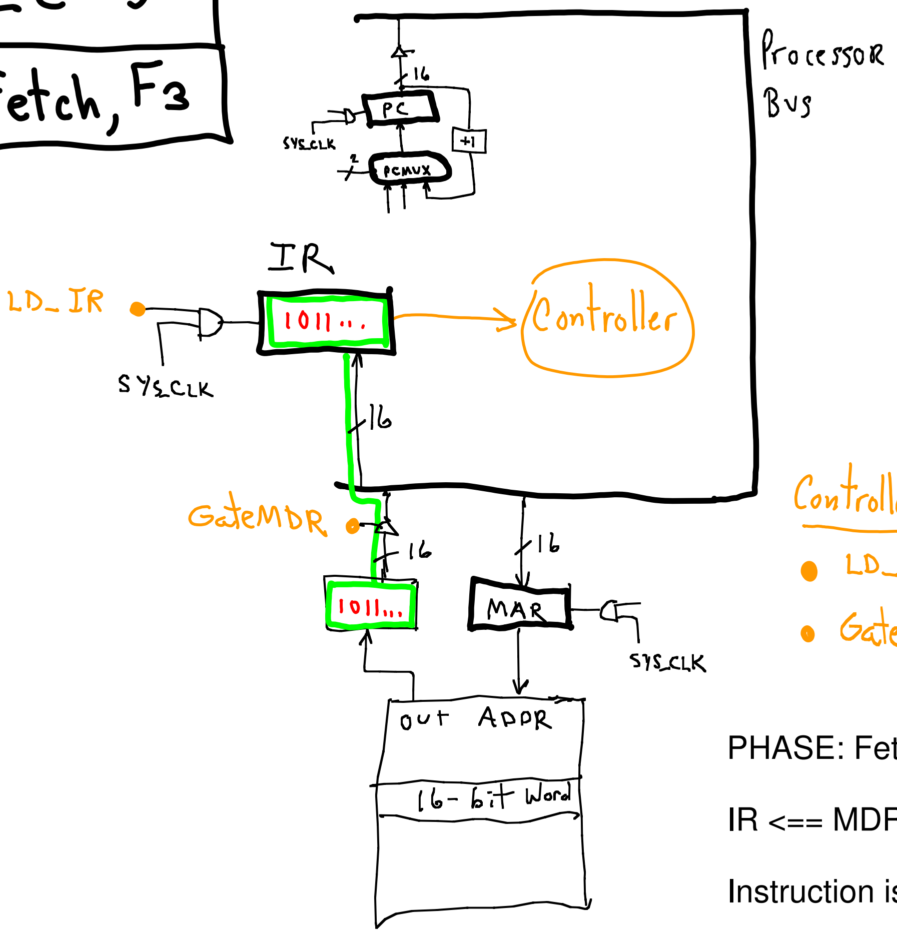
● LD_MDR

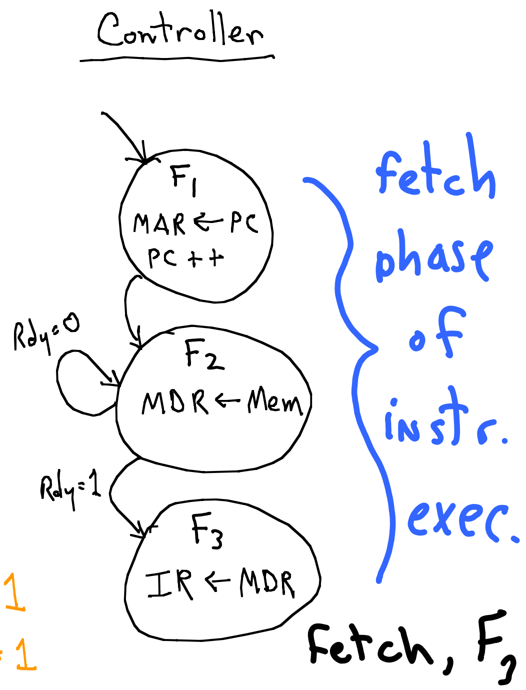
PHASE: Fetch instruction, --2nd step, F2

MDR <== Mem.out

Memory word selected by MAR is copied into MDR, when memory is ready.

