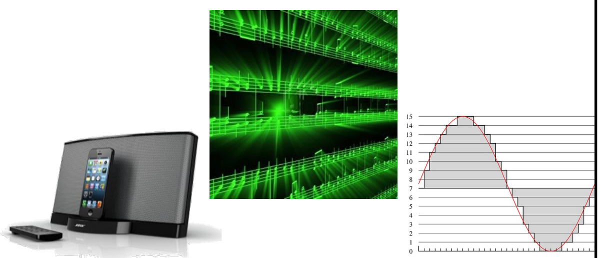
Penn Engineering

ESE



Lecture #15 – Minimal Processor

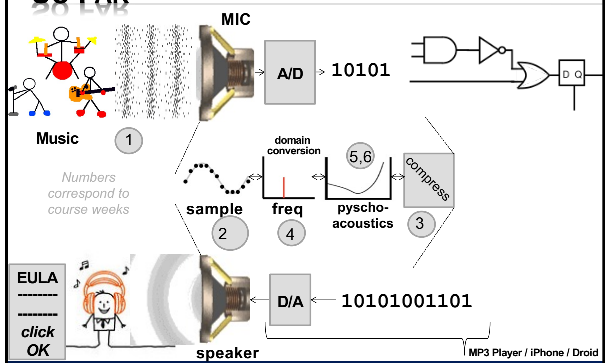
ESE 150 – Digital Audio Basics

ESE150 Spring 2022

Based on slides © 2009--2022 DeHon

1

So Far



2

How Process

- **How do we build a machine to perform these operations?**
  - From Digital Samples → compressed digital data → Digital Samples
- **With simple gates and registers**
  - can build a machine to perform *any* digital computation
  - …**if** we have *enough* of them.

3

Economy and Universality

- **What if we only have a small number of gates?**
- **OR … how many physical gates do we really need?**
  - How do we perform computation with minimal hardware?
- **How do we change the computation performed by our hardware?**

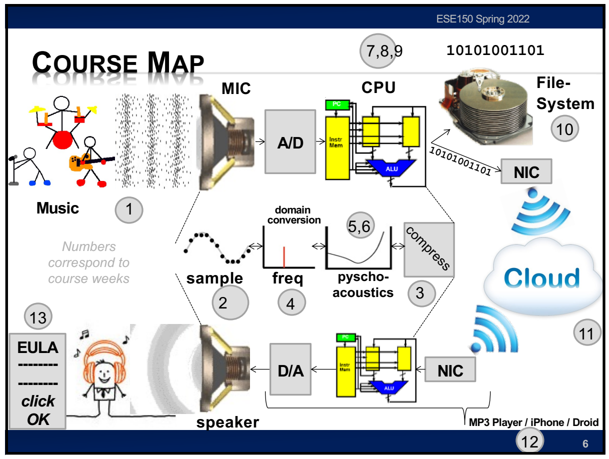
4

Lecture Topics

- **Setup**
- **Where are we?**
- **Review**
- **Memory**
- **One-gate processor**
  - Simplified warm-up of idea and key components
- **Next Lab**

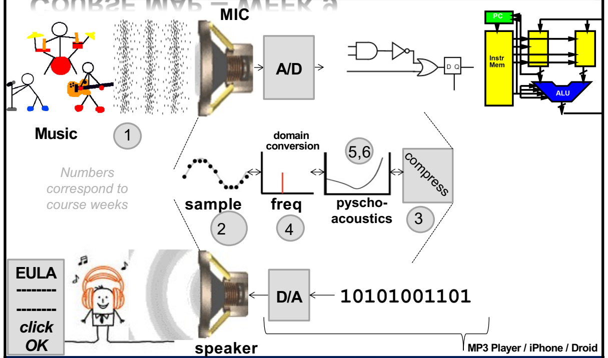
5

Course Map



6

Course Map – Week 9



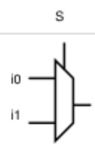
7

Quick Reminder

8

Multiplexer Gate

- **MUX**
  - When S=0, output=i0
  - When S=1, output=i1

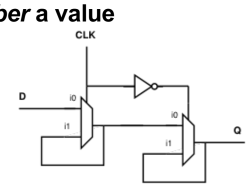


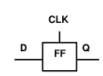
S	i0	i1	Mux2(S,i0,i1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

9

State Element

- **Latch or Register is a state element**
- **Allows circuit to *remember* a value**
- **Build computations that**
  - Depend on past inputs
  - Reuse hardware in time



