

Lecture #16 – Stored-Program Processors

**ESE 150 – DIGITAL AUDIO BASICS**

ESE150 Spring 2022

Based on slides © 2009–2022 DeHon

1

ESE150 Spring 2022

### LECTURE TOPICS

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

2

ESE150 Spring 2022

### COURSE MAP – WEEK 9

Music (1) → sample (2) → freq (4) → psycho-acoustics (3) → compress → D/A → speaker

10101001101

MP3 Player / iPhone / Droid

3

ESE150 Spring 2022

### QUICK REMINDER

4

ESE150 Spring 2022

### REVIEW

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

5

ESE150 Spring 2022

### PRECLASS 1

	A	T	F	10	11	0	1	2	3	4	5	6	7
3	G	&	0	1	3	1	1	0	0	0	0	0	0
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													

6

ESE150 Spring 2022

# STORED-PROGRAM PROCESSOR

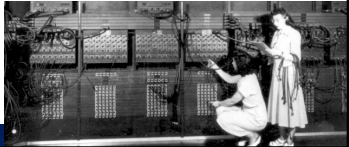
7

7

ESE150 Spring 2022

## “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)

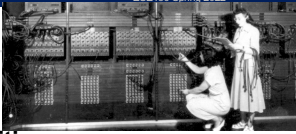


8

8

ESE150 Spring 2022

## 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

9

9

ESE150 Spring 2022

# MEMORY

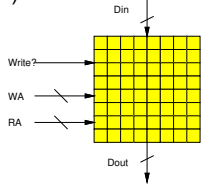
10

10

ESE150 Spring 2022

## 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

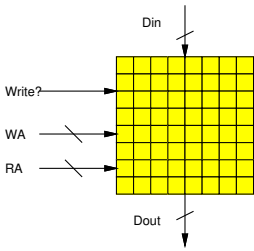
11

11

ESE150 Spring 2022

## KEY ENGINEERING PROPERTY

- ✗ Store state compactly in memory
- ✗ **A(memory cell) small**
  - +  $A(\text{mem}) < A(\text{gate})$
- ✗ **Depends on few inputs/outputs**
  - + Memory cells share inputs and outputs



18

18

ESE150 Spring 2022

Part 2

# EXPAND PROCESSOR

19

19

ESE150 Spring 2022

## BUILDING OUT

- ✗ **Deliberately simple**
  - + Single gate
  - + Lacks many things expect from processors --- that need to run C code
  - + ...or Java, Python...

The diagram shows a processor with an 'Add1' block and a 'Function' block. It has 'Instruction Memory' and 'Data Memory (80b)'. An 'Address (Program Counter)' is shown with a list of binary addresses: 0000000000000000, 0000000100000001, 0000000100000010, 010001000001010, 010001001010100, 010111011100011, 01000100001100. Below this, a 'writeback enable, address, value' block is shown with binary values: 010110000001011, 010110011010110, 100101110000001, 001001000000000. The processor has 8 inputs (Input 0-7) and 8 outputs (Output 0-7). It also has 'Type=READ' and 'Type=WRITE' signals, and 'writeback enable', 'address', and 'value' signals. A 'Carry Out' is also shown.

20

20

ESE150 Spring 2022

# PROCESSORS

21

21

ESE150 Spring 2022

## 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

22

ESE150 Spring 2022

## 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

The diagram shows four full adders, each with two 3-bit inputs (a[3:0] and b[3:0]) and a carry-in (c[3:0]). The outputs are 4-bit carry outputs (c[3], c[2], c[1], c[0]) and 4-bit data outputs (a[3:0] and b[3:0]).

23

23

ESE150 Spring 2022

## MULTIBIT BUS SYMBOLS

The diagram shows multibit bus symbols. On the left, a 4-bit adder symbol with inputs b[3:0] and a[3:0], and output c[3:0]. On the right, four 2-bit multiplier symbols with inputs a[3]b[2], a[2]b[1], a[1]b[0], and b[0]a[0], and outputs c[3], c[2], c[1], and c[0]. Below these is a 4x4 grid representing a multiplier array with inputs Write?, WA, and RA, and outputs Din and Dout.