LC-3 Fetch, F3





Controller

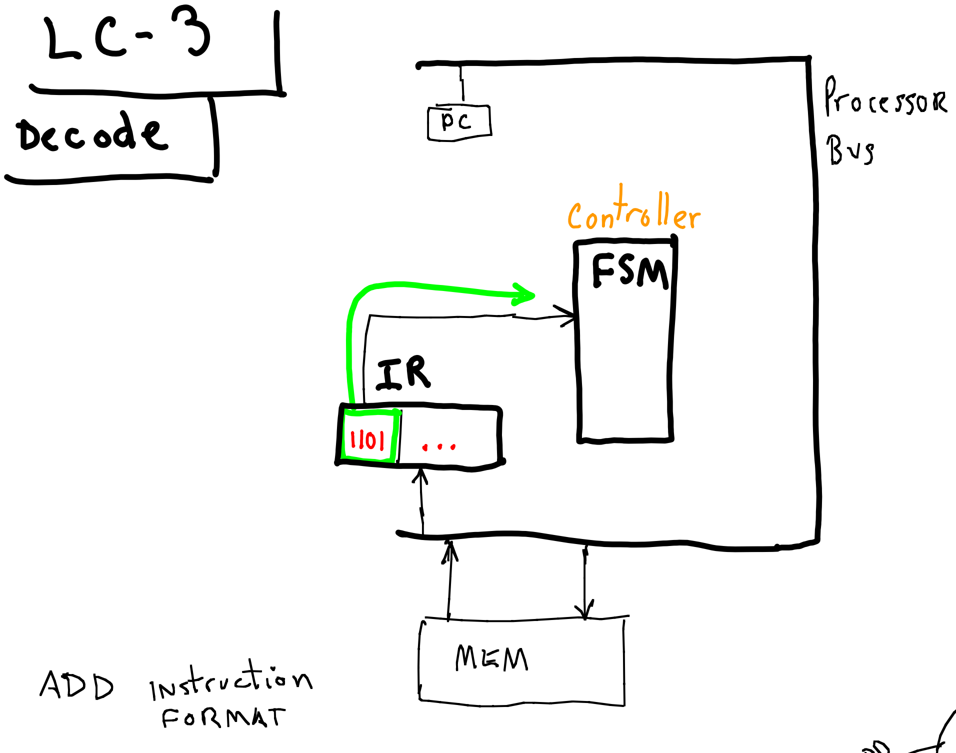
● LD_IR = 1

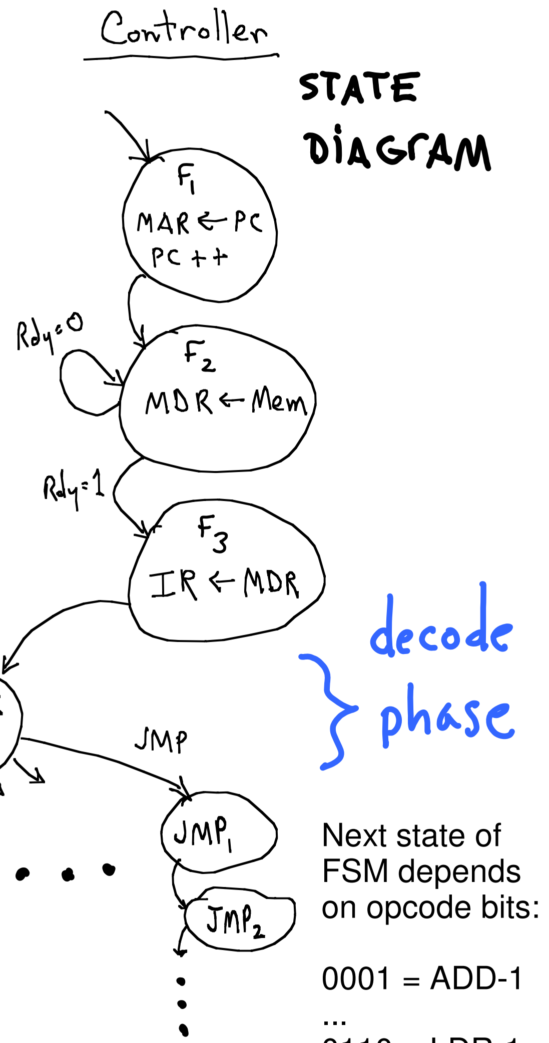
● GateMDR = 1

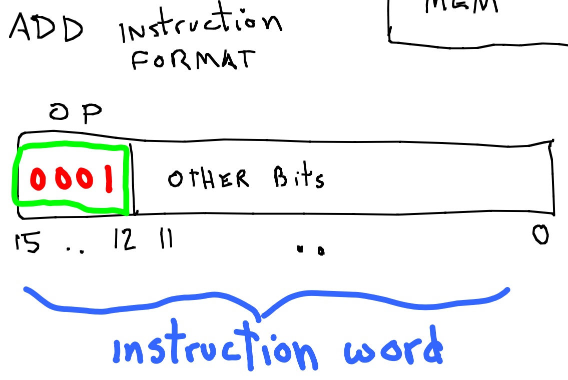
PHASE: Fetch instruction, 3rd step, F3

IR <== MDR

Instruction is remembered, ie., "registered".

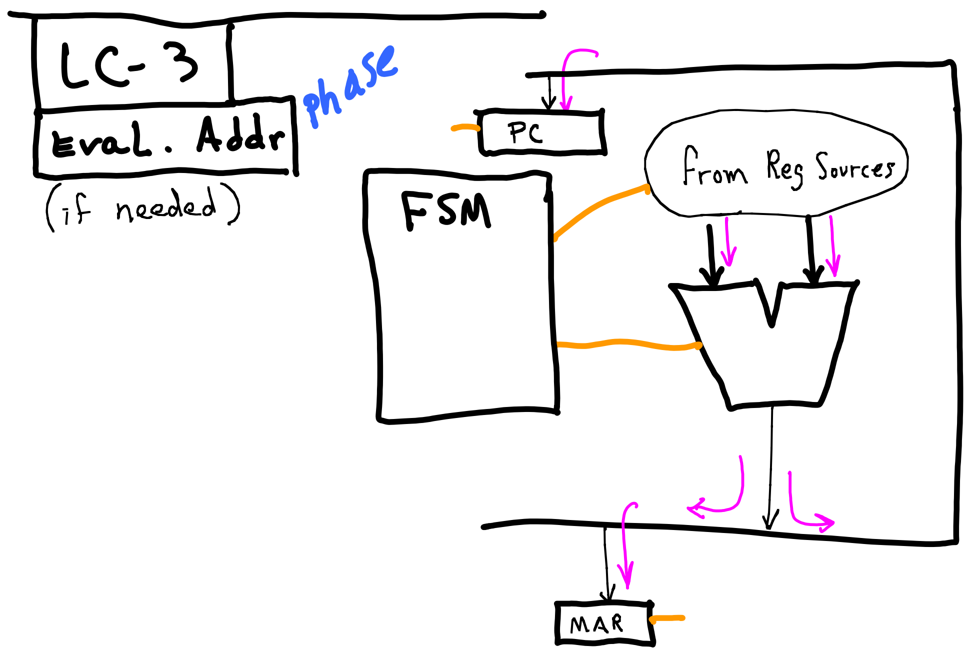






Next state of FSM depends on opcode bits:

0001 = ADD-1
...
0110 = LDR-1
...
1100 = JMP-1



Evaluate Address

Sources for address calculation are any machine registers:

IR
any of 8 registers in RegFile
PC
Interrupt Vector Register

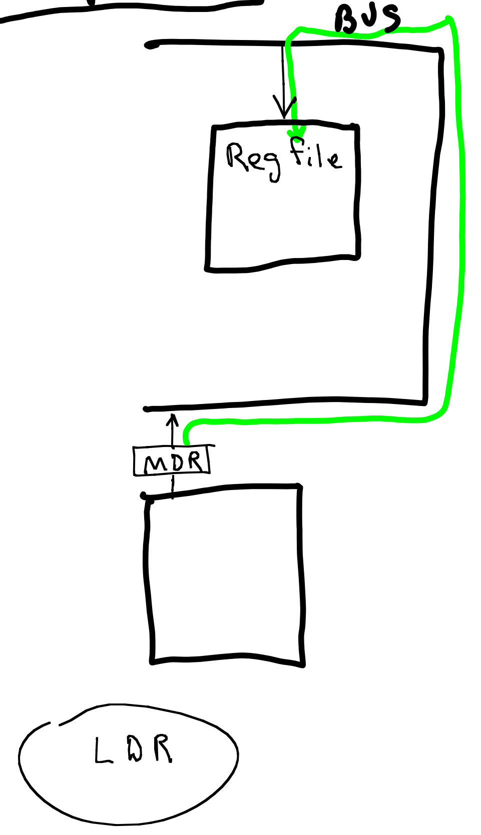
calculated address is used to access memory to,

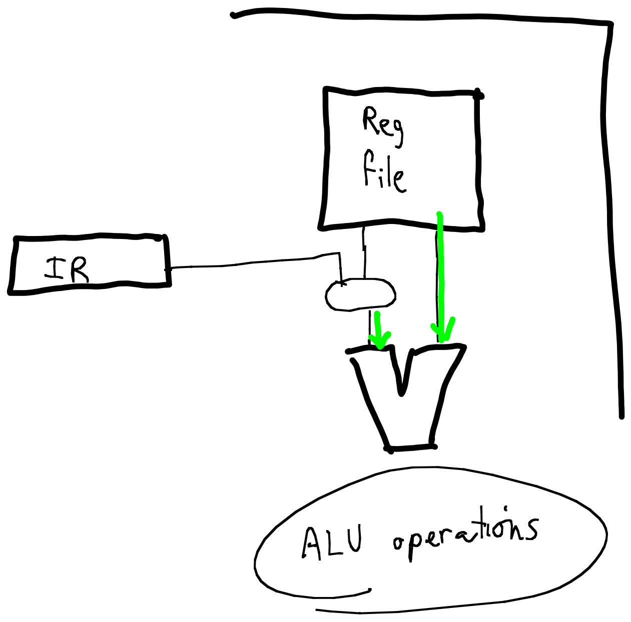
(1) get next instruction: JMP, BR, INT, TRAP, Exception

or

(2) transfer data between memory and register: LDR, STR

LC-3 Fetch operands phase



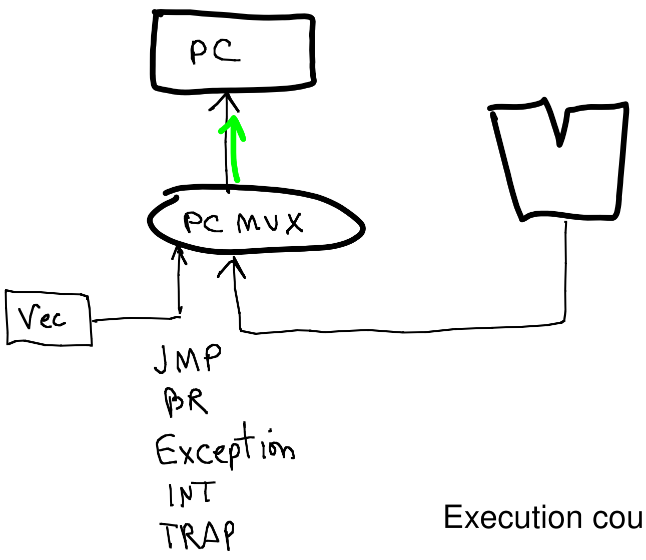


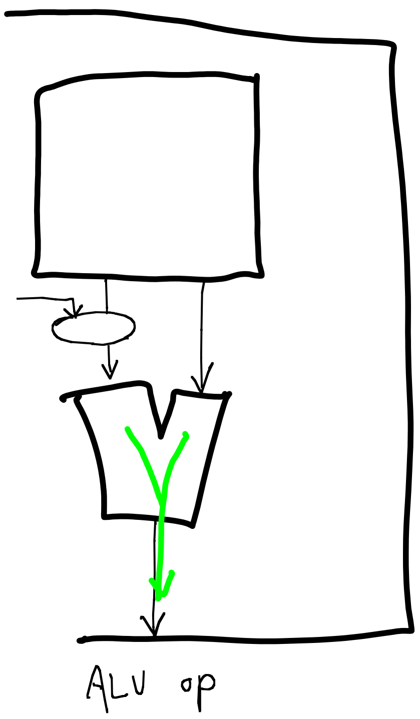
Fetching could mean,

1. move from memory to register
or
2. make available to ALU

LC3's "load from memory" instructions do nothing else but copy from the MDR to a register; ie., no further phases.

