Today

- Review
  - State machine
  - Circuit blocks
  - LDI computer
- Today
  - Tracing through execution of instructions

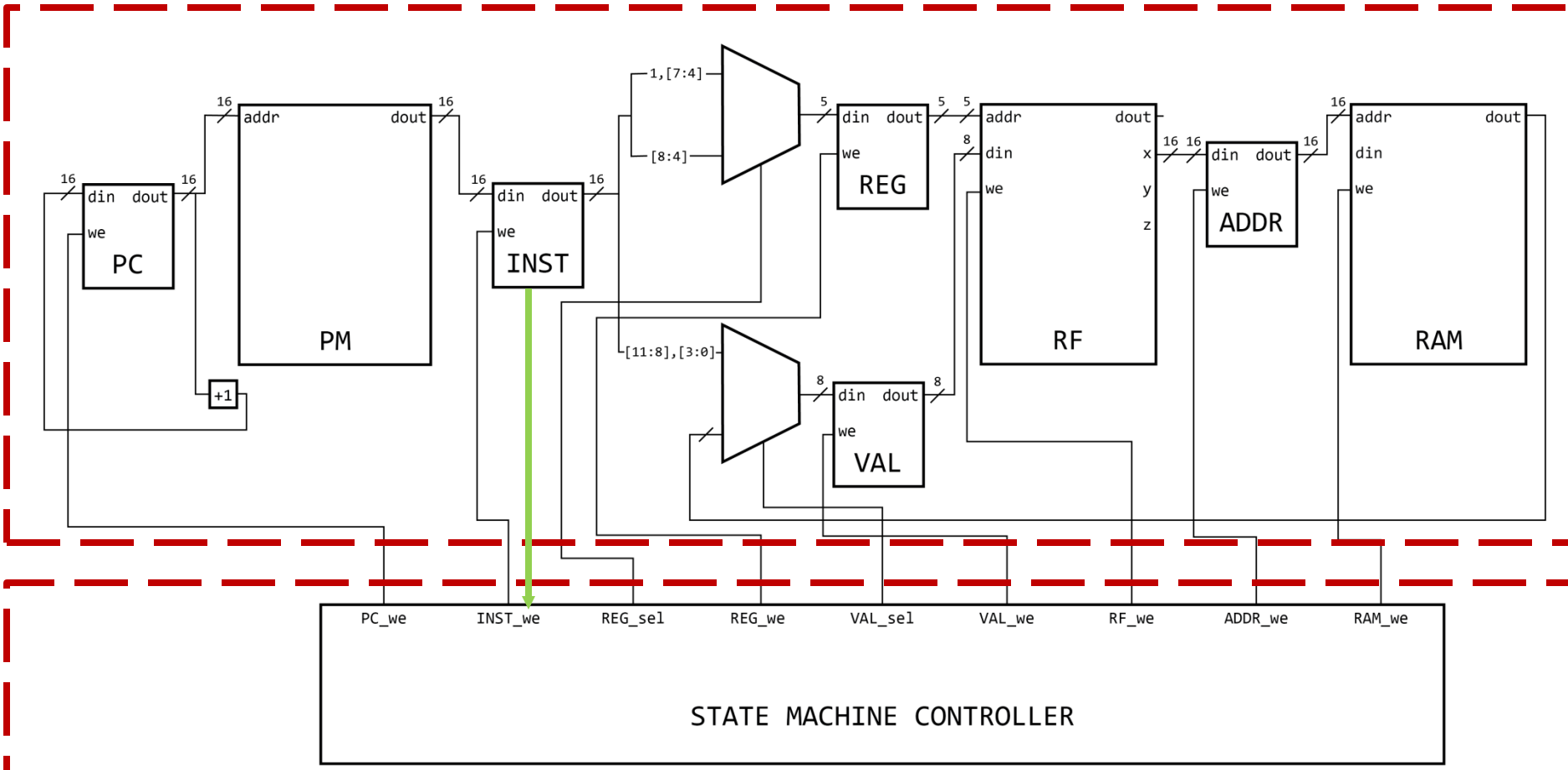
Concept

- Every instruction can be processed as 5 stages
  - Fetch, decode, execute, memory write-back
- Each stage maybe multiple “states”
- Each state takes one clock cycle
- State machine represents how the machine transitions to a set of states for each instruction

