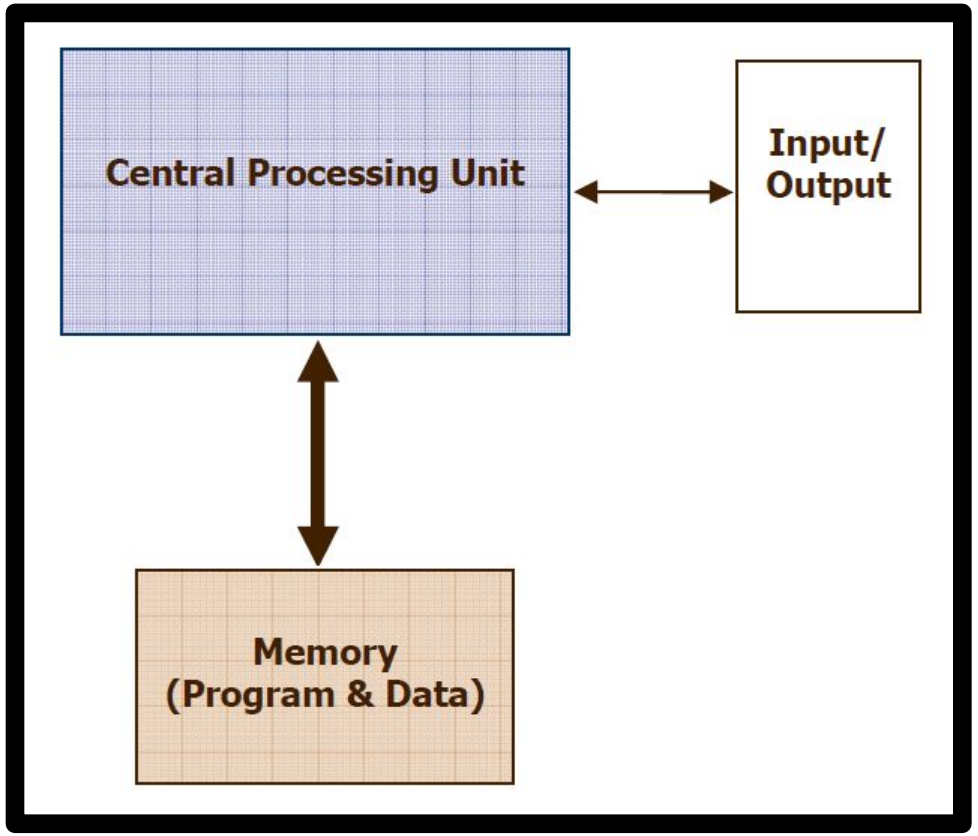
Due Dates and Announcements

1. Report should include demonstration of working CPU (waveform simulator, or video of the board in action) – Upload report on eCampus
    1. Deadline Monday December 4th
    2. **Try not to wait until the last minute**
    3. Explain how it all works together/what each component does **in your own words (follow "Contents of Report" on eCampus for more details)**
2. "Lab Portfolio" submission Friday December 8th

Project Overview

A microprocessor can

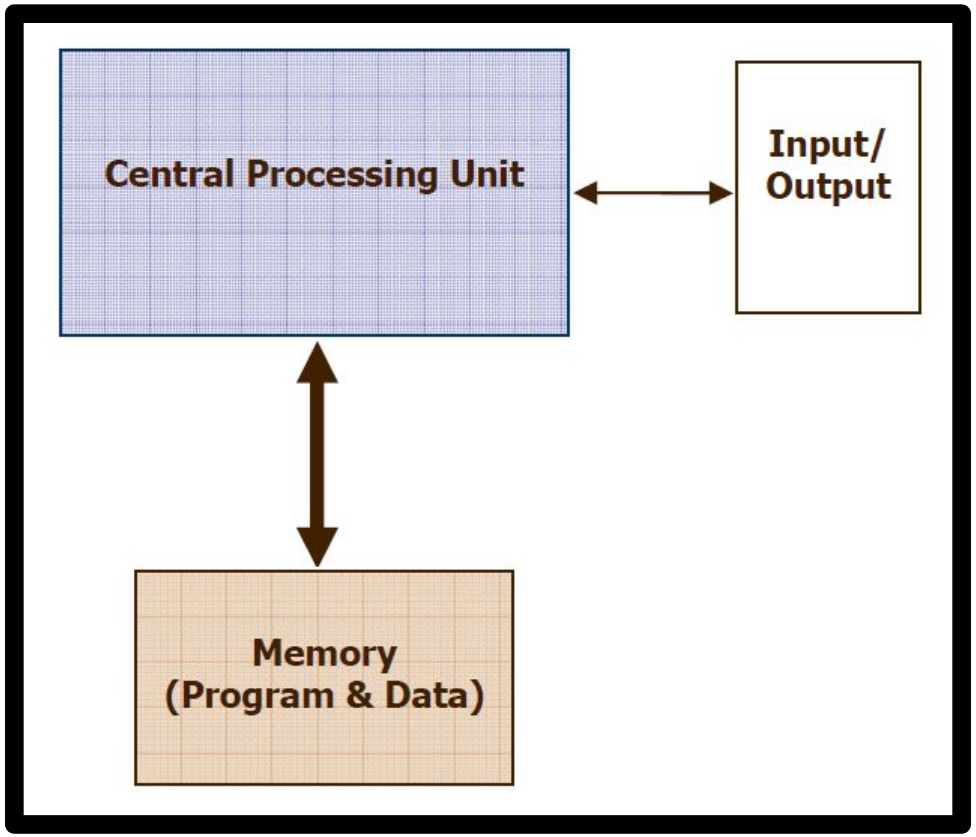
- be modeled as a programmable state machine
- perform operations that are stored in memory.



