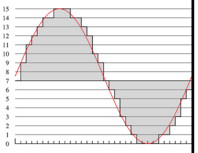
Penn Engineering ESE



Lecture #16 – Stored-Program Processors

ESE 150 – Digital Audio Basics

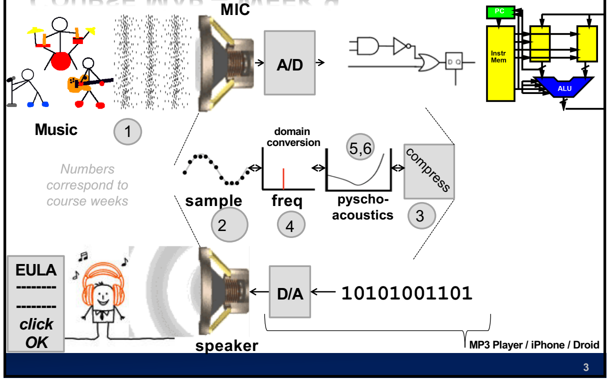
ESE150 Spring 2021

Based on slides © 2009--2021 DeHon

Lecture Topics

- Setup
- Where are we?
- Review
- Memory
- Wide-Word, Stored-Program Processor
- Contemporary Processor: ARM

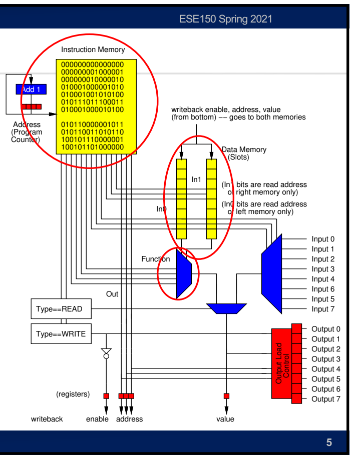
Course Map – Week 9



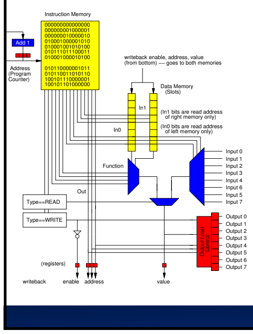
Quick Reminder

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



Preclass 1

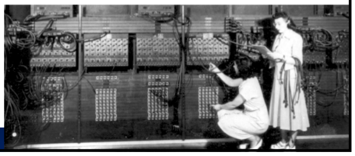


A	T	F	i1	i2	o	0	1	2	3	4	5	6	7
4	G	&	1	2	4	1	1	0	1	0	0	0	0
5	G	|	3	4	3								
6	G	&	0	2	4								
7	G	|	3	4	5								
8													

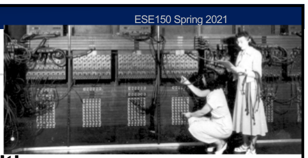
Stored-Program Processor

"Stored Program" Computer

- Can build physical machines that perform *any* computation.
- Can be built with limited hardware that is reused in time.
- Historically: this was a key contribution of Penn's Moore School
  - ENIAC→ EDVAC
  - Computer Engineers: Eckert and Mauchly
  - (often credited to Von Neumann)



Basic Idea

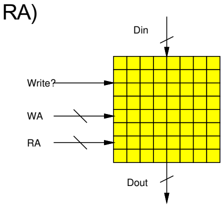


- Express computation in terms of a few primitives
  - E.g. Add, Multiply, OR, AND, NAND
- Provide one of each hardware primitive
- Store intermediates in memory
- Sequence operations on hardware to perform larger computation
- Store *description* of operation sequence in memory as well – hence "Stored Program"
- By filling in memory, can program to perform any computation

Memory

Random Access Memory

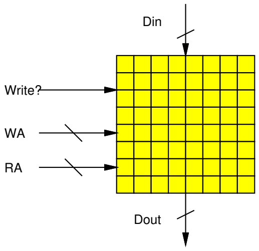
- A Memory:
  - Series of locations (slots)
  - Can write values a slot (specified by address, WA)
  - Read values from (by address, RA)
  - Return last value written