LC-3 Execute phase





Execution could mean,

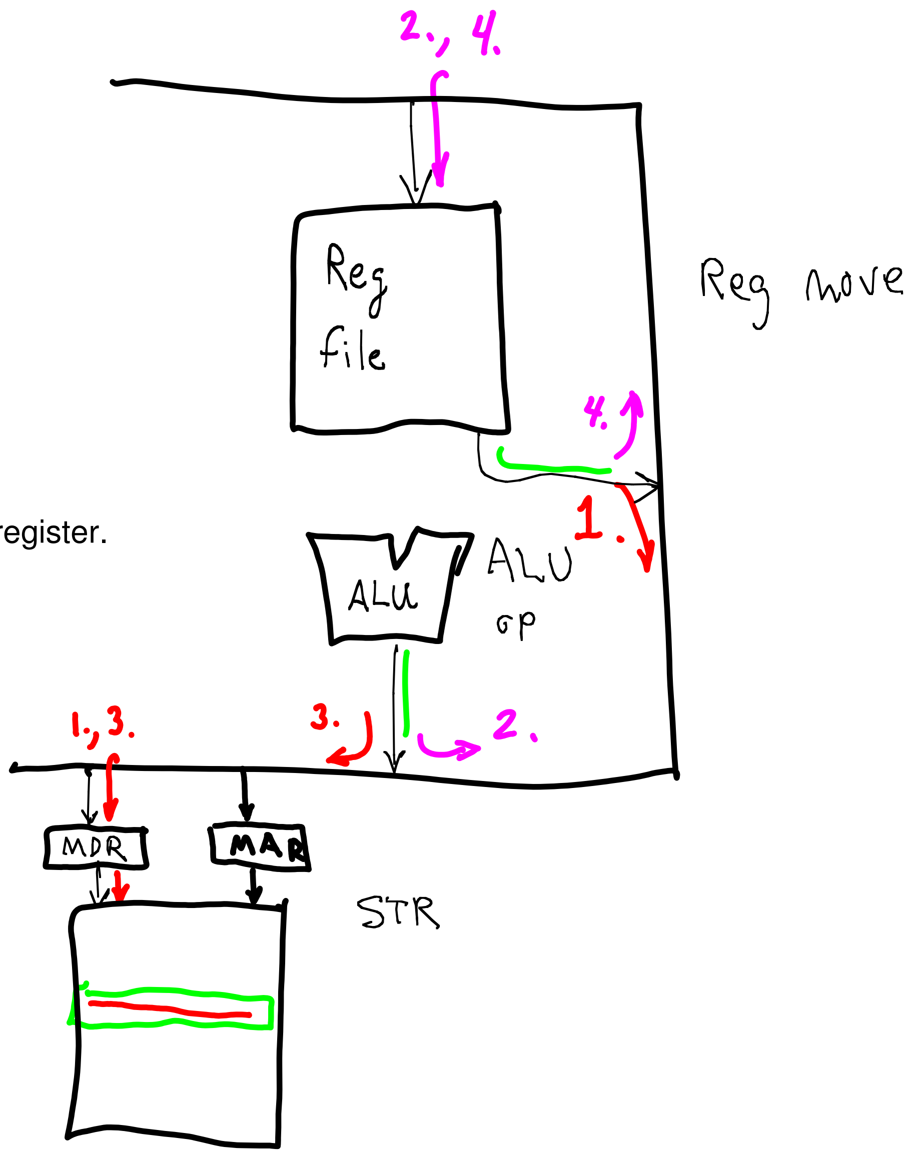
1. Performing ALU operation and making result available
2. Load PC with address calculated in Address Calc. Phase.

LC-3 STORE phase

Store could mean,

1. copy from a register into memory
2. move ALU result to register
3. move ALU result to memory
4. copy from one register to another

For LC3, ALU result always goes to register.