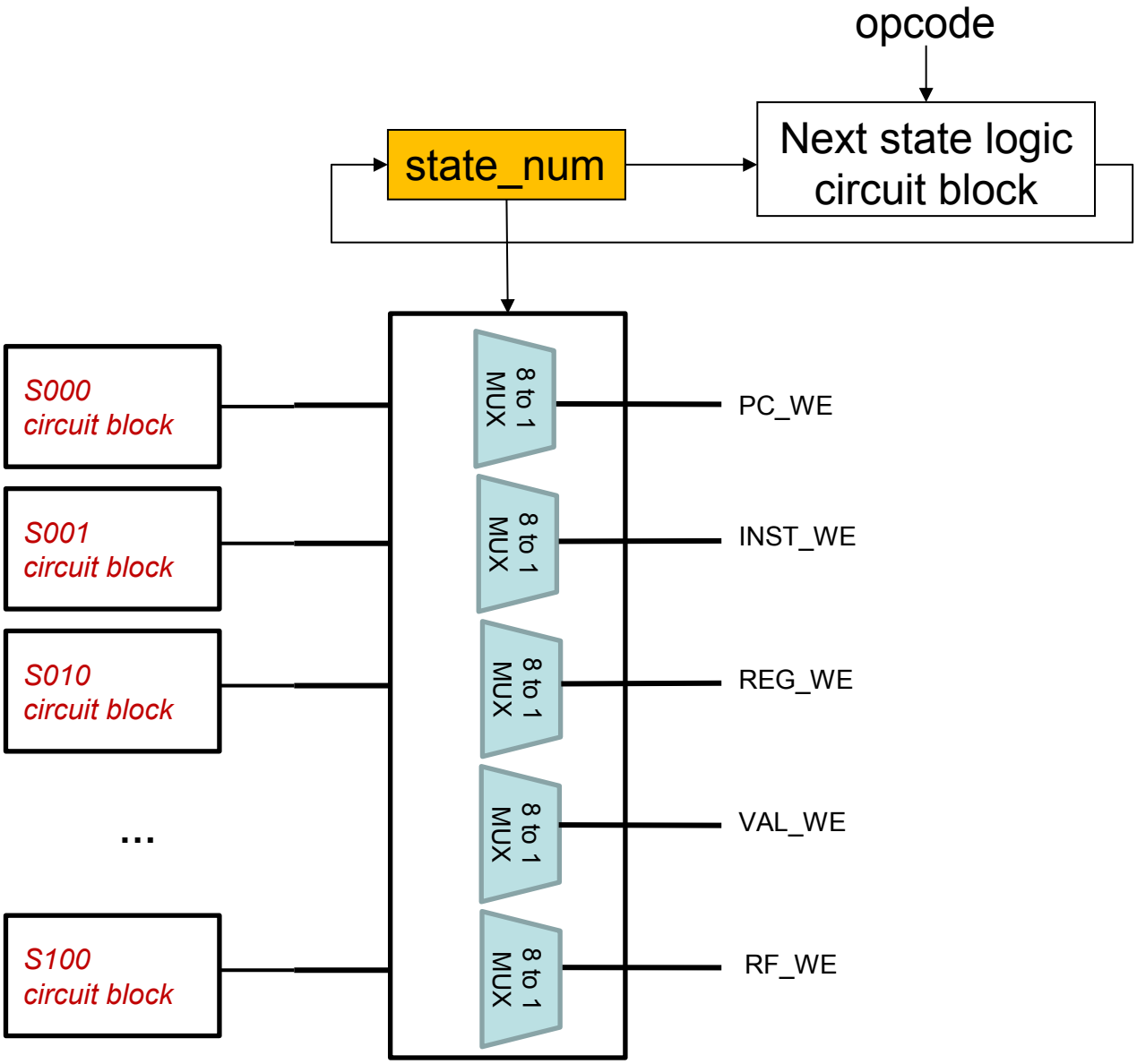
Implementation

- Datapath
  - Memories
  - Auxiliary registers
  - Circuit blocks
  - Wires connecting these all together
  - Control signals from state-machine controller
- Control-path (State machine controller)
  - Control signals at each cycle (state)
  - Logic for next state

Implementation



State machine controller

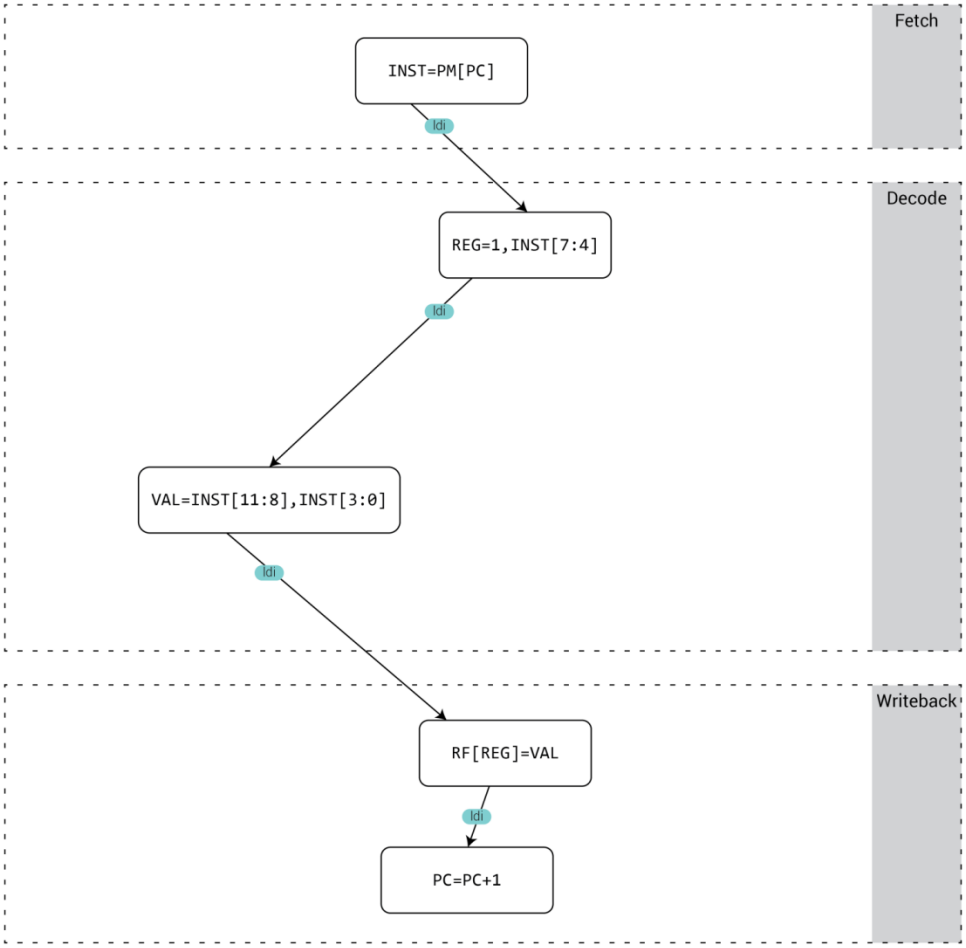


State transition table

		PC_we	INST_we	REG_we	REG_sel	VAL_we	VAL_sel	RF_we	ADDR_we	RAM_we
000	INST=PM[PC]	0	1	0		0		0	0	0
001	REG=1,INST[7:4]	0	0	1	0	0		0	0	0
010	REG=INST[8:4]	0	0	1	1	0		0	0	0
011	ADDR=X	0	0	0		0		0	1	0
100	VAL=INST[11:8],INST[3:0]	0	0	0		1	0	0	0	0
101	VAL=RAM[ADDR]	0	0	0		1	1	0	0	0
110	RF[REG]=VAL	0	0	0		0		1	0	0
111	PC=PC+1	1	0	0		0		0	0	0