Notation:
slash on wire
means multiple bits wide

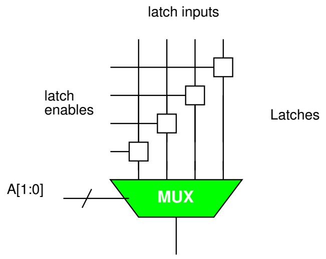
Two Pieces of a Memory

1. Element to remember a value
2. Way to address/select that element



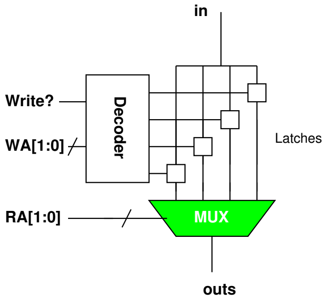
Could build Memory w/ Muxes & latches
… collection of registers

- Use latch to remember (store) values
- Perform read select (addressing) with a multiplexer



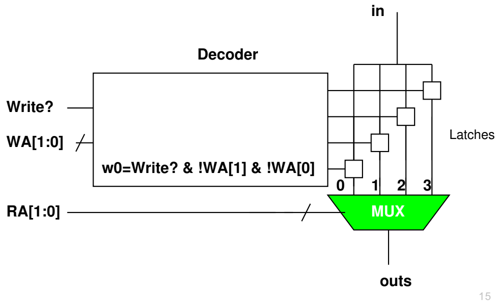
Could build Memory w/ Muxes & latches
… collection of registers

- Perform write select (addressing) with a decoder



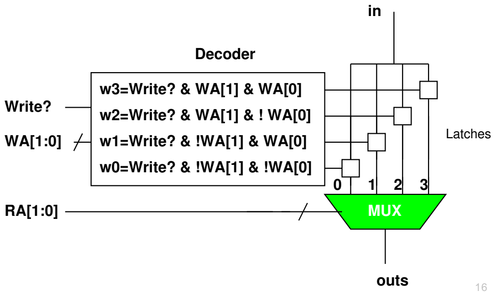
Could build Memory w/ Muxes & latches
… collection of registers

- Show Decoder logic

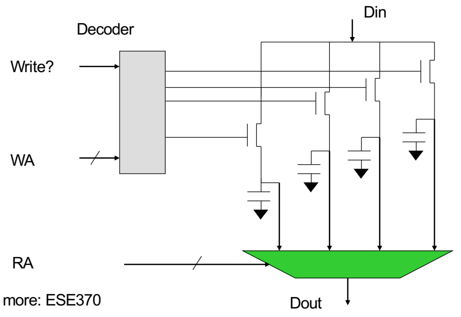


Could build Memory w/ Muxes & latches
… collection of registers

- Show Decoder logic


in
Decoder
w3=Write? & WA[1] & WA[0]
w2=Write? & WA[1] & ! WA[0]
w1=Write? & !WA[1] & WA[0]
w0=Write? & !WA[1] & !WA[0]
Write?
WA[1:0]
Latches
0
1
2
3
RA[1:0]
MUX
outs

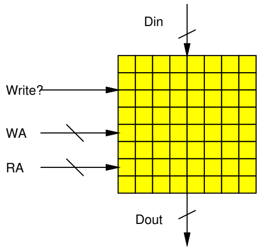
Random Access Memory (RAM) with Capacitor Memories



Learn more: ESE370

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



ESE150 Spring 2021

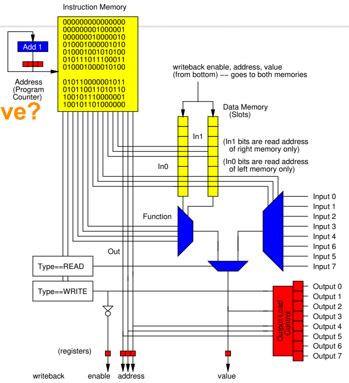
Expand Processor

19

ESE150 Spring 2021

Building Out

- How limited?
- How might improve?



20

ESE150 Spring 2021

Processors

21

ESE150 Spring 2021

Beyond Single Gate

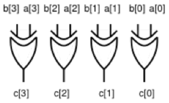
- **Single gate extreme to make the high-level point**
  - Except in some particular cases, not practical
- **Usually reuse larger blocks**
  - Multi-bit Adders
  - Multipliers
- **Get more done per cycle than one gate**
- **Now it's a matter of engineering the design point**
  - Where do we want to be between one gate and full circuit extreme?
  - How many gate evaluations should we physically compute each cycle?