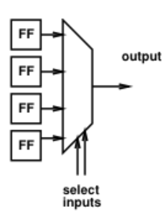


10

Mux can be a Programmable Gate

- **Programmable Gate**
  - Can be programmed to act as any gate
  - Use state (e.g. FF) to "program" truth table of a gate

Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	



11

Universality

- **Can implement any combinational logic function out of a collection**
  - AND, OR, NOT gates
  - Or Programmable MUX gates
    - Can program as AND, OR, NOT…

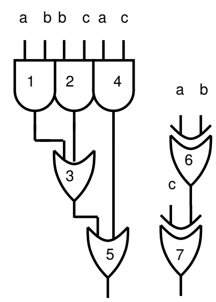
12

Preclass 1

- **What Function?**
  - o1=a&b | b&c | a&c;
  - o2=a^b^c;
- **How many gates?**

13

Preclass 1 in Gates



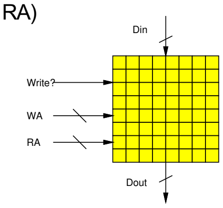
14

Memory

15

Random Access Memory (RAM)

- **A Memory:**
  - Series of locations (slots)
  - Can write values a slot (specified by address, WA)
  - Read values from (by address, RA)
  - Return last value written

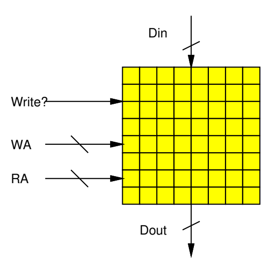


Notation:
slash on wire
means multiple bits wide

16

Key Engineering Property

- **Store state compactly in memory**
- **A(memory cell) small**
  - A(mem) < A(gate)
- **Depends on few inputs/outputs**
  - Memory cells share inputs and outputs



17

One-Gate Processor

18

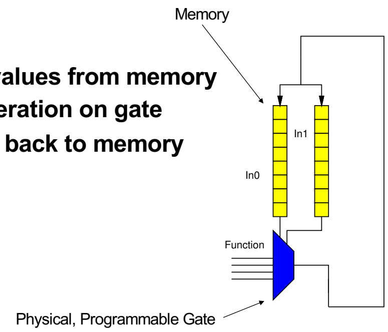
Idea

- Store *logical* (simulated) register and gate outputs in memory
- Compute one gate at a time
  - Using a single ***physical*** gate

Basic Idiom

Repeat:

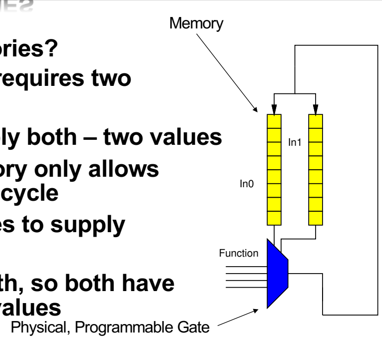
1. **Read gate values from memory**
2. **Perform operation on gate**
3. **Write result back to memory**



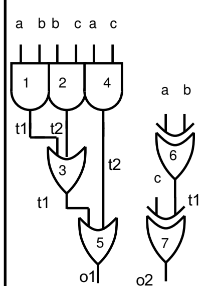
Two Memories

Why two memories?

- **2-input gate requires two binary inputs**
- **Want to supply both – two values**
- **Simple memory only allows one read per cycle**
- **Two memories to supply two values**
- **Write into both, so both have same set of values**

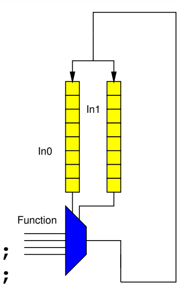


Operation



0	1	2	3	4	5	6	7
a	b	c	t1	t2	o1	o2	

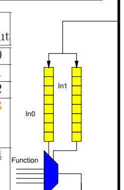
```
a=getInput(0);
b=getInput(1);
c=getInput(2);
t1=a&b;    // 1
t2=b&c;    // 2
t1=t1|t2;  // 3
t2=a&c;    // 4
o1=t1|t2;  // 5
t1=a^b;    // 6
o2=t1^c;   // 7
putOutput(1,o2);
putOutput(0,o1);
```