State transition table

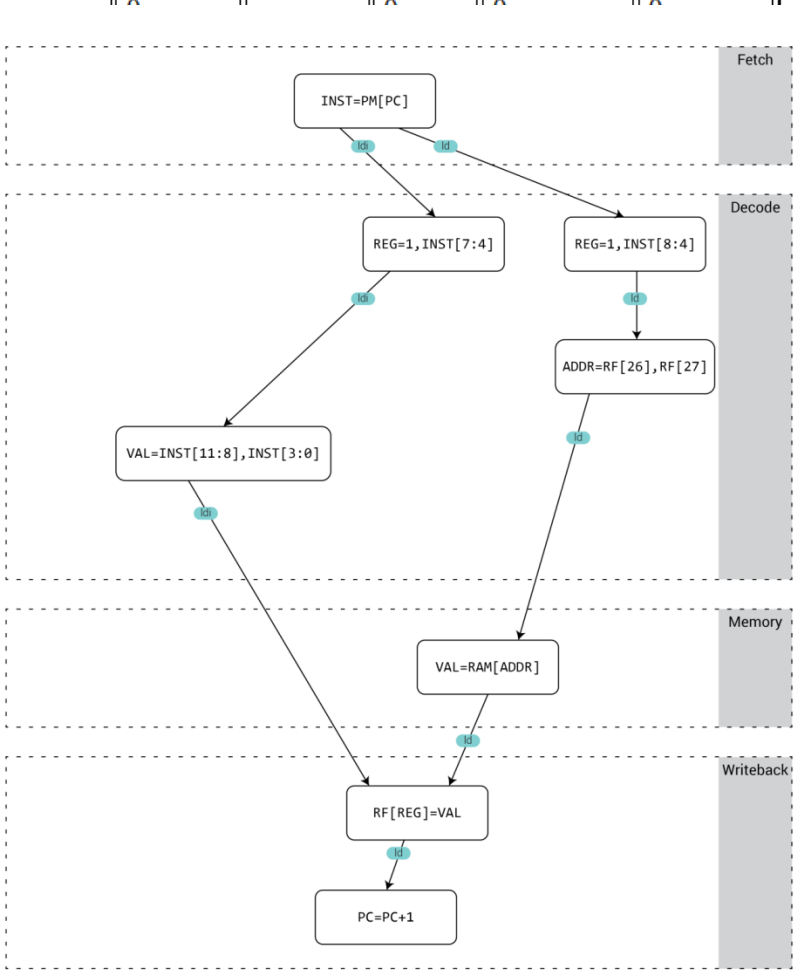
		PC_we	INST_we	REG_we	REG_sel	VAL_we	VAL_sel	RF_we	ADDR_we	RAM_we
000	INST=PM[PC]	0	1							
001	REG=1,INST[7:4]	0	0							
010	REG=INST[8:4]	0	0							
011	ADDR=X	0	0							
100	VAL=INST[11:8],INST[3:0]	0	0							
101	VAL=RAM[ADDR]	0	0							
110	RF[REG]=VAL	0	0							
111	PC=PC+1	1	0							



LDI:
ldi <reg>, imm
000 (Fetch)
001 (Decode)
100 (Decode)
110 (Writeback)
111 (Writeback)

State transition table

		PC_we	INST_we	REG_we	REG_sel	VAL_we	VAL_sel	RF_we	ADDR_we	RAM_we
000	INST=PM[PC]	0	1	0						
001	REG=1,INST[7:4]	0	0	1						
010	REG=INST[8:4]	0	0	1						
011	ADDR=X	0	0	0						
100	VAL=INST[11:8],INST[3:0]	0	0	0						
101	VAL=RAM[ADDR]	0	0	0						
110	RF[REG]=VAL	0	0	0						
111	PC=PC+1	1	0	0						



LD:
ld <reg>, X
- 000 (Fetch)
- 010 (Decode)
- 011 (Decode)
- 101 (Memory)
- 110 (Writeback)
- 111 (Writeback)

Next state logic circuit block

From CH6

Current-state	OPCODE	Next-state
000	1110	001
001	1110	100
100	1110	110
110	1110	111
111	1110	000
000	1010	010
010	1010	011
011	1010	101
101	1010	110
110	1010	111
110	1010	000

LDI:
ldi <reg>, imm
000 (Fetch)
001 (Decode)
100 (Decode)
110 (Writeback)
111 (Writeback)

LD:
ld <reg>, X
000 (Fetch)
010 (Decode)
011 (Decode)
101 (Memory)
110 (Writeback)
111 (Writeback)

Inst.	Opcode
ldi	1110
ld	**1010**
mov	001011
add	000011
sub	000110
inc	10010100011
dec	10010101010
and	001000
andi	0111
or	001010
ori	0110
com	10010100000
rjmp	1100
breq	111100001
brne	111101001
brsh	111101000
brlo	111100000
cp	000101
cpi	0011