22

ESE150 Spring 2021

Word-Wide Processors

- **Common to compute on multibit words**
  - Add two 16b numbers
  - Multiply two 16b numbers
  - Perform bitwise-XOR on two 32b numbers
- **More hardware**
  - 16 full adders, 32 XOR gates

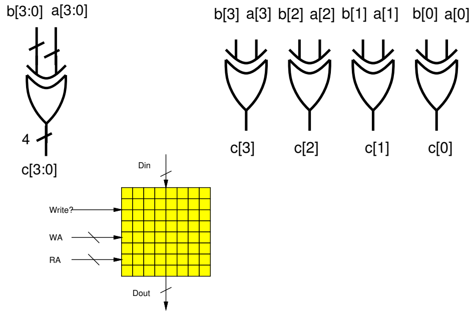


- **All programmable gates doing the *same* thing**
  - So don't require more instruction bits

23

ESE150 Spring 2021

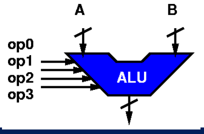
Multibit Bus Symbols



24

Arithmetic and Logic Unit (ALU)

- **A common logic primitive is the ALU**
  - Can perform any of a number of operations on a series of words (strings of bits)
  - **Operations:** Add, subtract, shift-left, shift-right, bitswise xor, and, or, invert, ….
  - Operates on "words"
- **Identify a set of control bits that select the operation it forms**
  - Makes it "programmable"



ALU Ops (on 8bit words)

- **ADD 00011000 00010100 =**
  - Add 0x18 to 0x14 result is:
  - Add 24 to 20

ALU Ops (on 8bit words)

- **ADD 00011000 00010100 = 00101100**
  - Add 0x18 to 0x14 =0x2C0
  - Add 24 to 20 =44
- **SUB 00011000 00010100 = 00000100**
  - Subtract 0x14 from 0x18 .. 0x04
- **INV 00011000 XXXXXXXX =**
  - Invert the bits in 0x18 ...gives us:

ALU Ops (on 8bit words)

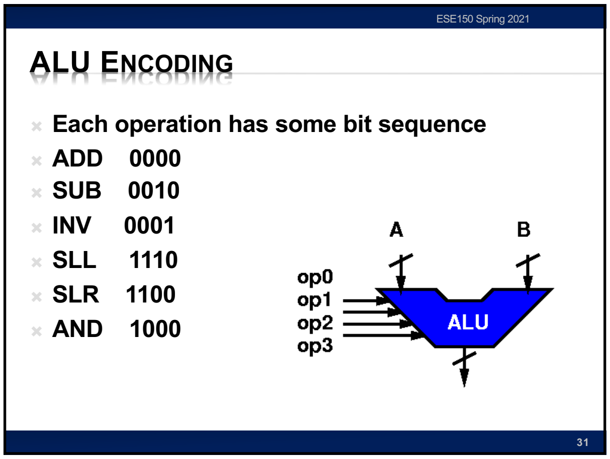
- **XOR 00011000 00010100 = 0001100**
  - xor 0x18 to 0x14 = 0x0C
- **ADD 00011000 00010100 = 00101100**
  - Add 0x18 to 0x14 =0x2C0
  - Add 24 to 20 =44
- **SUB 00011000 00010100 = 00000100**
  - Subtract 0x14 from 0x18 .. 0x04
- **INV 00011000 XXXXXXXX = 11100111**
  - Invert the bits in 0x18 …0xD7
- **SRL 00011000 XXXXXXXX =**
  - Shift right 0x18 … gives us:

ALU Ops (on 8bit words)

- **ADD 00011000 00010100 = 00101100**
  - Add 0x18 to 0x14 =0x2C0
  - Add 24 to 20 =44
- **SUB 00011000 00010100 = 00000100**
  - Subtract 0x14 from 0x18 .. 0x04
- **INV 00011000 XXXXXXXX = 11100111**
  - Invert the bits in 0x18 …0xD7
- **SRL 00011000 XXXXXXXX = 00001100**
  - Shift right 0x18 …0x0C

ALU Ops (on 8bit words)

- **ADD 00011000 00010100 = 00101100**
  - Add 0x18 to 0x14 =0x2C0
  - Add 24 to 20 =44
- **SUB 00011000 00010100 = 00000100**
  - Subtract 0x14 from 0x18 .. 0x04
- **INV 00011000 XXXXXXXX = 11100111**
  - Invert the bits in 0x18 …0xD7
- **SRL 00011000 XXXXXXXX = 00001100**
  - Shift right 0x18 …0x0C
- **XOR 00011000 00010100 = 0001100**
  - xor 0x18 to 0x14 = 0x0C