Operation Sequence

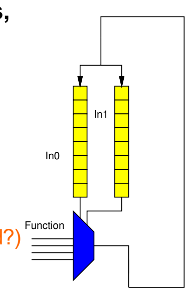
0	1	2	3	4	5	6	7
a	b	c	t1	t2	o1	o2	

		Instruction Fields				
C	Description	Type	Function	In0	In1	Out
a=getInput(0);	read input 0 and put in slot 0	READ	NONE	0	0	0
b=getInput(1);	read input 1 and put in slot 1	READ	NONE	1	0	1
c=getInput(2);	read input 2 and put in slot 2	READ	NONE	2	0	2
`t1=a&b;`	read value in slot 0 and value in slot 1, perform an AND on the values, and store into slot 3	GATE	AND	0	1	3
Missing C step?	read value in slot 1 and value in slot 2, perform an AND on the values, and store into slot 4	GATE	AND	1	2	4
`t1=t1|t2;`	read value in slot 3 and value in slot 4, perform an OR on the values, and store into slot 3	GATE	OR	3	4	3
`t2=a&c;`	Missing description?	GATE	AND	0	2	4



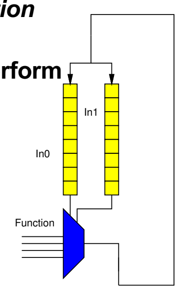
Observe

- **We can sequentialize operations, reusing the single gate**
- **As long as we can specify the operation to be performed**
- **What are we specifying?**
  - (break it down, what information need?)



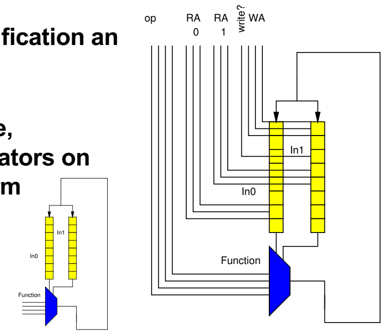
Instruction

- Call this specification an ***instruction***
- Instructs the programmable, reusable operators on what to perform



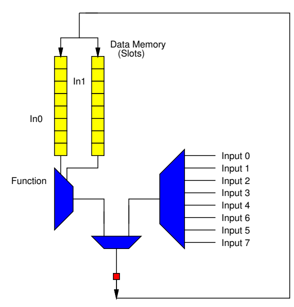
Instruction

- Call this specification an ***instruction***
- Instructs the programmable, reusable operators on what to perform



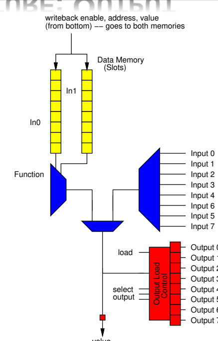
Expanding the Structure: Input

- Add a multiplexer to bring in inputs
- Allow as option to write into data memory



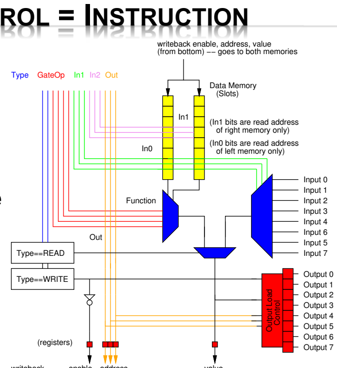
Expanding the Structure: Output

- Add way to load a designated output register



Expanded Control = Instruction

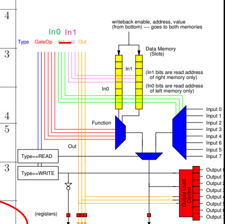
- Group the full control into instruction
- Set of bits that tells the structure what to do



Fillin missing Instruction

C	Description	Instruction Fields Type	Function	In0	In1	Out
a=getInput(0);	read input 0 and put in slot 0	READ	NONE	0	0	0
b=getInput(1);	read input 1 and put in slot 1	READ	NONE	1	0	1
c=getInput(2);	read input 2 and put in slot 2	READ	NONE	2	0	2
`t1=a&b;`	read value in slot 0 and value in slot 1, perform an AND on the values, and store into slot 3	GATE	AND	0	1	3
	read value in slot 1 and value in slot 2, perform an AND on the values, and store into slot 4	GATE	AND	1	2	4
`t1=t1|t2;`	read value in slot 3 and value in slot 4, perform an OR on the values, and store into slot 3	GATE	OR	3	4	3
`t2=a&c;`		GATE	AND	0	2	4
`o1=t1|t2;`	read value in slot 3 and value in slot 4, perform an OR on the values, and store into slot 5	GATE	OR	3	4	5
`t1=a^b;`	read value in slot 0 and value in slot 1, perform an XOR on the values, and store into slot 3	GATE	XOR	0	1	3
`o2=t1^c;`	read value in slot 3 and value in slot 2, perform an XOR on the values, and store into slot 6					