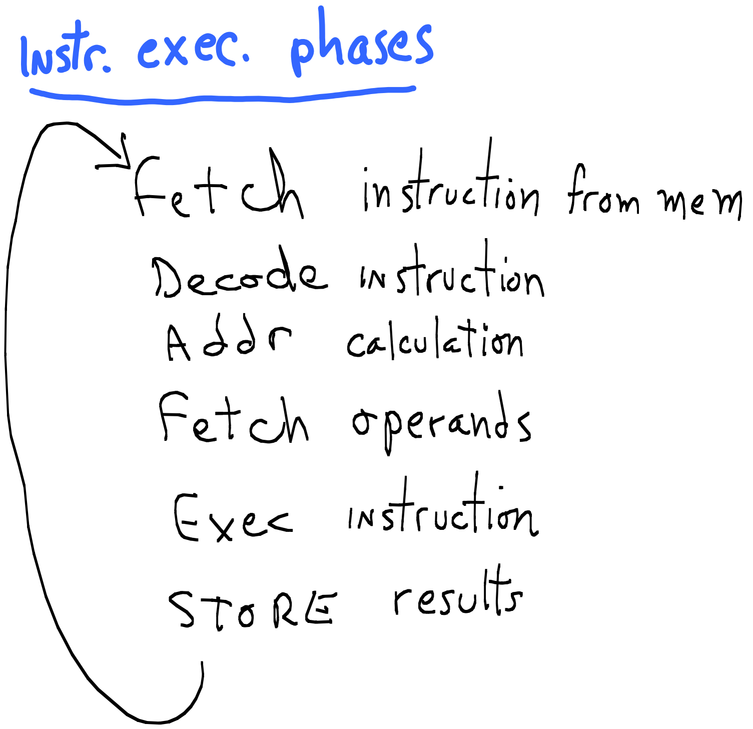


Instruction Cycle

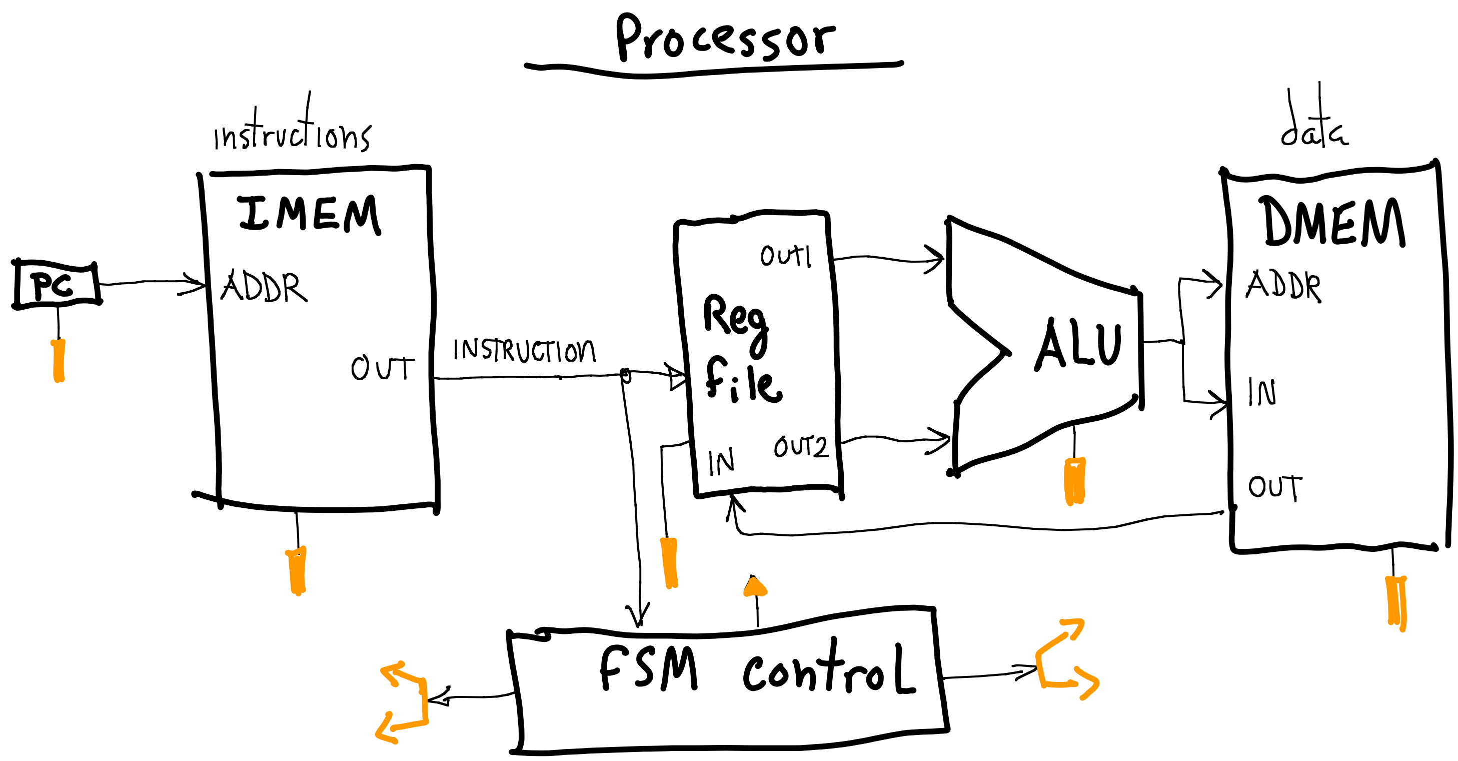
--- Not all instructions execute every phase.

--- Multiple instructions could be simultaneously in different phases. (How about same phases?)

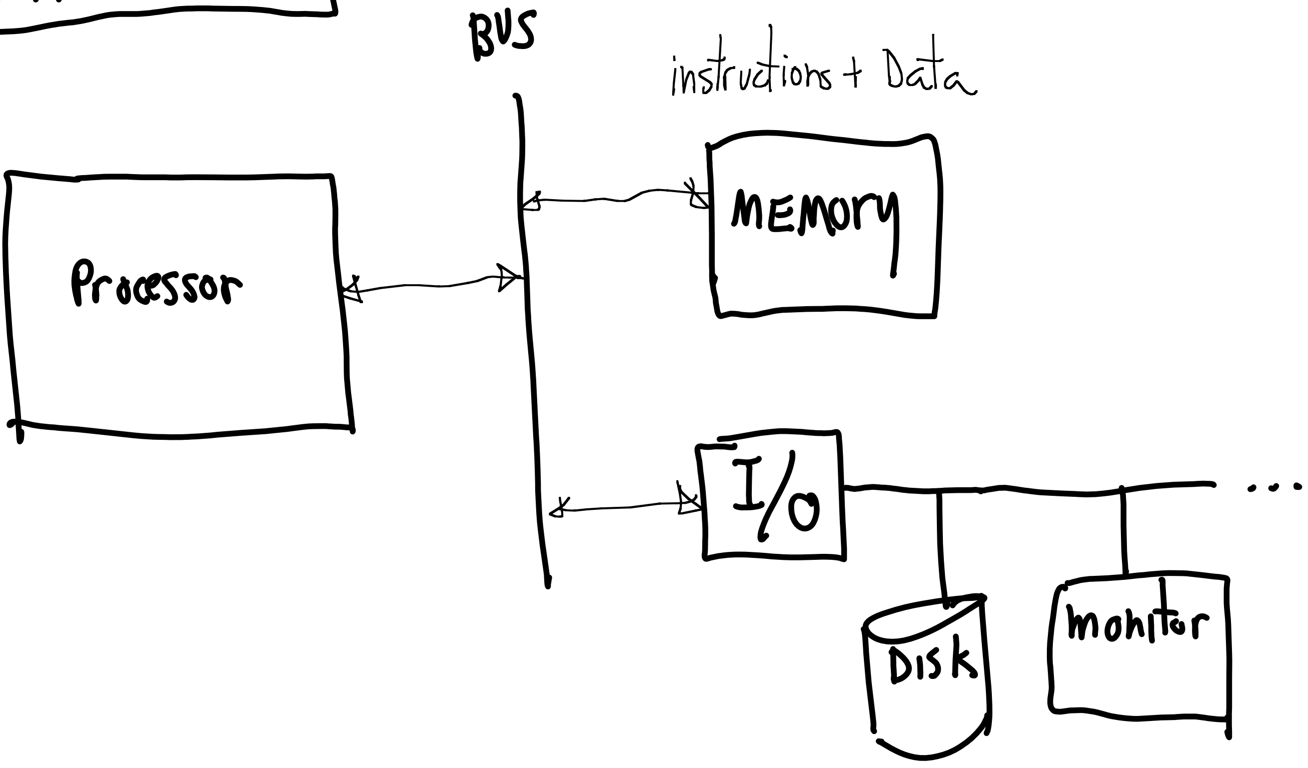
--- Some phases must wait for the previous phase to complete (eg., memory access)