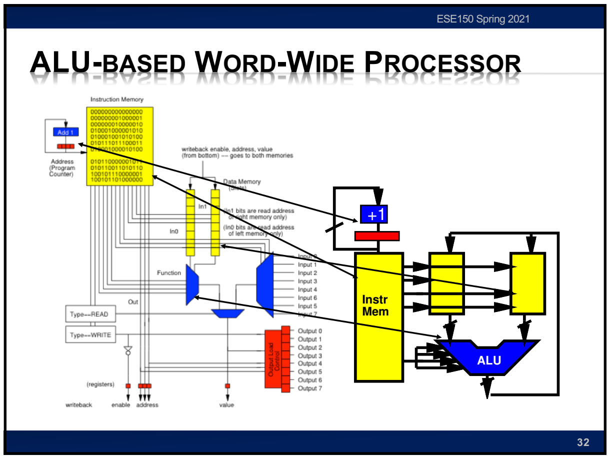

ESE150 Spring 2021

ALU Encoding

- Each operation has some bit sequence
- ADD 0000
- SUB 0010
- INV 0001
- SLL 1110
- SLR 1100
- AND 1000

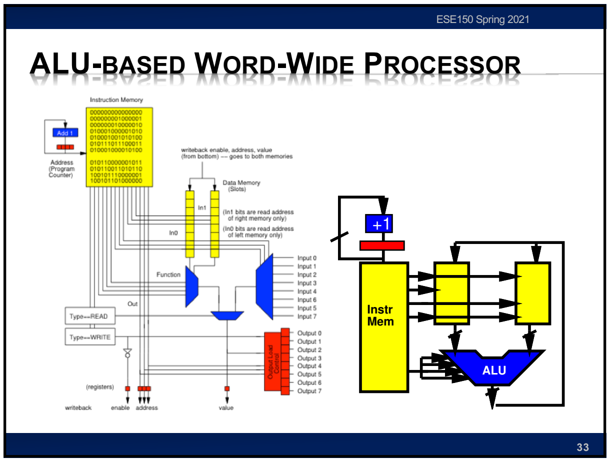
A
B
op0
op1
op2
op3
ALU

31


ESE150 Spring 2021

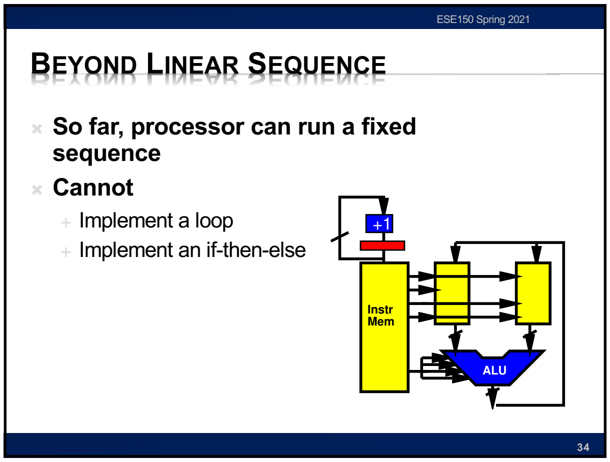
ALU-based Word-Wide Processor

32


ESE150 Spring 2021

ALU-based Word-Wide Processor

33

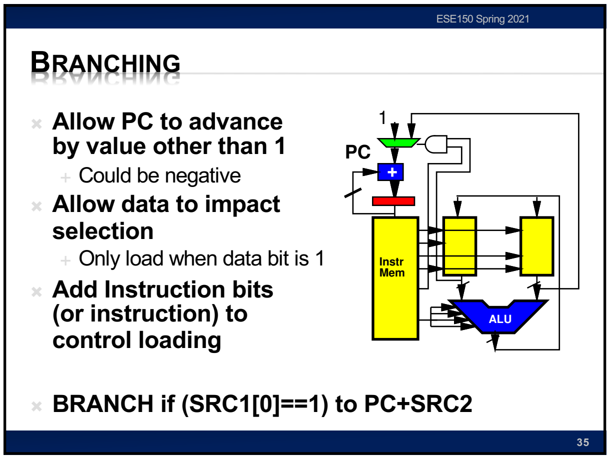

ESE150 Spring 2021

Beyond Linear Sequence

- So far, processor can run a fixed sequence
- Cannot
  - Implement a loop
  - Implement an if-then-else