Next state logic circuit block

Current-state	OPCODE	Next-state
000	1110	001
001	1110	100
100	1110	110
110	1110	111
111	1110	000
000	1010	010
010	1010	011
011	1010	101
101	1010	110
110	1010	111
110	1010	000

LDI:
ldi <reg>, imm
000 (Fetch)
001 (Decode)
100 (Decode)
110 (Writeback)
111 (Writeback)

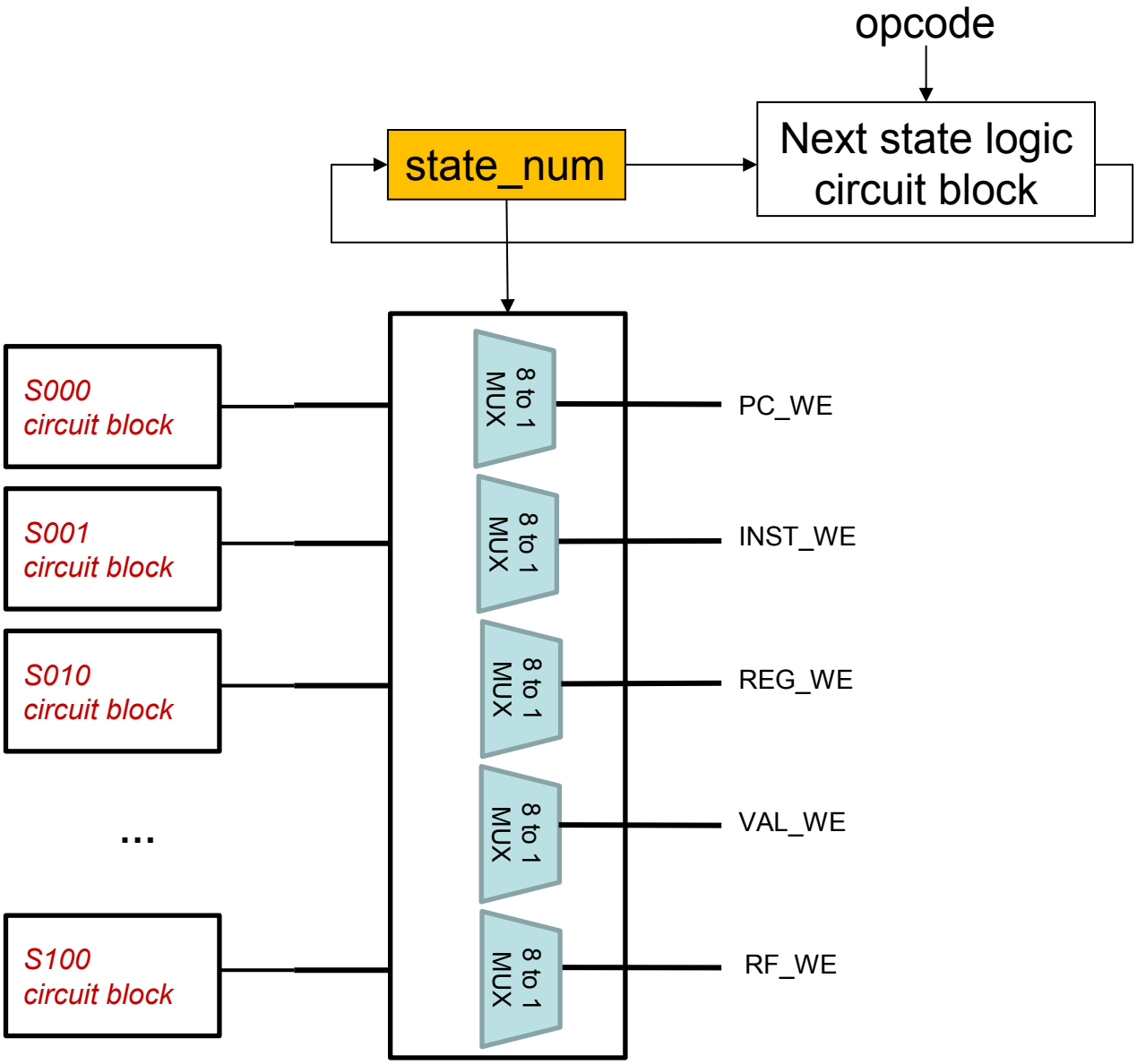
LD:
ld <reg>, X
000 (Fetch)
010 (Decode)
011 (Decode)
101 (Memory)
110 (Writeback)
111 (Writeback)

Inst.	Opcode
ldi	1110
ld	**1010**
mov	001011
add	000011
sub	000110
inc	10010100011
dec	10010101010
and	001000
andi	0111
or	001010
ori	0110
com	10010100000
rjmp	1100
breq	111100001
brne	111101001
brsh	111101000
brlo	111100000
cp	000101
cpi	0011