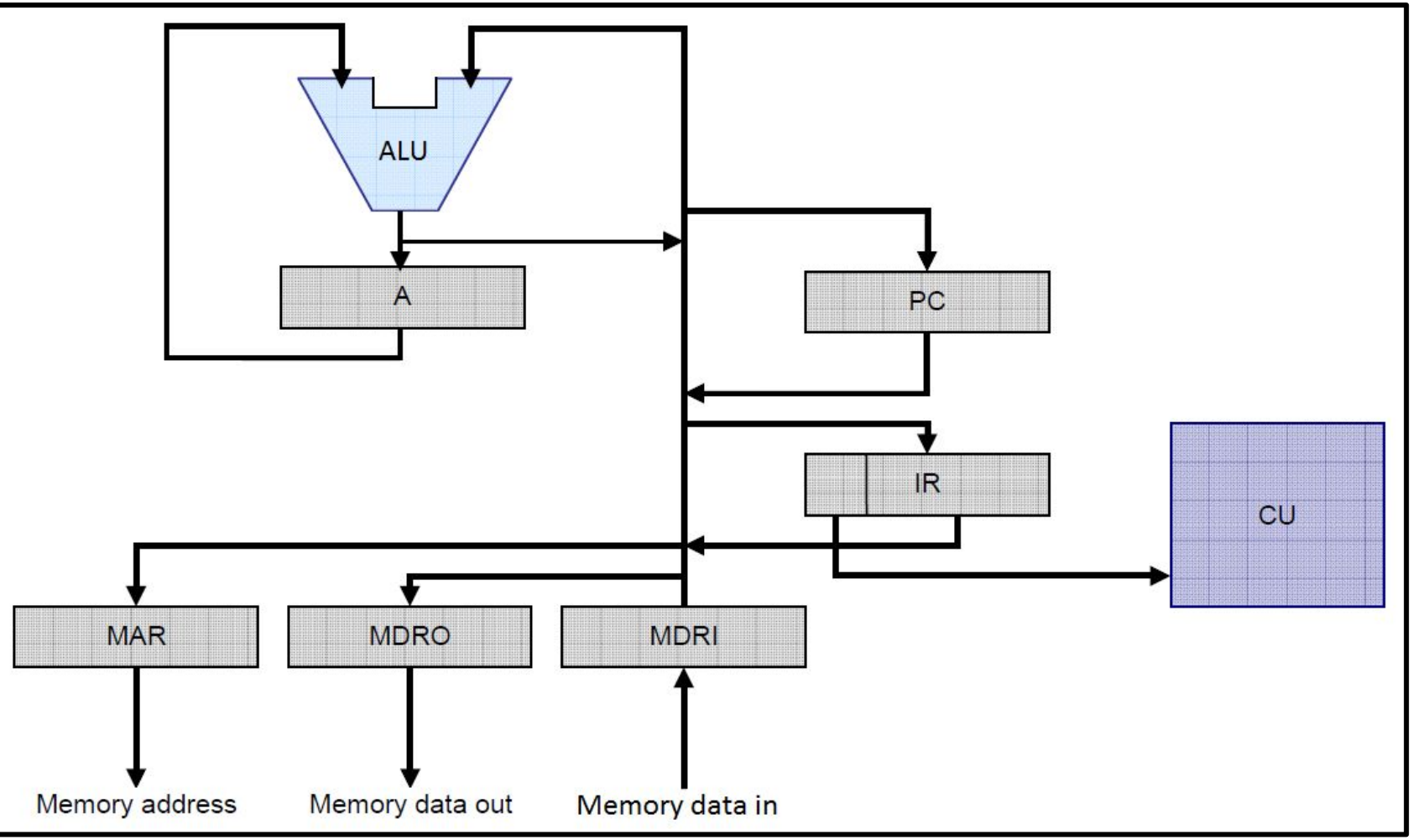
Project Overview

Operationally the microprocessor…

1. Retrieves an instruction from memory
2. Decodes the instruction to determine what actions need to be performed
3. Performs the necessary actions
4. Begins retrieving the next instruction
   - i.e. starts back at #1