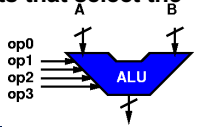
24

24

ESE150 Spring 2022

## 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, bitwise xor, and, or, invert, ....
  - + Operates on "words" –fixed number of bits – e.g. 16
    - × Can interpret as number or address
- × **Identify a set of control bits that select the operation it forms**
  - + Makes it "programmable"



25

25

ESE150 Spring 2022

## ALU OPS (ON 8BIT WORDS)

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

26

26

ESE150 Spring 2022

## 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:

27

27

ESE150 Spring 2022

## 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

28

28

ESE150 Spring 2022

## 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 = 00001100**
  - × xor 0x18 to 0x14 = 0x0C

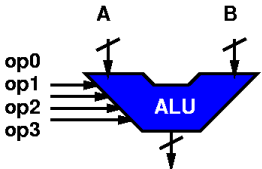
29

29

ESE150 Spring 2022

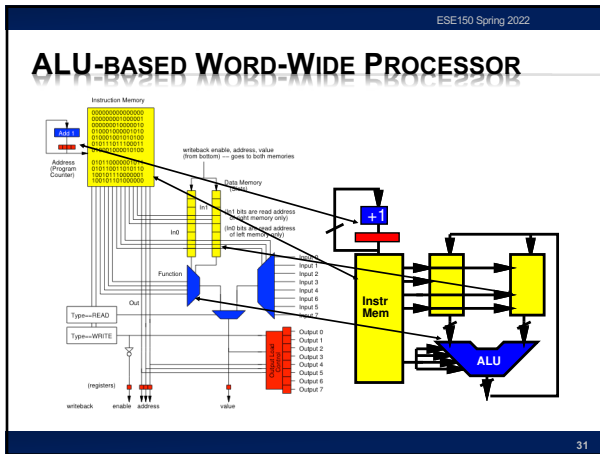
## ALU ENCODING

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

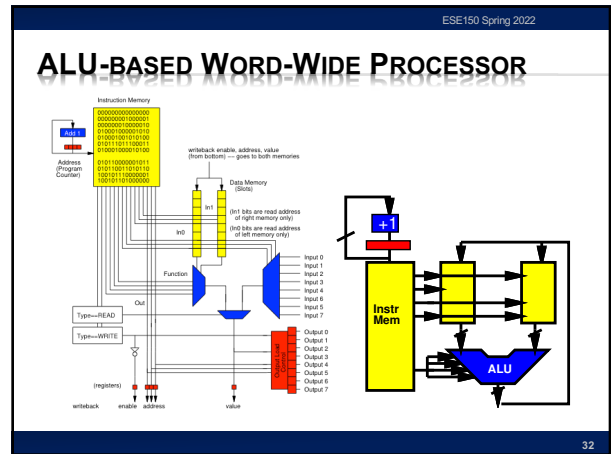


30

30



31



32

### BEYOND LINEAR SEQUENCE

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

33

33

### BRANCHING

- ✗ Allow PC to advance by value other than 1
  - + Could be negative

34

34

### 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

35

35

### 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

36

36

ESE150 Spring 2022

## 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

37

37

ESE150 Spring 2022

## BRANCHING

Given instruction: BR SRC1 SRC2  
BRANCH if (SRC1[0]==1) to PC+SRC2

How

- + Branch to top of loop?
  - × Say loop is instructions 5—15
- + To cause control to loop back to instruction 5?
  - × What should instruction 16 be?
  - × What values need to be in SRC1, SRC2 for this to work?

38

38

ESE150 Spring 2022

## BRANCHING

Given instruction: BR SRC1 SRC2  
BRANCH if (SRC1[0]==1) to PC+SRC2

How

- + Branch to top of loop?
  - × Say loop is instructions 5—15
- + To cause control to loop back to instruction 5?
  - × What should instruction 16 be?
  - × What values need to be in SRC1, SRC2 for this to work?
  - × BR R0 R1
  - × // R0 hold 1, R1 hold -11

39

39

ESE150 Spring 2022

## BRANCHING

How

- + Conditionally branch to top of loop?
  - × Say loop is instructions 5—15
- + BR R0 R1 // branch back 10 instrs
  - × // compute condition into r0
  - × // R1 hold -11

40

40

ESE150 Spring 2022

## BRANCHING

```

if (condition)
{true-case}
else
{false-case}
after-loop
    
```

× How

- + Implement if-then?
  - × <condition>
  - × <false-case-code>
  - × <true-case-code>
  - × <after-loop-code>

41

41

ESE150 Spring 2022

## BRANCHING

× How

- + Implement if-then?
  - + Sketch
    - × // compute condition into R0
    - × //Need TRUE\_OFFSET in R7
    - × BR R0 R7
    - × <false-case-code>
    - × // 1 in r0
    - × // load length of true case into R7
    - × BR R0 R7 // branch over true
    - × <true-case-code, target of TRUE\_OFFSET>
    - × <after-loop-code>

42

42



**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

49

49

**LEARN MORE**

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

50

50

**REMINDERS**

- × **Feedback including lab**
- × **Lab 7 due today**
- × **Lab 8 on Wednesday**
  - + Posted on syllabus
  - + Back to using ItsyBitsy
  - + Prelab on Arduino software you installed on your computer for Lab 1
  - + Bring kit to lab

51

51