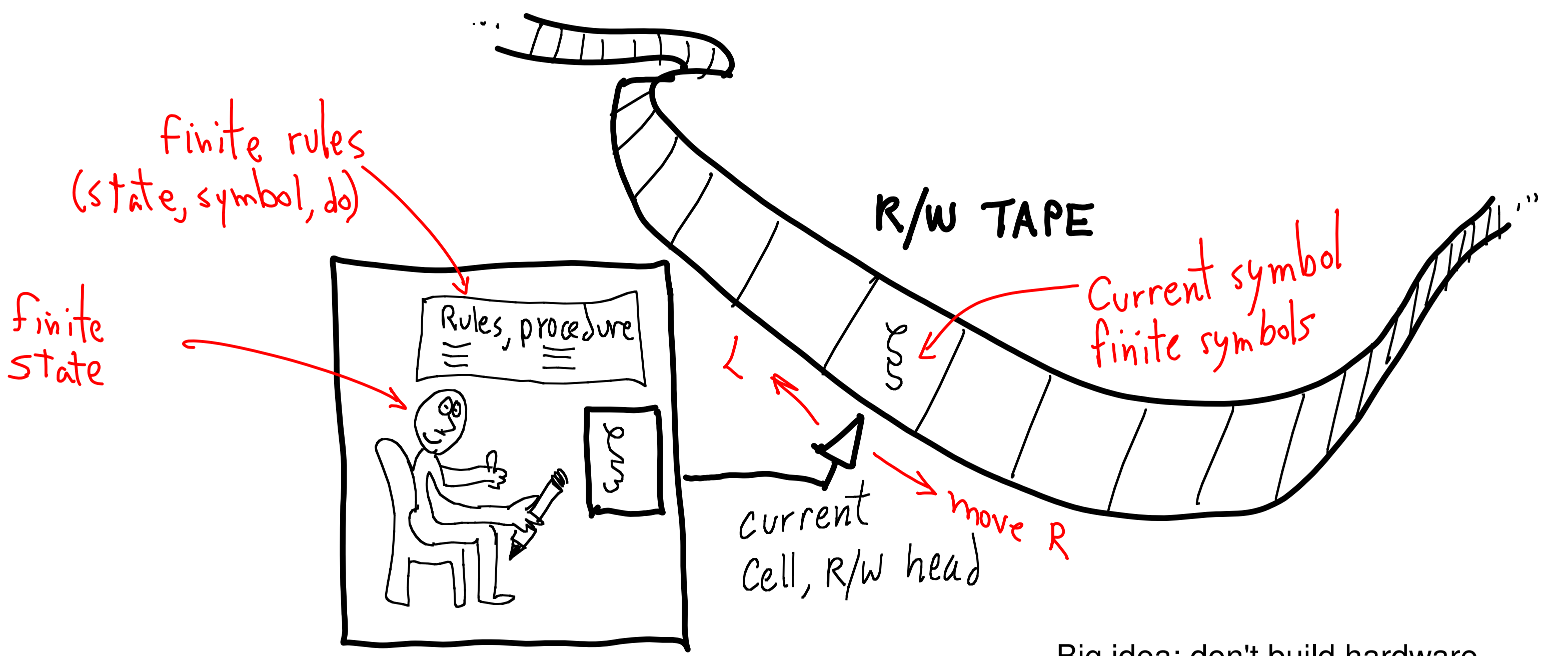
Harvard Architecture



System | von Neumann

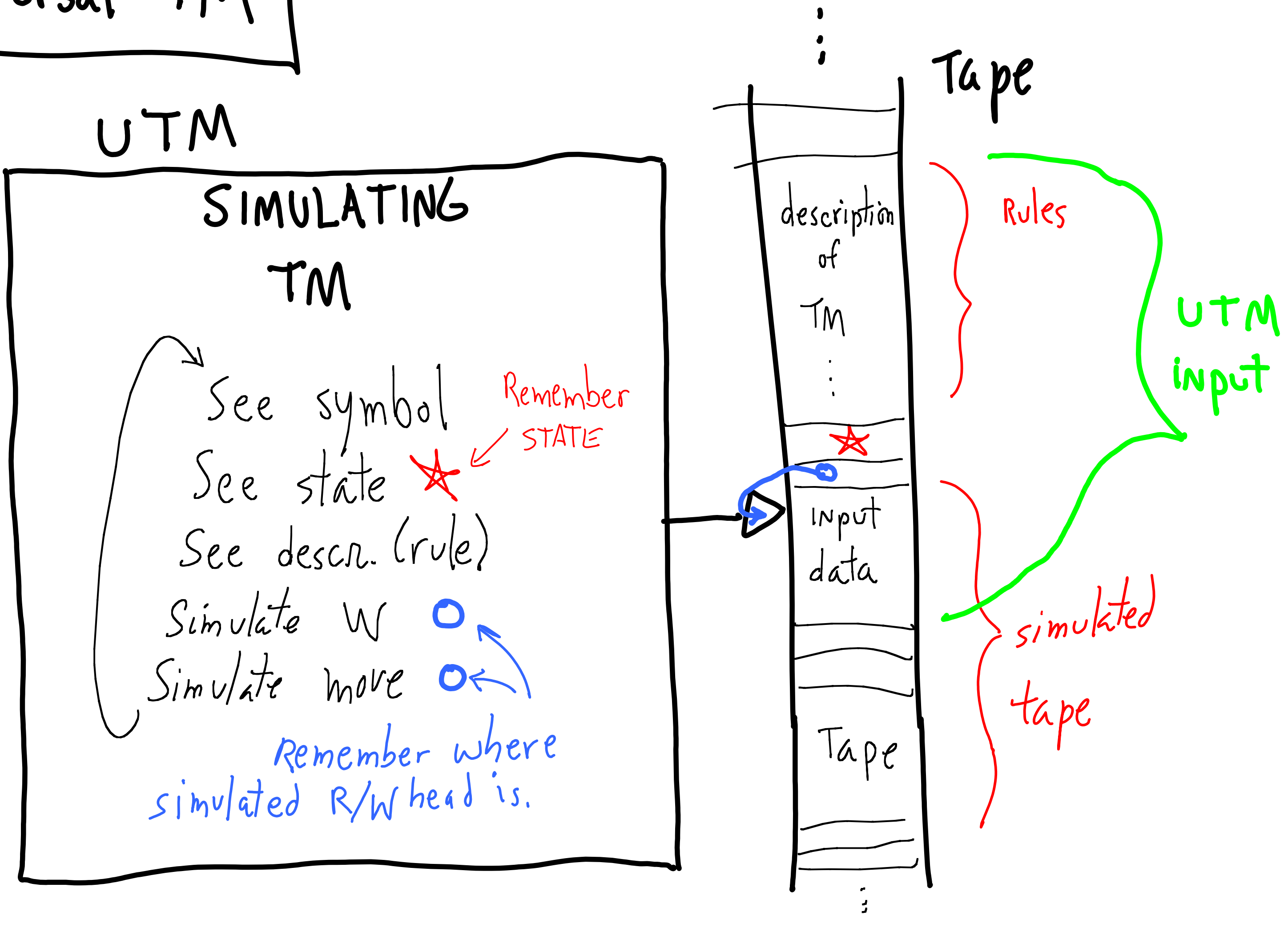


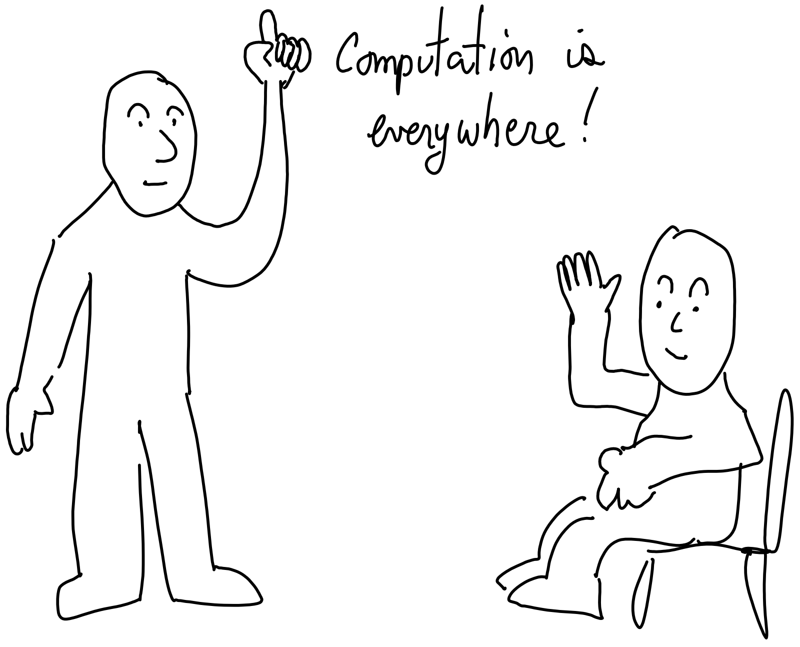
Turing Machine Concept



Big idea: don't build hardware, describe it and have it simulated == programming.

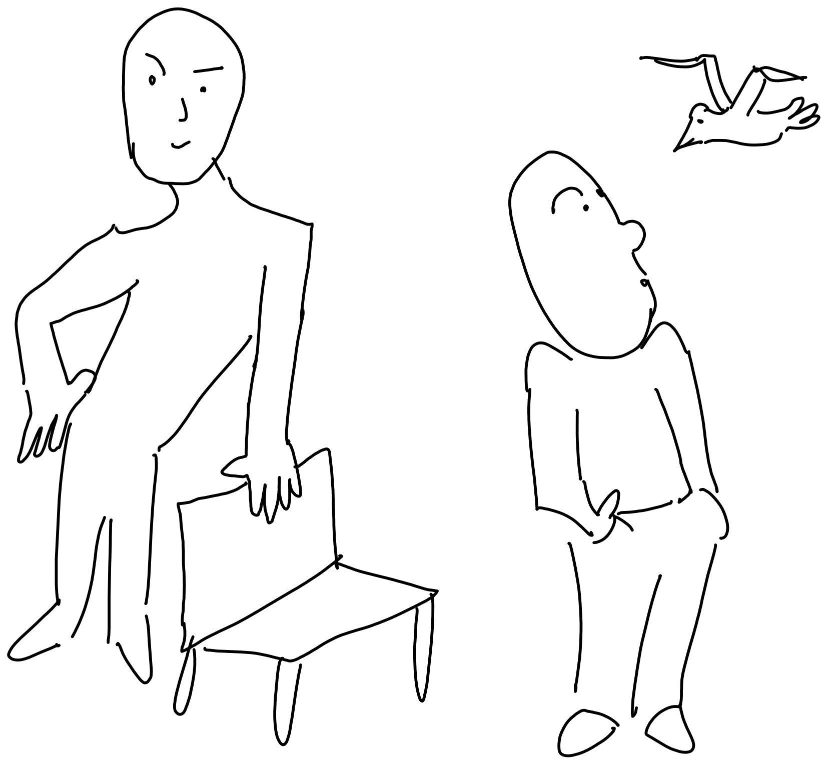
Universal TM

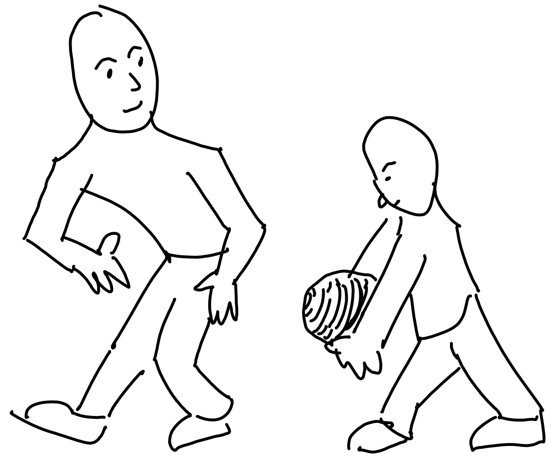




Eham: Computation is everywhere.
Drah: Where?
E: Everywhere!
D: A car crash?
E: Yes.
D: A doll house?
E: Yes.
D: Me?
E: Yes.
D: What is the same about them?
E: They all change.
D: So, computation is change?
E: Yes.
D: Everything changes, so computation is everywhere?
E: Yes.
D: What is computation?
E: Change.