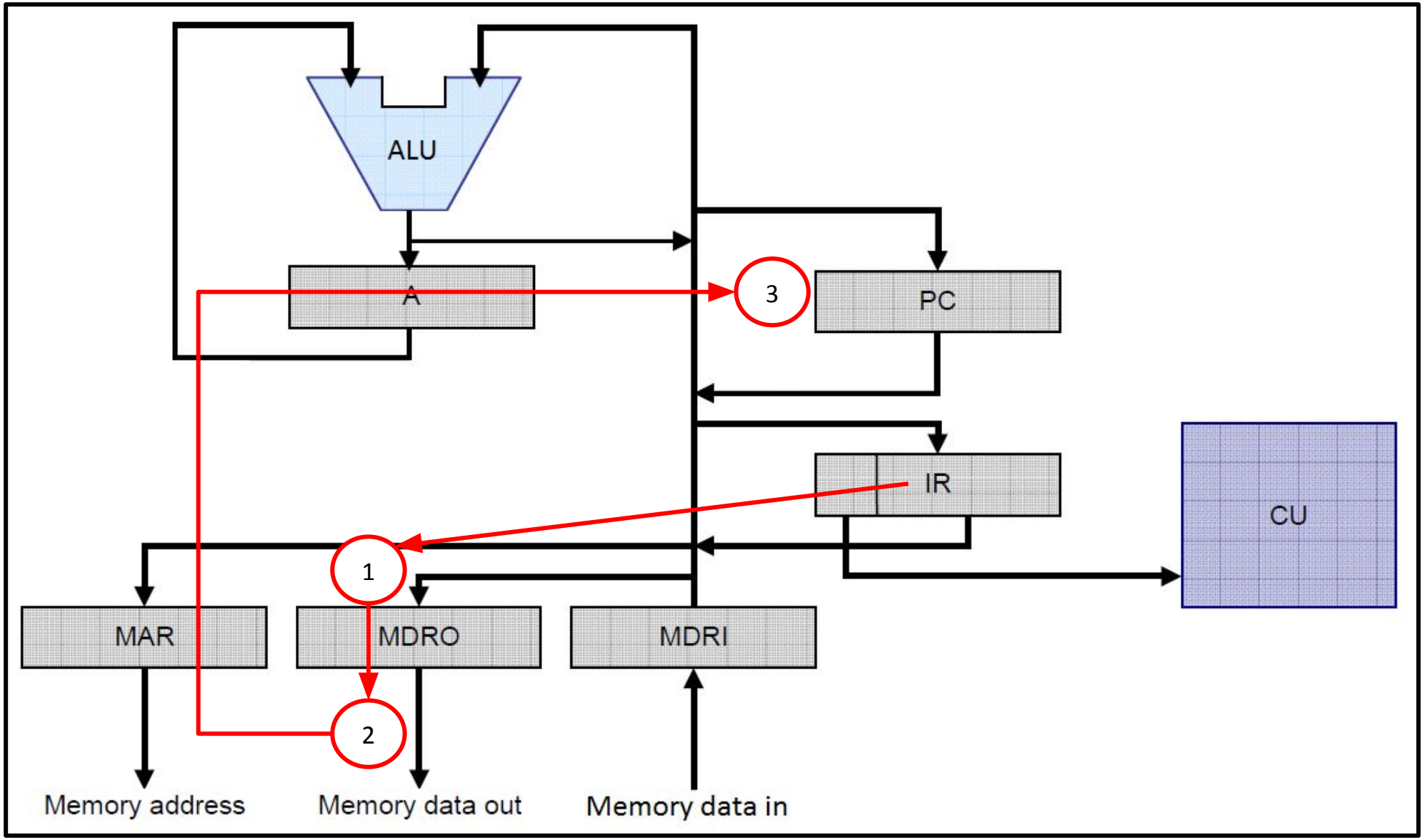


Simplified CPU Architecture



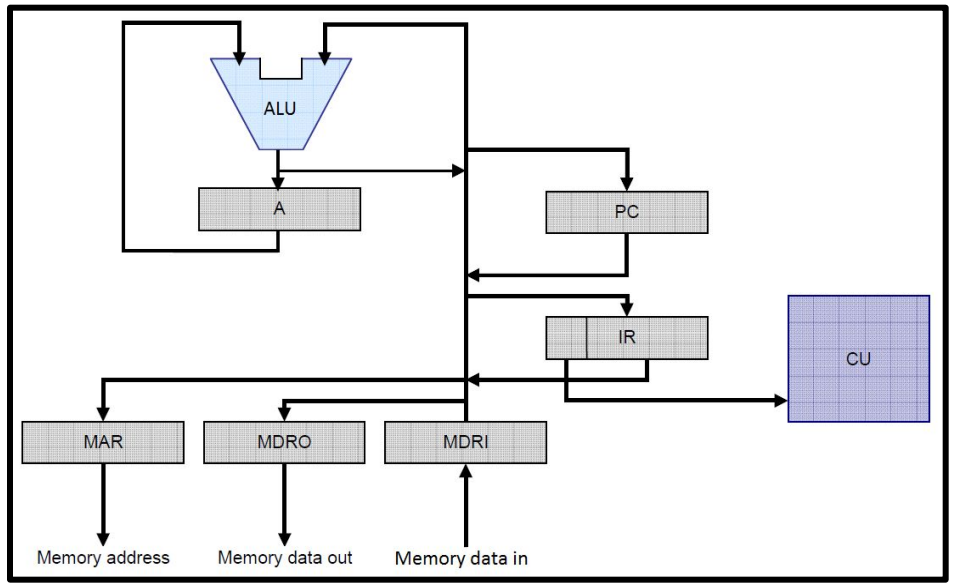
Pros *and* Cons of this project:
It can be completely coded at home.

Simplified CPU Architecture



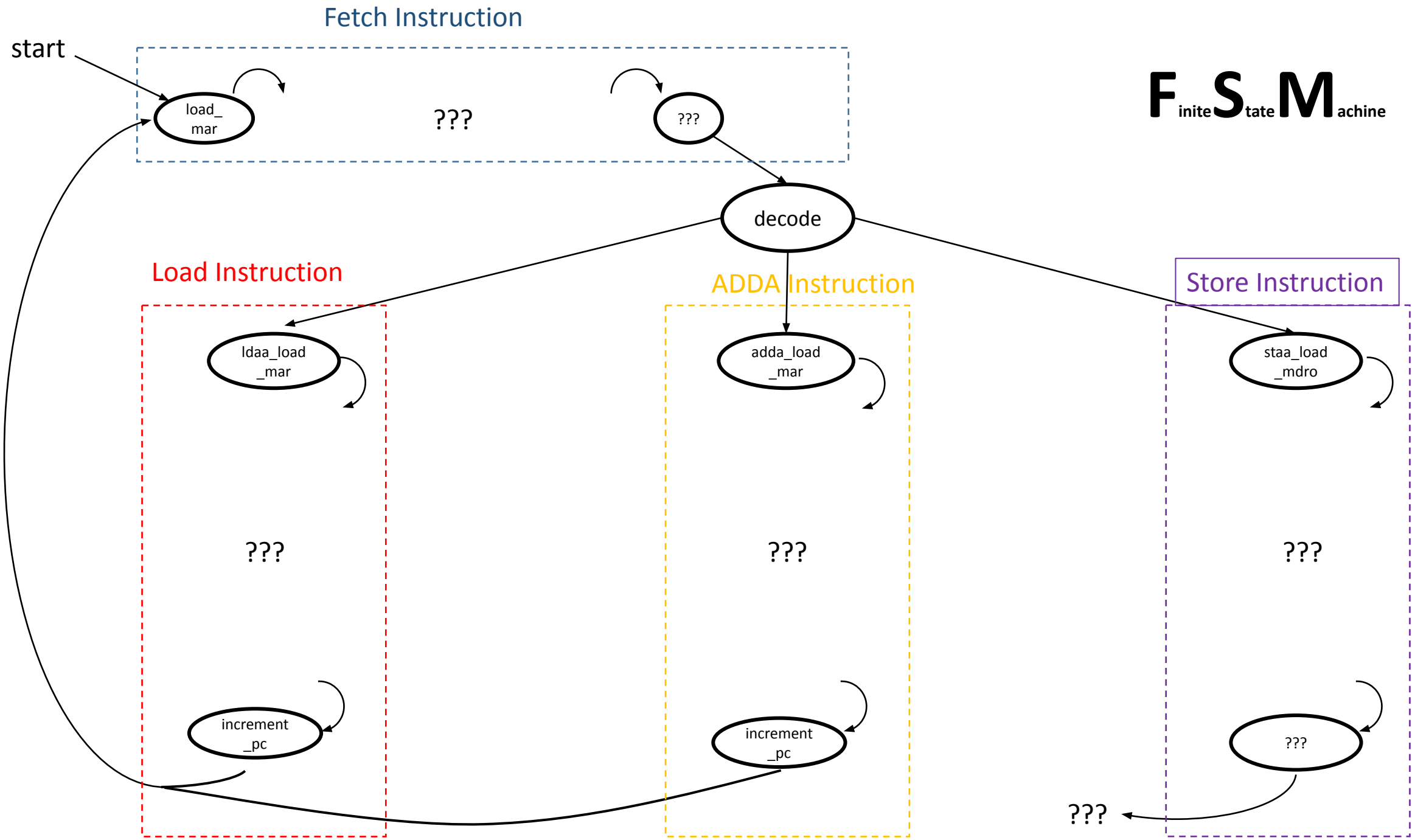
Pros *and* Cons of this project: It can be completely coded at home.