+1
Instr Mem
ALU

34

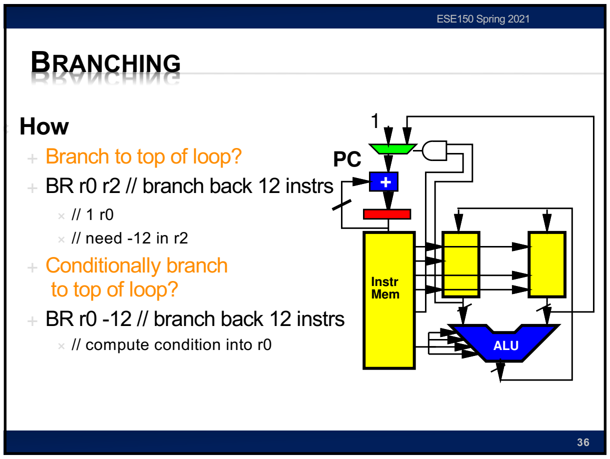

ESE150 Spring 2021

Branching

- Allow PC to advance by value other than 1
  - Could be negative
- Allow data to impact selection
  - Only load when data bit is 1
- Add Instruction bits (or instruction) to control loading
- BRANCH if (SRC1[0]==1) to PC+SRC2

1
PC
+
Instr Mem
ALU

35

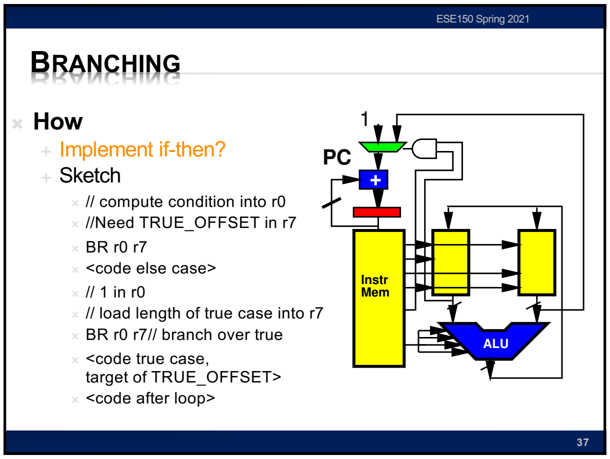

ESE150 Spring 2021

Branching

How
- Branch to top of loop?
- BR r0 r2 // branch back 12 instrs
  - // 1 r0
  - // need -12 in r2
- Conditionally branch to top of loop?
- BR r0 -12 // branch back 12 instrs
  - // compute condition into r0

1
PC
+
Instr Mem
ALU

36

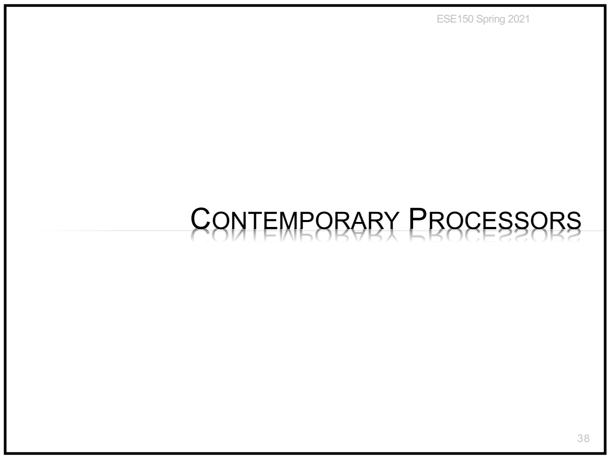


ESE150 Spring 2021

Branching

- How
  - Implement if-then?
  - Sketch
    - // compute condition into r0
    - //Need TRUE_OFFSET in r7
    - BR r0 r7
    - <code else case>
    - // 1 in r0
    - // load length of true case into r7
    - BR r0 r7// branch over true
    - <code true case, target of TRUE_OFFSET>
    - <code after loop>