252's Cool Axiom for Ch7

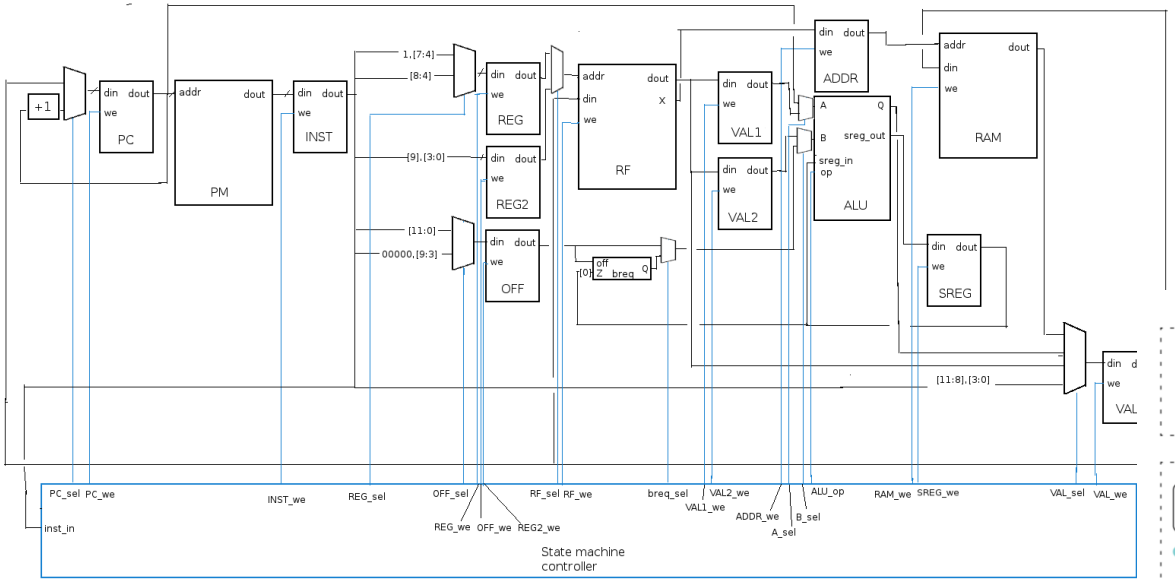
- Given a set of inputs as boolean values and a corresponding set of boolean outputs, one can automatically create a boolean function (circuit block) that will produce that output