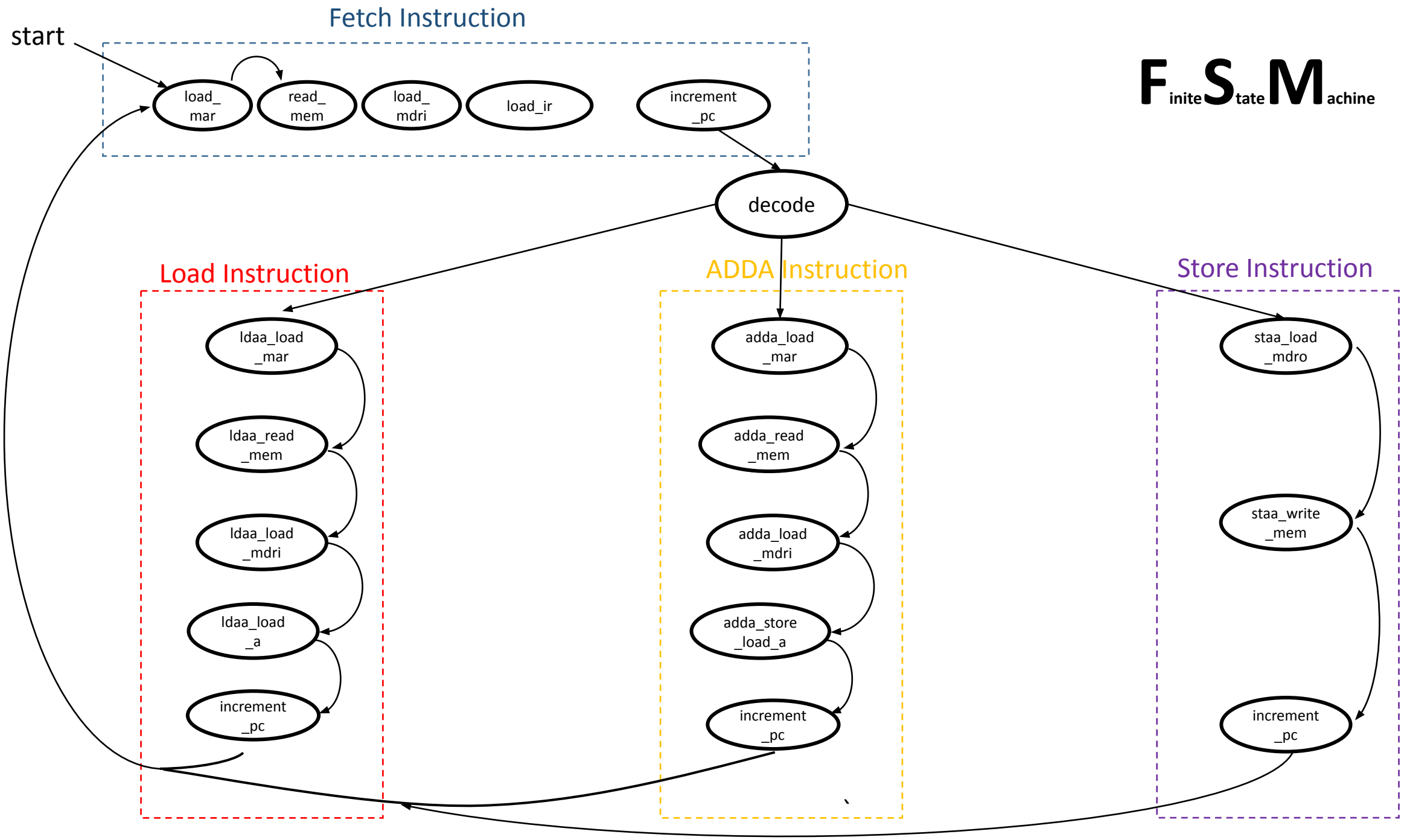
Full CPU Architecture



Your job!

See my “in-depth notes” to get started on expanding this diagram

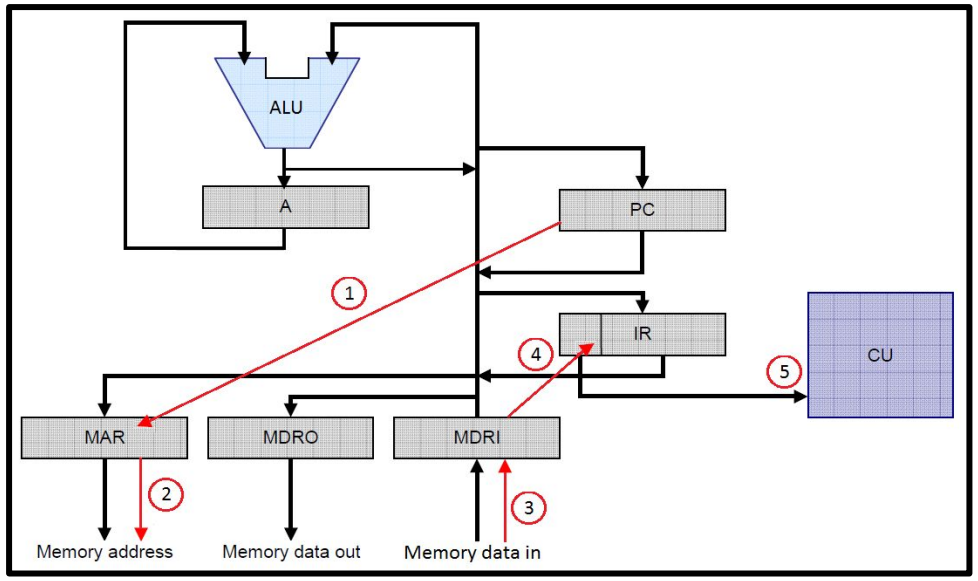

Fetch Instruction
start
load_
mar
???
???
decode
Load Instruction
ADDA Instruction
Store Instruction
ldaa_load
_mar
adda_load
_mar
staa_load
_mdro
???
???
???
increment
_pc
increment
_pc
???
???
FSM
Finite State Machine