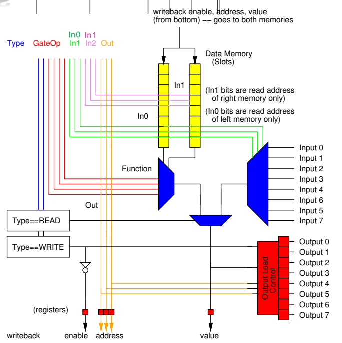
0	1	2	3	4	5	6	7
a	b	c	t1	t2	o1	o2	



Instruction Bits

GATE	AND	0	1	2	010001000001010

- Instructions are just a set of bits
- Type – 2 bits
- GateOp – 4 bits
- In1 – 3 bits
  - Assume 8 slots
- In2 – 3 bits
- Out – 3 bits



31

Instruction Bits Example

- Fillin Missing

GATE	AND	0	1	2	010001000001010

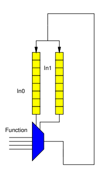
READ=00; GATE=01; WRITE=11;
AND=0001; OR=0111; XOR=0110; NONE=0000; SEL0=0101

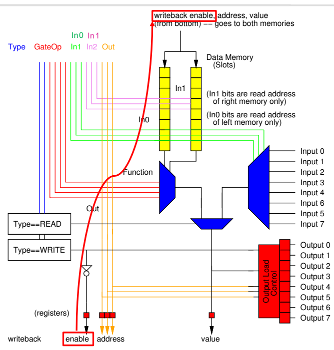
t1=t1|t2;	read value in slot 3 and value in slot 4, perform an OR on the values, and store into slot 3	GATE	OR	3	4	3	010111011100011
t2=a&c;		GATE	AND	0	2	4	010001000010100
o1=t1|t2;	read value in slot 3 and value in slot 4, perform an OR on the values, and store into slot 5	GATE	OR	3	4	5	

32

Note Write

- Note write enable
  - Fed back for next cycle
  - Also address, value

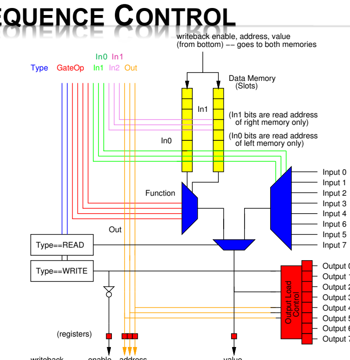




33

Instruction Sequence Control

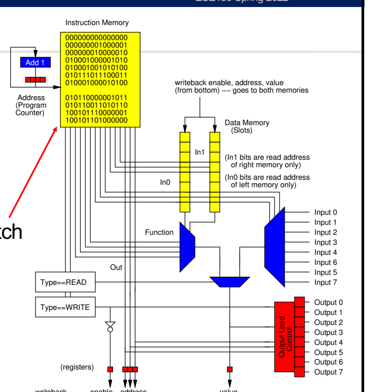
- How provide the sequence of instructions?



34

Instruction Memory

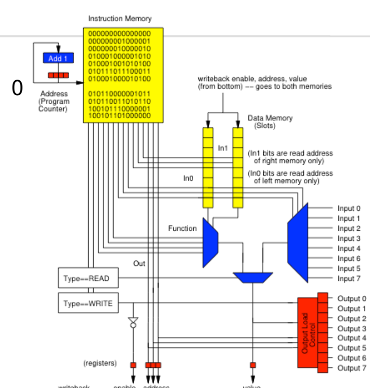
- Add Memory to hold set of Instructions
  - Note contents match table on p. 2 of preclass
- Counter to sequence instructions