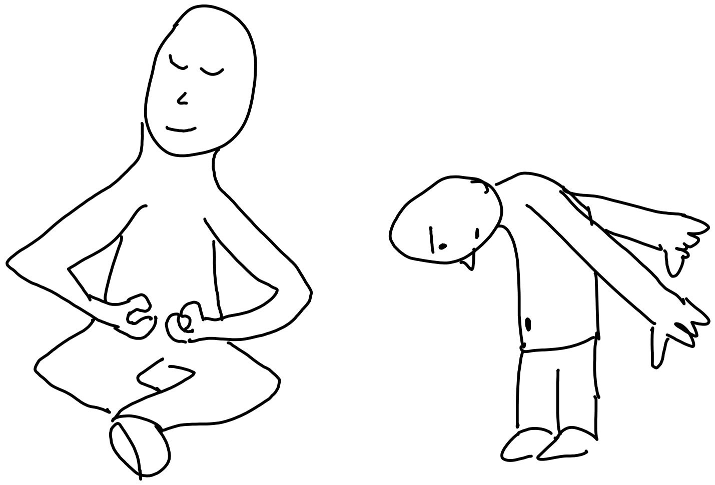
D: So, everything changes, and because everything changes, everything is computation, and computation is change.
E: Yes!
D: Oh.
E: You see, it is really quite simple.
D: How simple?
E: There is a model.
D: A model?
E: Yes.
D: How is there a model?
E: Things are one way, then they are another.
D: And that means there is a model?
E: Exactly.
D: How do I know there is a model?
E: That is an existence proof.
D: What is?
E: I just said there is a model, didn't I?



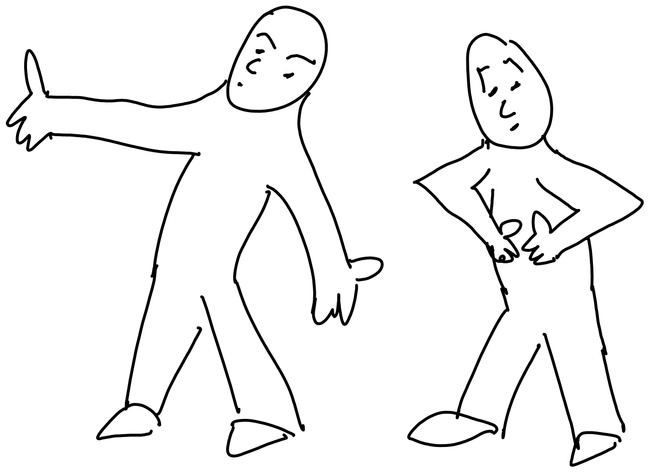


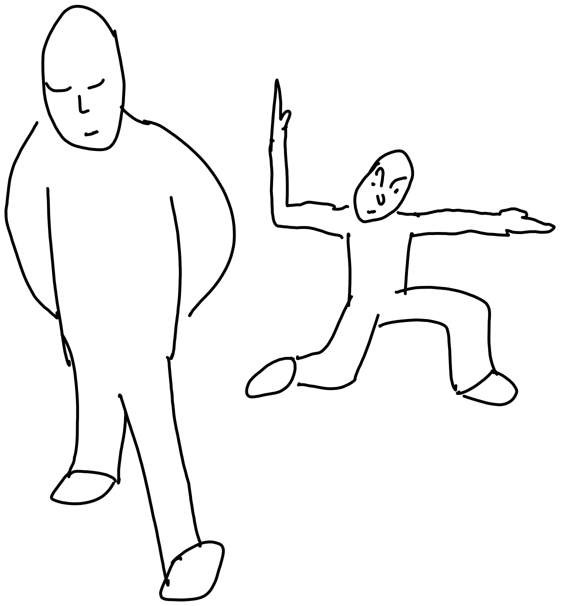
D: And a model means things are one way, then another.
E: Now you've got it.
D: Isn't that the same as change?
E: Quite right.
D: So, a model is change and change is computation and change is computation because there is a model?
E: See, now you're getting the hang of it.
D: Oh.

D: So, what is a computer?
E: Something that does computation.
D: Doing computation?
E: That's it, computing.
D: So, computers compute?
E: Obviously.
D: And computing is change?
E: What else could it be?
D: Everything changes, so everything is a computer?
E: Yes, absolutely.



D: I am a computer?
E: Without a doubt. When you change, which you do constantly, you are computation.
D: Then, I'm not me before, nor me after, but I'm me as I change?
E: Computation is everything and everywhere, all things are changing, you are changing, you are computation.
D: What if I don't change?
E: Everything changes.
D: So, there is nothing that doesn't change?
E: That's right, nothing doesn't change.
D: So nothing isn't computation. Does nothing exist?
E: Of course nothing exists. There is zero, zero exists.





D: So zero is not computation?
E: Yes, because zero is nothing. If it were something, then it would be computation, because all things change.
D: So, does one exist.
E: As surely as anything exists, as certainly as zero exists.
D: But they don't change, zero and one, I mean?
E: Of course not.
D: Then something exists which is not computation?
E: Absolutely.
D: But, if computation is everywhere, where is zero and one.
E: Right there.
D: Where? On the ceiling?
E: Of course. See that thing there? There is only one of them there.
D: So that's the existence of one?
E: What could be clearer?

D: Which part is the one which is not changing, and which part is the not-one which is changing?
E: Everything changes.
D: Oh. So, one is computation, too?
E: Of course not. Zero plus one, now that's computation.
D: Isn't that one?
E: Yes.
D: Did it change?
E: Did what change?
D: Well, the zero or the one, or something.
E: I have no patience for this. Numbers cannot change, they just are. Addition is a mapping, not change.
D: So, addition is not computation because nothing changes?
E: Don't be silly. If anything is computation, then addition is. Haven't you ever used a computer?
D: Yes, I guess.
E: I think you need to take more computer science classes so you will know something about computing. Then we can talk again. Run along now.
D: Thank you, professor Eham.