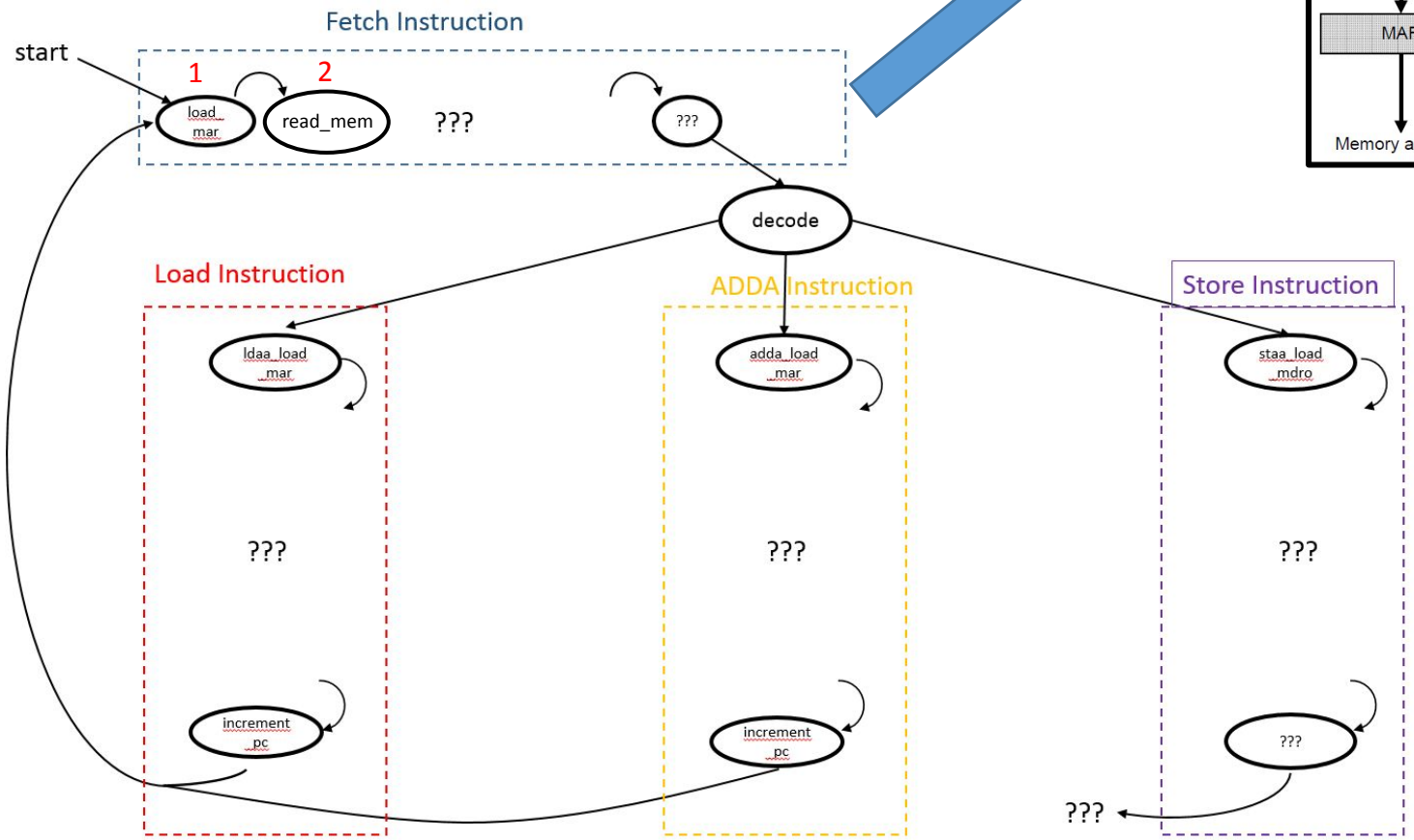


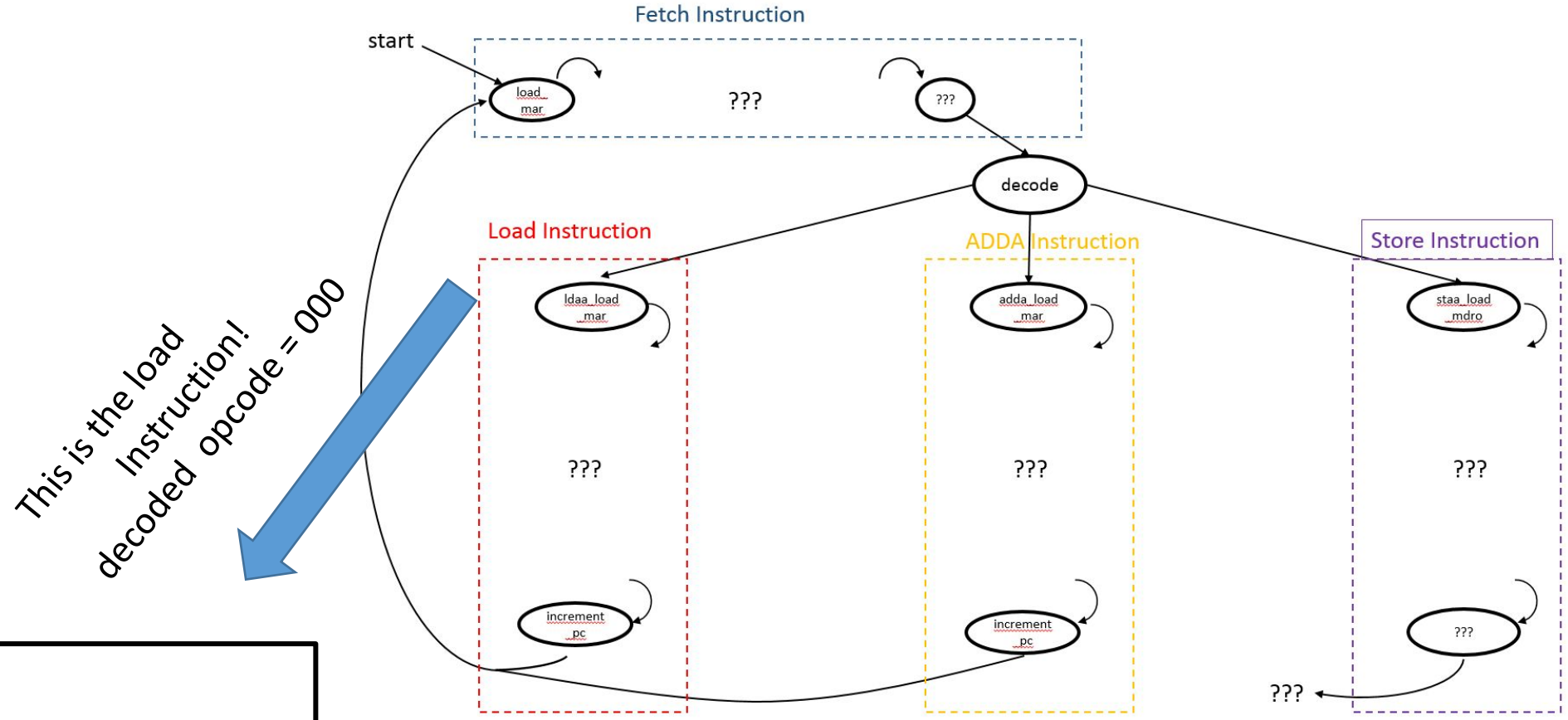
Finite **S**tate **M**achine

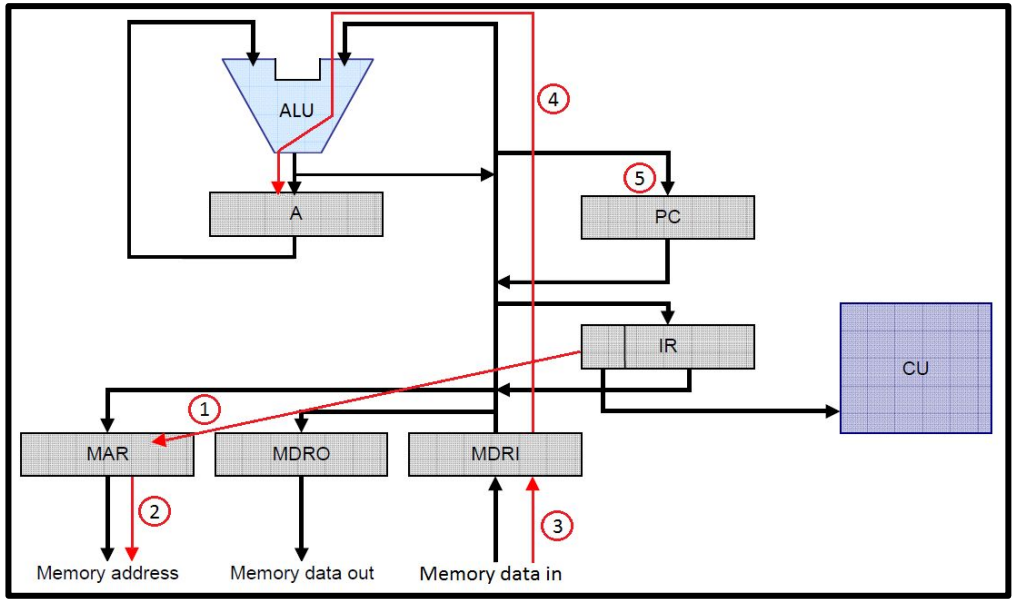
Described to you on page 17

This is the fetch Instruction!