State machine controller



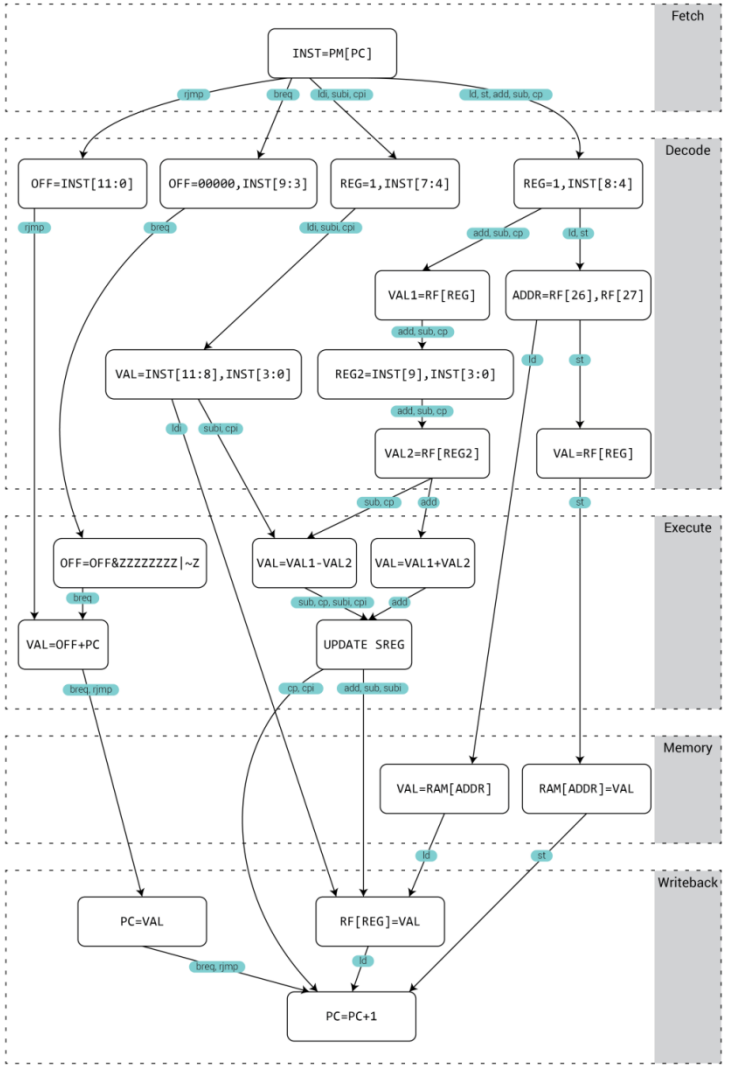
Complete machine

- Complete state machine
- State transition table
- Datapath

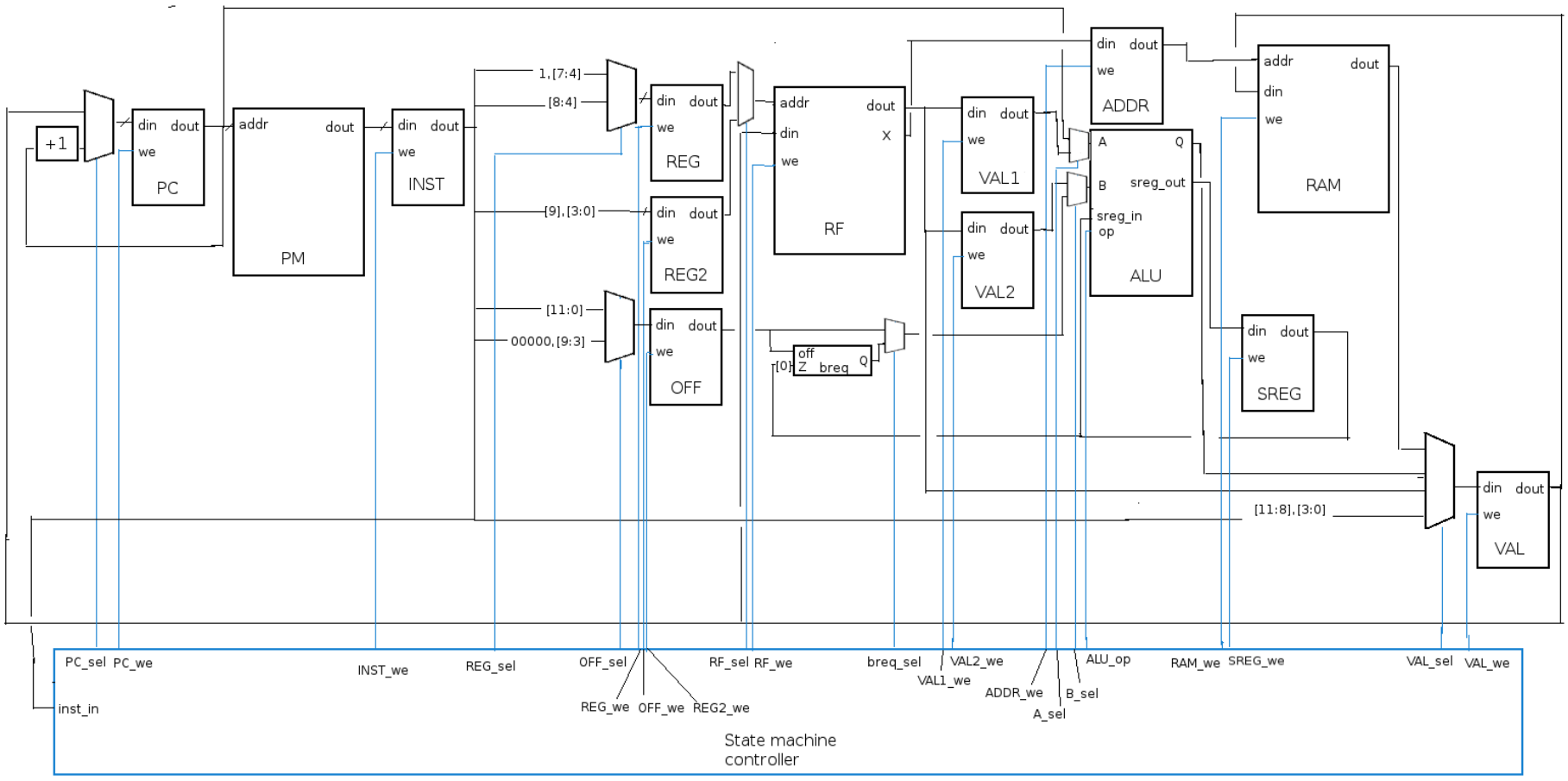


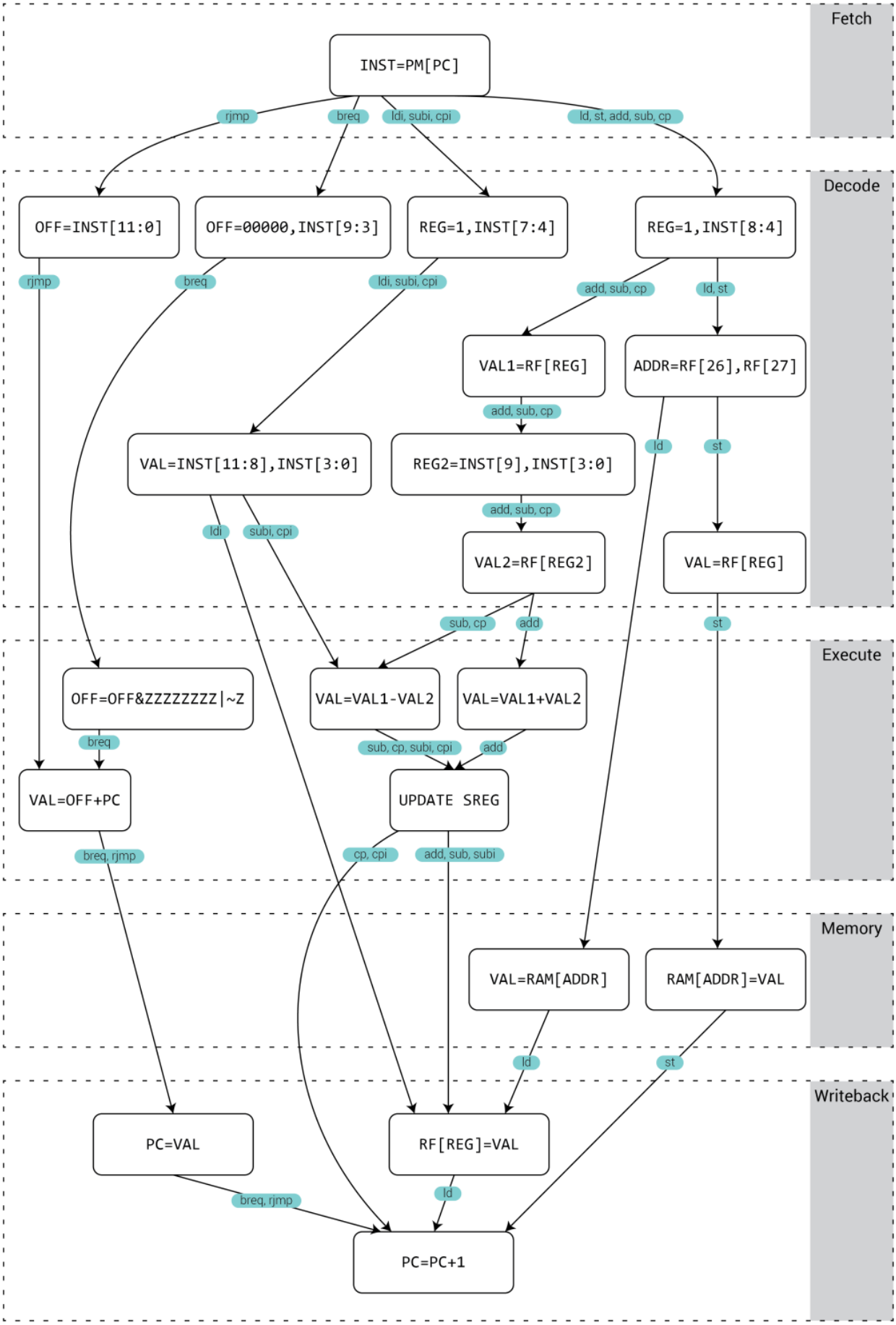
Inst.	Opcode
ldi	1110
ld	1010
mov	001011
add	000011
sub	000110