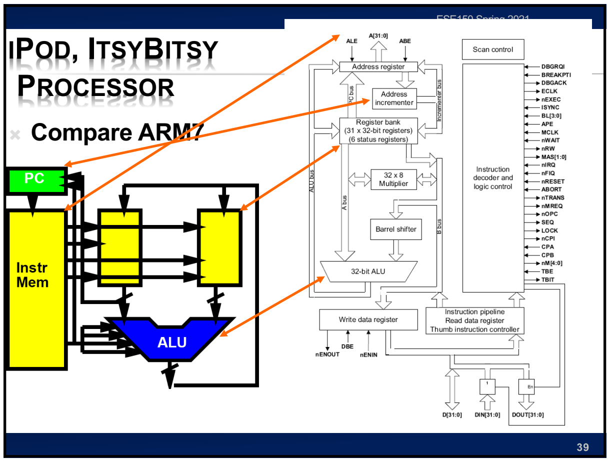
37



ESE150 Spring 2021

Contemporary Processors

38

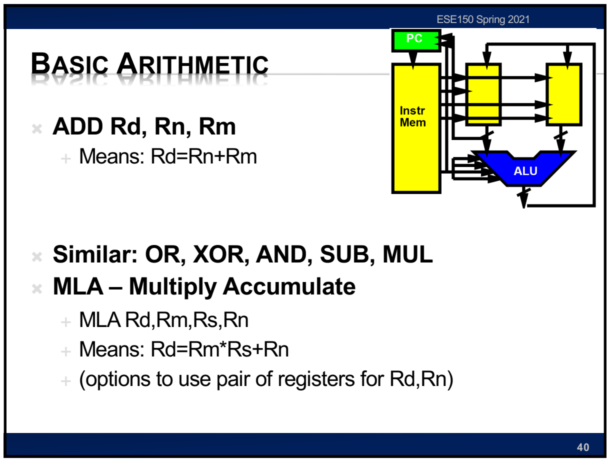


ESE150 Spring 2021

iPod, ItsyBitsy Processor

- Compare ARM7

39

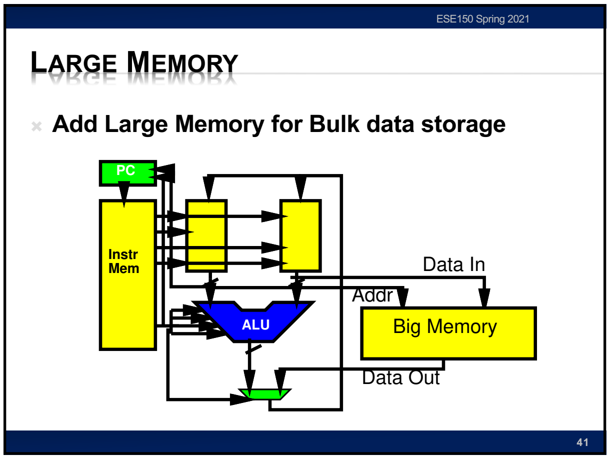


ESE150 Spring 2021

Basic Arithmetic

- ADD Rd, Rn, Rm
  - Means: Rd=Rn+Rm
- Similar: OR, XOR, AND, SUB, MUL
- MLA – Multiply Accumulate
  - MLA Rd,Rm,Rs,Rn
  - Means: Rd=Rm*Rs+Rn
  - (options to use pair of registers for Rd,Rn)