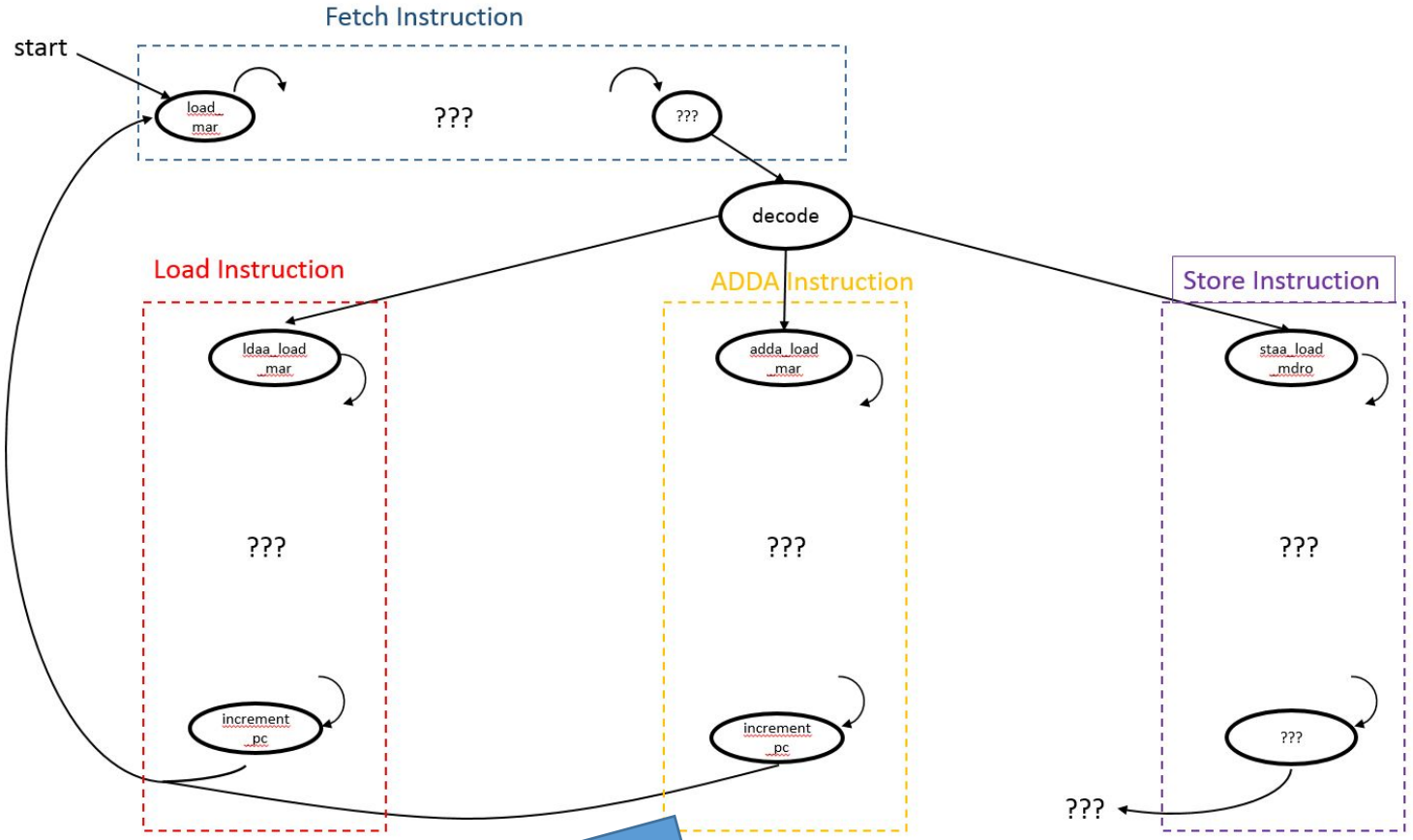


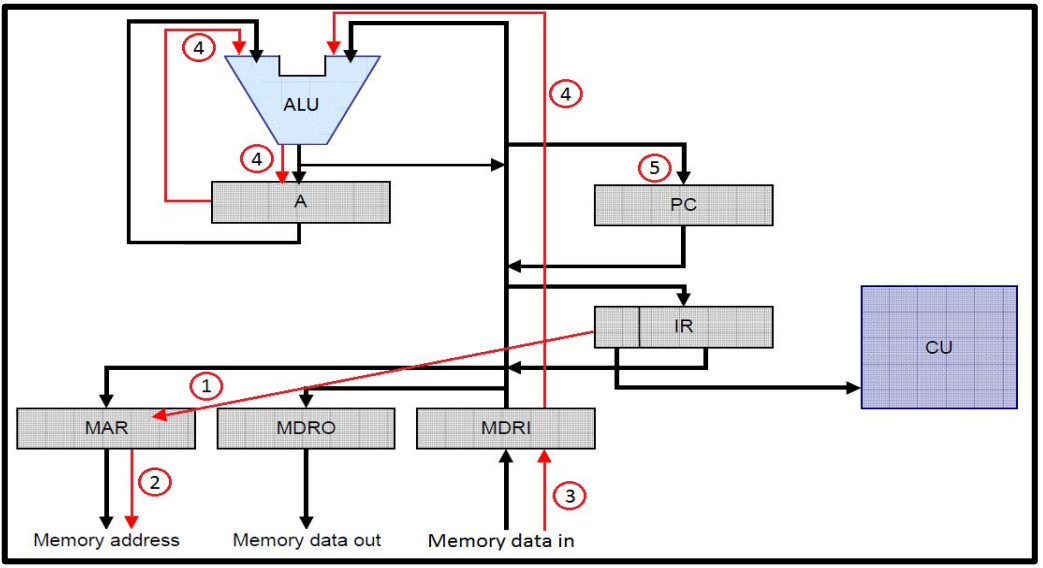




Described to you on page 18

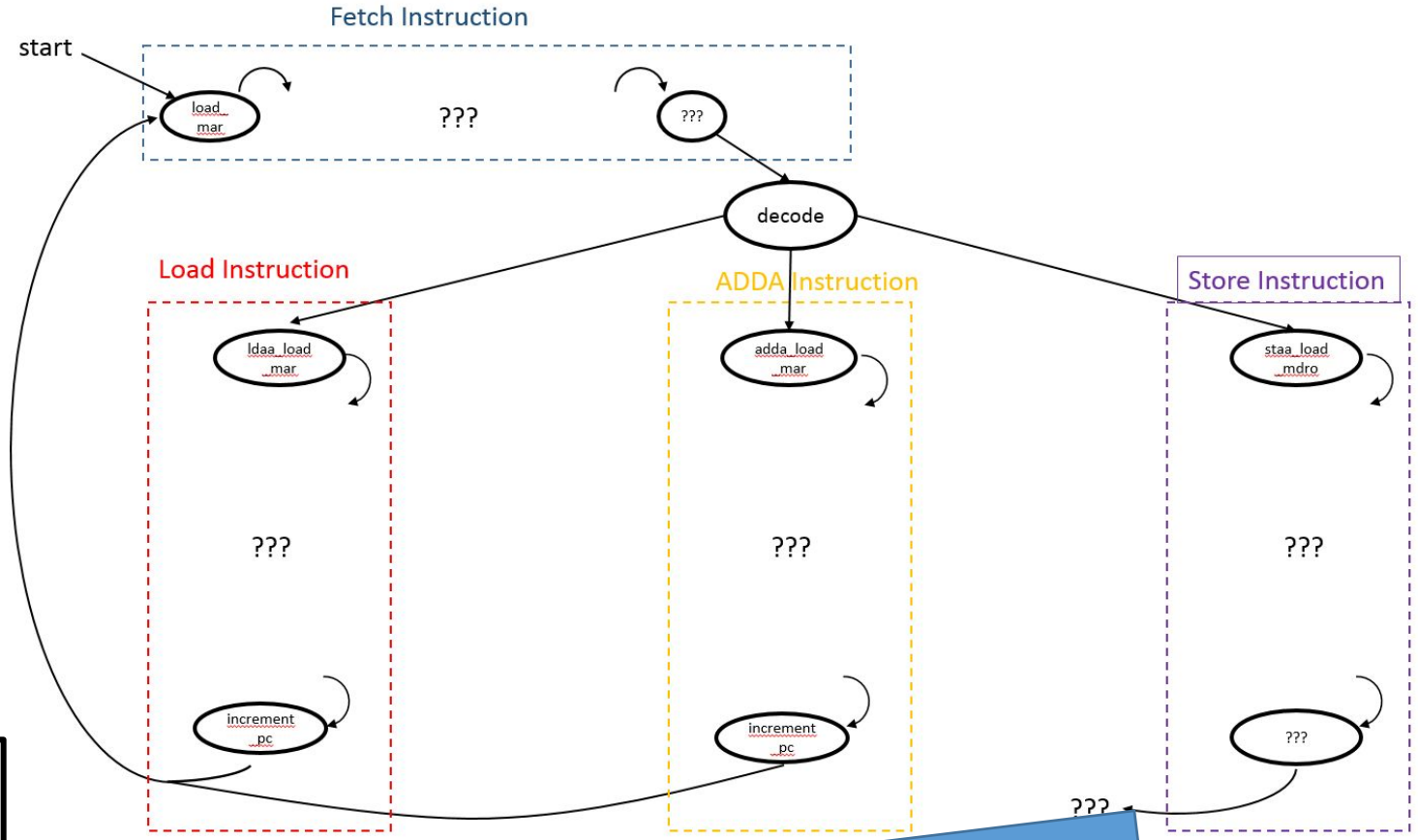
Described to you on page 19

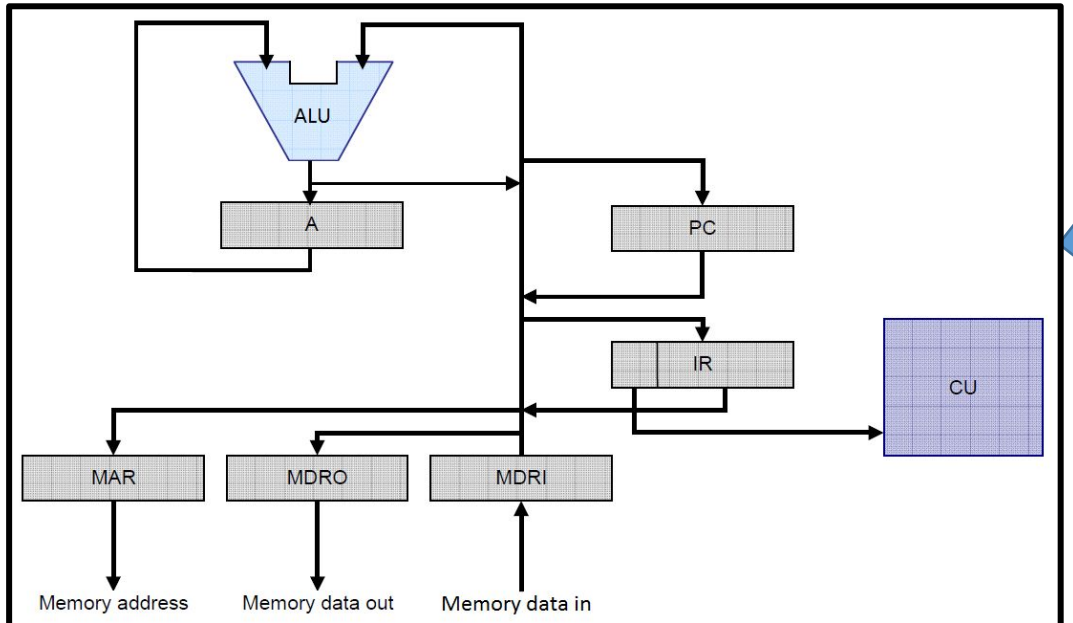




This is the add Instruction!
decoded opcode = 001

Described to you on page 20





This is the store Instruction!
decoded opcode = 010

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

Mem contents that are hardcoded into our VHDL for us – hardcoding the values same as we did with the RAM experiment

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101				
1	00100011				
2	01000111				
3	00000111				
4	00101000				
5	00000110				
6	00010100				
7	00001101				
8	00000001				

Pin Outputs

- Show the output on LEDs for the following:
  - aOut (Accumulator Output)
  - irOutput (IR Output)
  - pcOUT (PC Output)
- Set the clk to a pushbutton to increment each instruction from memory
- **IF outputs are shown on 7-Segment Displays +5 Bonus Points**

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the first instruction in address 0 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this first instruction will *load* the memory *contents* in address 5 in the accumulator

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5		
1	00100011				
2	01000111				
3	00000111				
4	00101000				
5	00000110	←	6_{10}		
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the first instruction in address 0 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this first instruction will *load* the memory *contents* in address 5 in the accumulator

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011				
2	01000111				
3	00000111				
4	00101000				
5	00000110				
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the 2nd instruction in address 1 and parse it

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011				
2	01000111				
3	00000111				
4	00101000				
5	00000110				