	PC_sel	PC_we	INST_we	REG_sel	REG_we	OFF_we	OFF_sel	REG2_we	RF_sel	RF_we	breq_sel	VAL1_we	VAL2_we	ADDR_we	A_sel	B_sel	ALU_op	RAM_we	SREG_we	VAL_sel	VA
INST=PM[PC]		0	1		0	0		0		0		0	0	0				0	0		0
REG=1,INST[7:4]		0	0	0	1	0		0		0		0	0	0				0	0		0
REG=INST[8:4]		0	0	1	1	0		0		0		0	0	0				0	0		0
OFF=00000,INST[9:3]		0	0		0	1	1	0		0		0	0	0				0	0		0
OFF=INST[11:0]		0	0		0	1	0	0		0		0	0	0				0	0		0
VAL=INST[11:8],INST[3:0]		0	0		0	0		0		0		0	0	0				0	0	3	1
VAL=RF[REG]		0	0		0	0		0	0	0		0	0	0				0	0	2	1
VAL1=RF[REG]		0	0		0	0		0	0	0		1	0	0				0	0		0
VAL2=RF[REG2]		0	0		0	0		0	1	0		0	1	0				0	0		0
ADDR=X		0	0		0	0		0		0		0	0	1				0	0		0
REG2=INST[9],INST[3:0]		0	0		0	0		1		0		0	0	0				0	0		0
VAL=VAL1+VAL2		0	0		0	0		0		0		0	0	0	0	0	0	0	0	1	1
VAL=VAL1-VAL2		0	0		0	0		0		0		0	0	0	0	0	1	0	0	1	1
Update SREG		0	0		0	0		0		0		0	0	0				0	0		0
VAL=OFF+PC		0	0		0	0		0		0	0	0	0	0	1	1	0	0	0		1
OFF=OFF&ZZZZZZZZ | ~Z		0	0		0	0		0		0	1	0	0	0	1	1	0	0	0		0
VAL=RAM[ADDR]		0	0		0	0		0		0		0	0	0				0	0	0	1
RAM[ADDR] = VAL		0	0		0	0		0		0		0	0	0				0	0		0
RF[REG]=VAL		0	0		0	0		0	0	1		0	0	0				0	0		0
PC=PC+1	0	1	0		0	0		0		0		0	0	0				0	0		0
PC=VAL	1	1	0		0	0		0		0		0	0	0				0	0		0



Its all numbers and we can trace the execution of any instruction and any program!


+1
PC
din dout
we
addr dout
PM
INST
1,[7:4]
[8:4]
[9],[3:0]
[11:0]
00000,[9:3]
REG
REG2
OFF
RF
addr dout
din X
we
off
Z breq Q
[0]
VAL1
VAL2
ADDR
A Q
B sreg_out
sreg_in
op
ALU
RAM
addr dout
din
we
SREG
VAL
[11:8],[3:0]
PC_sel PC_we
inst_in
INST_we
REG_sel
OFF_sel
REG_we OFF_we REG2_we
RF_sel RF_we
breq_sel
VAL2_we
VAL1_we
ADDR_we
A_sel
B_sel
ALU_op
RAM_we SREG_we
VAL_sel
VAL_we
State machine
controller


Fetch
INST=PM[PC]
rjmp
breq
ldi, subi, cpi
ld, st, add, sub, cp
Decode
OFF=INST[11:0]
OFF=00000,INST[9:3]
REG=1,INST[7:4]
REG=1,INST[8:4]
rjmp
breq
ldi, subi, cpi
add, sub, cp
ld, st
VAL1=RF[REG]
ADDR=RF[26],RF[27]
add, sub, cp
VAL=INST[11:8],INST[3:0]
REG2=INST[9],INST[3:0]
ld
st
add, sub, cp
ldi
subi, cpi
VAL2=RF[REG2]
VAL=RF[REG]
sub, cp
add
st
Execute
OFF=OFF&ZZZZZZZZ|~Z
VAL=VAL1-VAL2
VAL=VAL1+VAL2
breq
sub, cp, subi, cpi
add
VAL=OFF+PC
UPDATE SREG
breq, rjmp
cp, cpi
add, sub, subi
Memory
VAL=RAM[ADDR]
RAM[ADDR]=VAL
ld
st
Writeback
PC=VAL
RF[REG]=VAL
breq, rjmp
ld
PC=PC+1