40

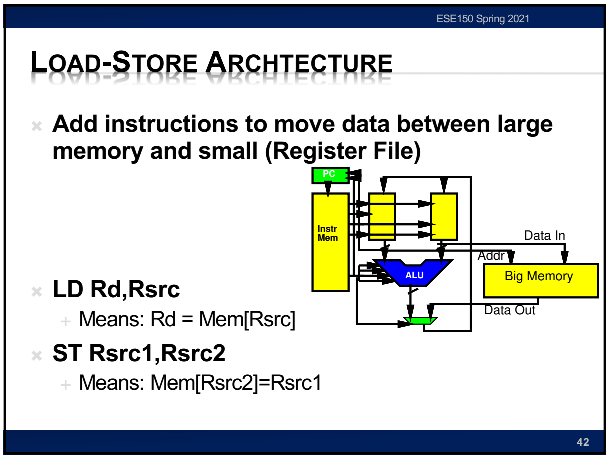


ESE150 Spring 2021

Large Memory

- Add Large Memory for Bulk data storage

41

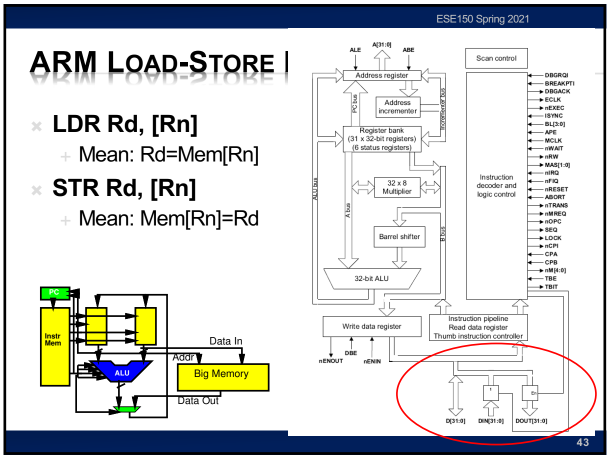


ESE150 Spring 2021

Load-Store Archtecture

- Add instructions to move data between large memory and small (Register File)
- LD Rd,Rsrc
  - Means: Rd = Mem[Rsrc]
- ST Rsrc1,Rsrc2
  - Means: Mem[Rsrc2]=Rsrc1

42

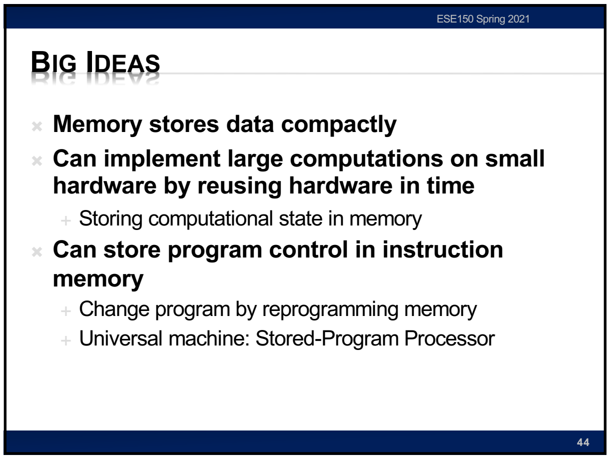


ESE150 Spring 2021

ARM Load-Store |

- LDR Rd, [Rn]
  - Mean: Rd=Mem[Rn]
- STR Rd, [Rn]
  - Mean: Mem[Rn]=Rd

43

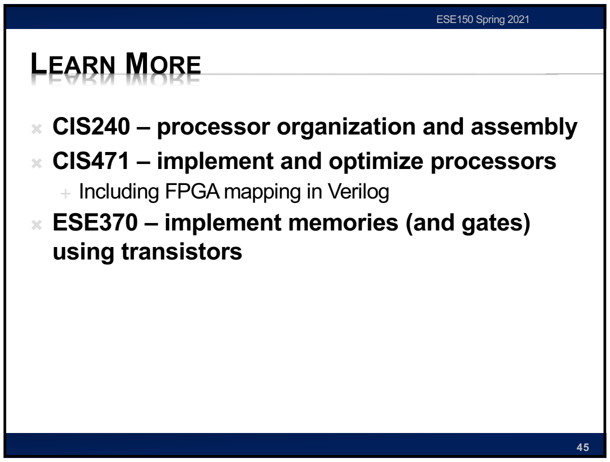


ESE150 Spring 2021

Big Ideas

- Memory stores data compactly
- 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

44

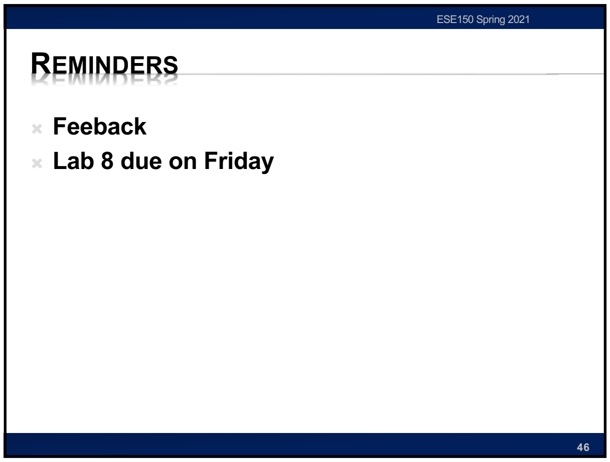


ESE150 Spring 2021

Learn More

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

45



ESE150 Spring 2021

Reminders

- Feeback
- Lab 8 due on Friday

46