

Lecture #16 – Stored-Program Processors

ESE 1500 – DIGITAL AUDIO BASICS

ESE1500 Spring 2023

Based on slides © 2009–2023 DeHon

1

ESE1500 Spring 2023

LECTURE TOPICS

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

2

ESE1500 Spring 2023

COURSE MAP – WEEK 9

Music (1)

Numbers correspond to course weeks

sample (2)

freq (4)

psycho-acoustics (5,6)

compress (3)

D/A ← 10101001101

speaker

MP3 Player / iPhone / Droid

3

ESE1500 Spring 2023

REMINDER MEMORY

4

ESE1500 Spring 2023

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

5

ESE1500 Spring 2023

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

12

ESE1500 Spring 2023

QUICK REMINDER

13

13

ESE1500 Spring 2023

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

14

14

ESE1500 Spring 2023

PRECLASS 1

	A	T	F	i0	i1	o	0	1	2	3	4	5	6	7
3	G	&	0	1	3	1	1	0	0	0	0	0	0	0
4	G	&	1	2	4	1	1	0	1	0	0	0	0	0
5	G		3	4	3									
6	G	&	0	2	4									
7	G		3	4	5									
8														

15

15

ESE1500 Spring 2023

STORED-PROGRAM PROCESSOR

16

16

ESE1500 Spring 2023

“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)

17

17

ESE1500 Spring 2023

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

18

18

ESE1500 Spring 2023

Part 2

EXPAND PROCESSOR

19

19

ESE1500 Spring 2023

BUILDING OUT

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

20

20

ESE1500 Spring 2023

PROCESSORS

21

21

ESE1500 Spring 2023

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

ESE1500 Spring 2023

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

23

ESE1500 Spring 2023

MULTIBIT BUS SYMBOLS

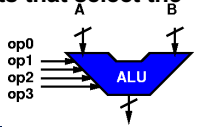
24

24

ESE1500 Spring 2023

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

ESE1500 Spring 2023

ALU OPS (ON 8BIT WORDS)

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

26

26

ESE1500 Spring 2023

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

ESE1500 Spring 2023

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

ESE1500 Spring 2023

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

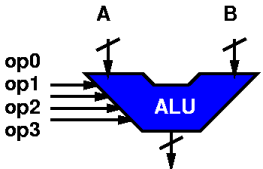
29

29

ESE1500 Spring 2023

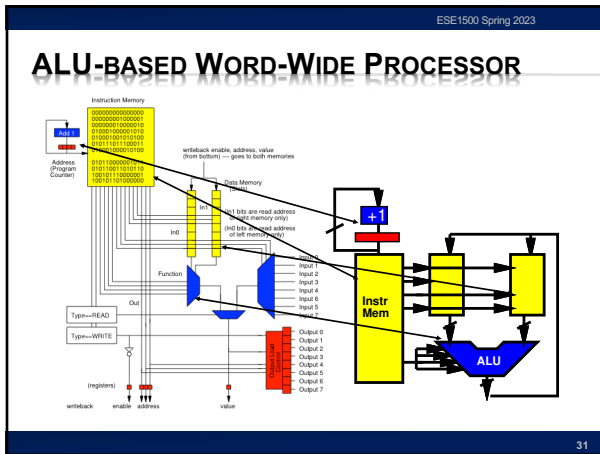
ALU ENCODING

- * **Each operation has some bit sequence**
- * **ADD 0000**
- * **SUB 0010**
- * **INV 0001**
- * **SLL 1110**
- * **SRL 1100**
- * **SRA 1101**
- * **AND 1000**

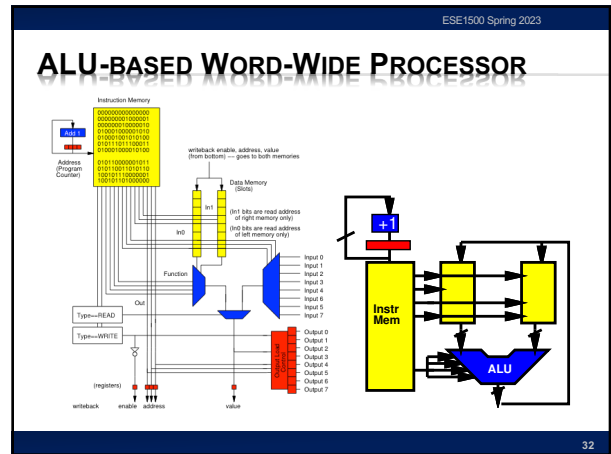


30

30



31



32

BEYOND LINEAR SEQUENCE

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

33

BRANCHING

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

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

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

ESE1500 Spring 2023

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

ESE1500 Spring 2023

BRANCHING

- Given instruction: BR SRC1 SRC2
- BRANCH if (SRC1[0]==1) to PC+SRC2
- Start R0=0, R1=1, R2=a, R3=b, R4=?, R5=4, R6=-3

Addr	Instruction
100	SUB R4 R4 R4
101	BR R2 R5
102	ADD R4 R1 R4
103	SRA R2 none R2
104	BR R1 R6
105	

Function? High-level control operation?

38

38

ESE1500 Spring 2023

LOOPING

```
while (condition)
{body}
after-loop
```

39

39

ESE1500 Spring 2023

BRANCHING

- Given instruction: BR SRC1 SRC2
- BRANCH if (SRC1[0]==1) to PC+SRC2
- Start R0=0, R1=1, R2=a, R3=b, R4=?, R7=3, R8=2, R9=?

Addr	Instruction
200	SUB R2 R3 R9
201	BR R9 R7
202	MOV R1 R4
203	BR R1 R8
204	MOV R0 R4
205	

Function? High-level control operation?

40

40

ESE1500 Spring 2023

CONDITIONAL

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

41

41

ESE1500 Spring 2023

CONTEMPORARY PROCESSORS

42

42

POD, IT'SYBITSY PROCESSOR

- Compare ARM7

32b processor

43

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)

44

LARGE MEMORY

- Add Large Memory for Bulk data storage

Big Memory

45

LOAD-STORE ARCHITECTURE

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

Big Memory

46

ARM LOAD-STORE

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

Big Memory

47

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

48

LEARN MORE

- × **CIS2400 – processor organization and assembly**
- × **CIS4710 – implement and optimize processors**
 - + Including FPGA mapping in Verilog
- × **ESE3700 – implement memories (and gates) using transistors**

49

49

REMINDERS

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

50

50