	PC_sel	PC_we	INST_we	REG_sel	REG_we	OFF_we	OFF_sel	REG2_we	RF_sel	RF_we	breq_sel	VAL1_we	VAL2_we	ADDR_we	A_sel	B_sel	ALU_op	RAM_we	SREG_we	VAL_sel	VAL_we
INST=PM[PC]		0	1		0	0		0		0		0	0	0				0	0		0
REG=1,INST[7:4]		0	0	0	1	0		0		0		0	0	0				0	0		0
REG=INST[8:4]		0	0	1	1	0		0		0		0	0	0				0	0		0
OFF=00000,INST[9:3]		0	0		0	1	1	0		0		0	0	0				0	0		0
OFF=INST[11:0]		0	0		0	1	0	0		0		0	0	0				0	0		0
VAL=INST[11:8],INST[3:0]		0	0		0	0		0		0		0	0	0				0	0	3	1
VAL=RF[REG]		0	0		0	0		0	0	0		0	0	0				0	0	2	1
VAL1=RF[REG]		0	0		0	0		0	0	0		1	0	0				0	0		0
VAL2=RF[REG2]		0	0		0	0		0	1	0		0	1	0				0	0		0
ADDR=X		0	0		0	0		0		0		0	0	1				0	0		0
REG2=INST[9],INST[3:0]		0	0		0	0		1		0		0	0	0				0	0		0
VAL=VAL1+VAL2		0	0		0	0		0		0		0	0	0	0	0	0	0	0	1	1
VAL=VAL1-VAL2		0	0		0	0		0		0		0	0	0	0	0	1	0	0	1	1
Update SREG		0	0		0	0		0		0		0	0	0				0	0		0
VAL=OFF+PC		0	0		0	0		0		0	0	0	0	0	1	1	0	0	0		1
OFF=OFF&ZZZZZZZZ | ~Z		0	0		0	0		0		0	1	0	0	0	1	1	0	0	0		0
VAL=RAM[ADDR]		0	0		0	0		0		0		0	0	0				0	0	0	1
RAM[ADDR] = VAL		0	0		0	0		0		0		0	0	0				0	0		0
RF[REG]=VAL		0	0		0	0		0	0	1		0	0	0				0	0		0
PC=PC+1	0	1	0		0	0		0		0		0	0	0				0	0		0
PC=VAL	1	1	0		0	0		0		0		0	0	0				0	0		0

Cycle	PC	INST	REG	REG2	VAL2	VAL2	ADDR	SREG	VAL

Cycle	PC_ sel	PC_ we	INST _we	REG _sel	OFF _Sel	REG _we	OFF _we	REG 2_we	RF_s el	RF_ we	Breq _sel	VAL1 _we	VAL2 _we

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	

Summary

- State machine + datapath == computer
- Tracing of instruction execution

CS 252
Lecture 24; 2015 Nov 9th; Transcribed Lecture notes

Outline

- Review state machine, circuit blocks, and LDI computer
- Tracing through execution of instructions

Review

- Every instruction can be processed as 5 stages
  - fetch
  - decode
  - execute
  - memory
  - write back
- Each stage may be multiple “states”
- Implementation
  - Important to know how stuff are implements down the the transistor and semiconductor level.
  - Control signals select different multiplexers to do different things
  - Control path has 2 pieces: (1) control signals for each state, and (2) some logic to transition to the next state
  - State machine controlling implementation: states generate constants. The select lines of the mux depends on the state. There is next state logic block to implement all possible transitions that needs to be done by current state.
  - Example 1: LDI instruction goes fetch -> decode -> decode -> writeback -> writeback
  - Example 2: LD instruction goes from fetch -> decode -> decode -> memory -> writeback -> writeback
  - Machine can be at exactly one state at a given point of time
  - Next state logic circuit block: opcodes from instructions tell the controller how to transition to the next state.
  - Given current state and opcode transition, we can create a table to transition to the next state.
  - Can create a large table with a set of states, set of opcodes, and set of transition next states. This table will provide everything that the machine needs to implement.
  - 252’s cool axiom for chapter 7: given a set of inputs as boolean values and corresponding set of boolean outputs, one can create a boolean function that will produce that output.
  - complete machine: state machine, state transition table, and datapath needs to be implemented for every instruction.
  - Start with a set of opcodes. Based on the instruction, transition to the next state or perform some sort of computation. Then we can organize all of ours memories

and connect them together. Next, we can generate the corresponding multiplexer select signals for the state transition table.

Once we have the design, we will convince ourself that it can implement every instruction.

It is all numbers that flows through the circuit block to execute instructions.

Trace through instruction to see state transitions

Handout provides the state transition for every instruction and every state.

Our job is to fill up all the register values.

In-class exercise to fill out registers and state transitions.

```
ldi r16, 12
ldi r17, 30
add r16, r17
```

State transitions

Cycle	PC	INST	REG	REG2	VAL1	VAL2	ADDR	SREG	VAL
0	0	undefined	undefined	undefined	undefined	undefined	undefined	undefined	undefined
0	0	ldi r16, 12 0xe00c	undefined	undefined	undefined	undefined	undefined	undefined	undefined
1	0	ldi r16, 12 0xe00c	1,INST[7:4] 0b1000 0 0d16	undefined	undefined	undefined	undefined	undefined	undefined
2	0	ldi r16, 12 0xe00c	16	undefined	undefined	undefined	undefined	undefined	0b00001100 0d12
3	0	ldi r16, 12 0xe00c	16	undefined	undefined	undefined	undefined	undefined	0b00001100 0d12
4	1	ldi r16, 12 0xe00c	16	undefined	undefined	undefined	undefined	undefined	0b00001100 0d12
5	1	ldi r17, 30 0xe11e	16	undefined	undefined	undefined	undefined	undefined	0b00001100 0d12
6	1	ldi r17, 30 0xe11e	1,INST[7:4] 0b1000 1 0d17	undefined	undefined	undefined	undefined	undefined	0b00001100 0d12

At cycle 3, the machine copies 12 to the register 16 in the register file.

Control Signals

PC_sel	PC_we	INST_we	REG_sel	REG_we	...			