35

Animate

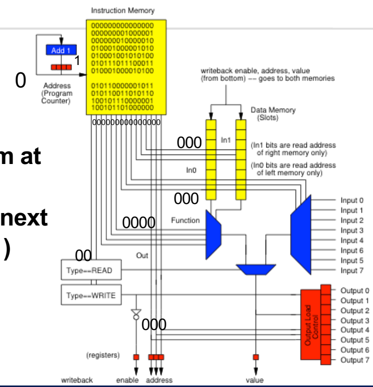
- Start at PC=0



36

ANIMATE

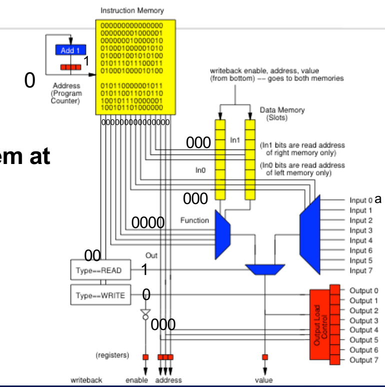
- Start at PC=0
- Read Instr. Mem at 0
- (also compute next PC by adding 1)



37

ANIMATE

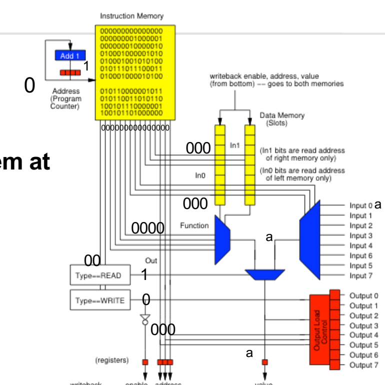
- Start at PC=0
- Read Instr. Mem at 0
- Decode



38

ANIMATE

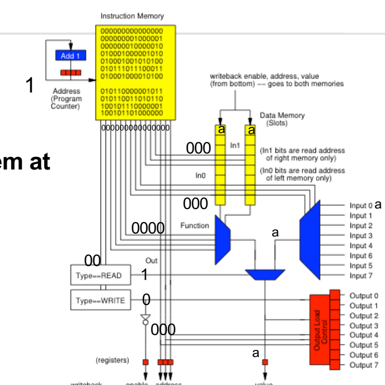
- Start at PC=0
- Read Instr. Mem at 0
- Decode
- From input



39

ANIMATE

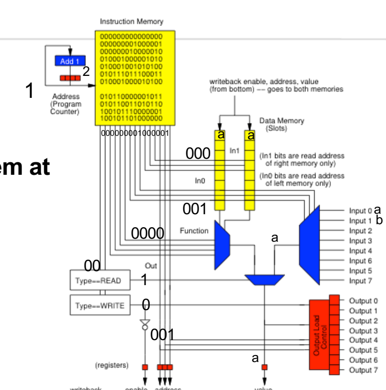
- Start at PC=0
- Read Instr. Mem at 0
- Decode
- From input
- Write Back
- Update PC



40

ANIMATE

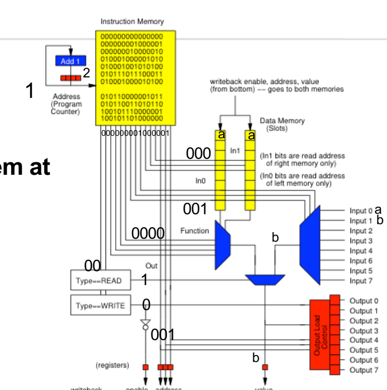
- PC=1
- Read Instr. Mem at 1



41

ANIMATE

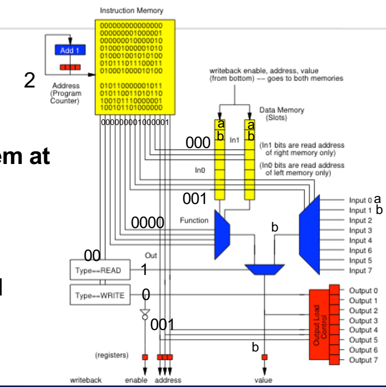
- PC=1
- Read Instr. Mem at 1
- Decode
- From Input



42

Animate

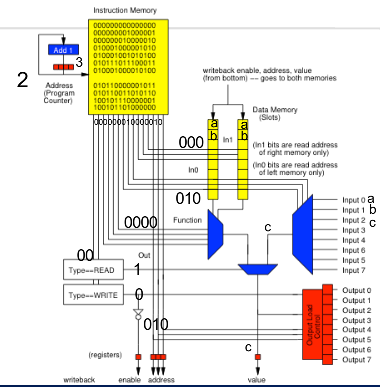
- PC=1
- Read Instr. Mem at 1
- Decode
- From Input
- Writeback and update PC