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the 2nd instruction in address 1 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this instruction will *add* the memory *contents* in address 6 *with* the amount currently stored in the accumulator (which is 10)

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011	001 – addA	00011 – 3		
2	01000111				
3	00000111	← 7_{10}			
4	00101000				
5	00000110				
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the 2nd instruction in address 1 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this instruction will *add* the memory *contents* in address 6 *with* the amount currently stored in the accumulator (which is 10)

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011	001 – addA	00011 – 3	7	13
2	01000111				
3	00000111	← 7_{10}			
4	00101000				
5	00000110				
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the 3rd instruction in address 2 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this instruction will *store* the current accumulator value in memory at address 7

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011	001 – addA	00011 – 3	7	13
2	01000111	010 – storeA	00111 – 7		
3	00000111				
4	00101000				
5	00000110				
6	00010100				
7	01010101				
8	00000001				

Let’s say it’s all coded up correctly… How will this monstrosity work on the board?

We *fetch* the 3rd instruction in address 2 and parse it

- First 3 MSBs represent the OpCode
- Last 5 bits represents another address in memory

Looks like this instruction will *store* the current accumulator value in memory at address 7

Memory Data					
RAM	Binary	OpCode	Address	Value	Accumulator
0	00000101	000 – loadA	00101 – 5	6	6
1	00100011	001 – addA	00011 – 3	7	13
2	01000111	010 – storeA	00111 – 7	13	--
3	00000111	000 – loadA	00111 – 7	30	30
4	00101000	001 – addA	01000 – 8	1	31
5	00000110				
6	00010100				
7	00001101	← Was 01010101, now 00001101			
8	00000001				