43

Animate

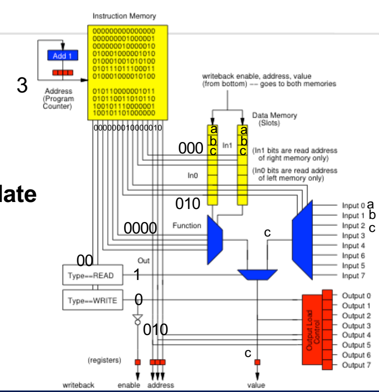
- PC=2
- Another read



44

Animate

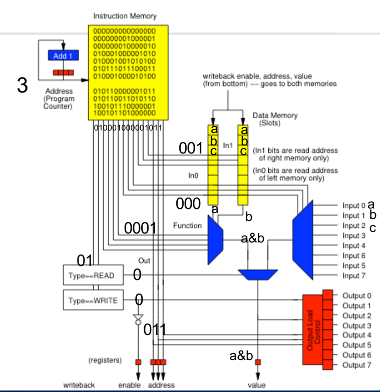
- PC=2
- Another read
- Writeback, update PC



45

Animate

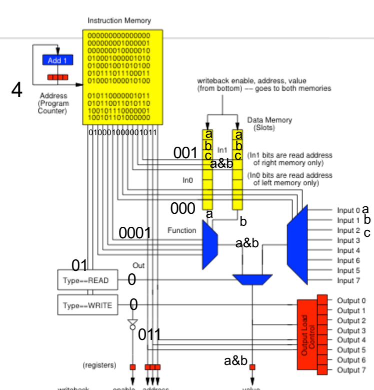
- PC=3



46

Animate

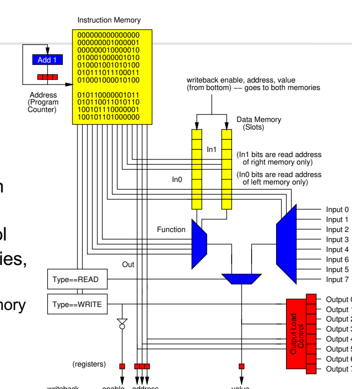
- PC=3
- Writeback and update PC



47

Processor

- Continue this sequence
  - Given PC
  - Read from Instruction Memory
  - Instruction bits control the datapath (memories, function, muxes)
    - Read from data memory
    - Perform operation
  - Write results back to memory; update PC

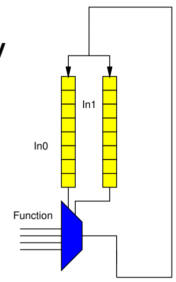


48

Basic Idiom

Repeat:

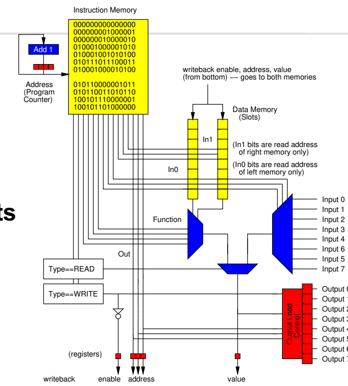
1. Read gate values from memory
2. Perform operation on gate
3. Write result back to memory



49

Universal Processor

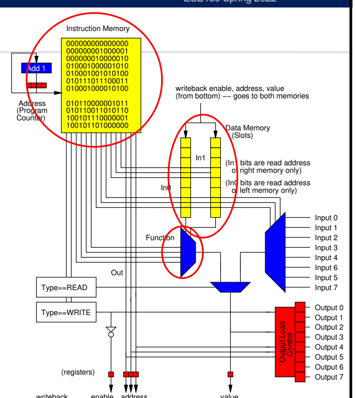
- Can change computation simply be changing contents of instruction memory



50

Review

- Single active compute element (programmable gate)
- Sequence in time
- Store state in memory
- Use Instruction memory to select and sequence operations
- Can compute a large number of gates



51

Next Lab

- Look at Instruction-Level code for ARM
- Understand performance from instruction-level code

52

Big Ideas

- Can implement large computations on small hardware by reusing hardware in time
  - Storing computational state in memory
- Can store program control in instruction memory
  - Change program by reprogramming memory
  - Universal machine: Stored-Program Processor

53

Learn More

- CIS240 – processor organization and assembly
- CIS471 – implement and optimize processors
  - Including FPGA mapping in Verilog
- ESE370 – implement memories (and gates) using transistors

54

Reminders

- Feedback
- Lab 7 today

55