

## Control & DataPath Part 2

Ref: Chapter 5

CompOrg Fall 2000 - Control & DataPath Part 2

1

---

---

---

---

---

---

---

---

## Datapath

- The *datapath* is the interconnection of the components that make up the processor.
- The datapath must provide connections for moving bits between memory, registers and the ALU.

CompOrg Fall 2000 - Control & DataPath Part 2

2

---

---

---

---

---

---

---

---

## Control

- The control is a collection of signals that enable/disable the inputs/outputs of the various components.
- You can think of the control as the brain, and the datapath as the body.
  - the datapath does only what the brain tells it to do.

CompOrg Fall 2000 - Control & DataPath Part 2

3

---

---

---

---

---

---

---

---

## Datapaths

We looked at individual datapaths that support:

1. Fetching Instructions
2. Arithmetic/Logical Instructions
3. Load & Store Instructions
4. Conditional branch

We need to combine these in to a single datapath.

CompOrg Fall 2000 - Control & DataPath Part 2

4

---

---

---

---

---

---

---

---

## Issues

- When designing one datapath that can be used for any operation:
  - the goal is to be able to handle one instruction per cycle.
  - must make sure no datapath *resource* needs to be used more than once at the same time.
    - if so – we need to provide more than one!

CompOrg Fall 2000 - Control & DataPath Part 2

5

---

---

---

---

---

---

---

---

## Sharing Resources

- We can share datapath resources by adding a multiplexor (and a control line).
  - for example, the second input to the ALU could come from either:
    - a register (as in an arithmetic instruction)
    - from the instruction (as in a load/store – when computing the memory address).

CompOrg Fall 2000 - Control & DataPath Part 2

6

---

---

---

---

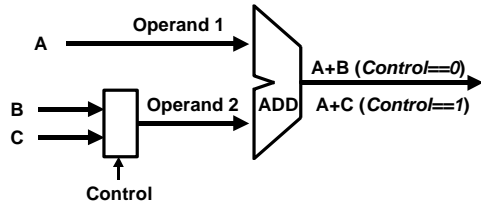
---

---

---

---

## Sharing with a Multiplexor Example



CompOrg Fall 2000 - Control & DataPath Part 2

7

---

---

---

---

---

---

---

---

## Combining Datapaths for memory instructions and arithmetic instructions

- Need to share the ALU
  - For memory instructions used to compute the address in memory.
  - For Arithmetic/Logical instructions used to perform arithmetic/logical operation.

CompOrg Fall 2000 - Control & DataPath Part 2

8

---

---

---

---

---

---

---

---

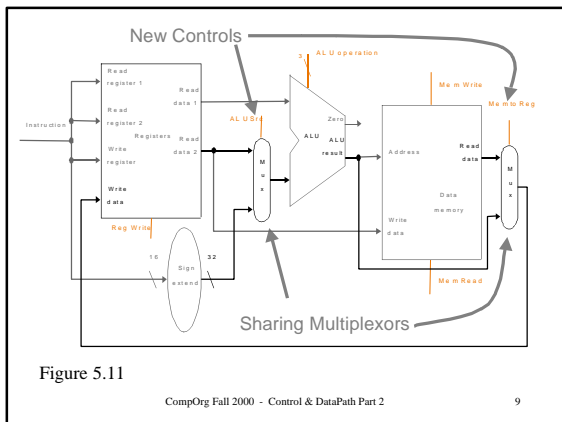


Figure 5.11

CompOrg Fall 2000 - Control & DataPath Part 2

9

---

---

---

---

---

---

---

---

## Adding the Instruction Fetch

- One memory for instructions, separate memory for data.
  - otherwise we might need to use the memory twice in the same instruction.
- Dedicated Adder for updating the PC
  - otherwise we might need to use the ALU twice in the same instruction.

---

---

---

---

---

---

---

---

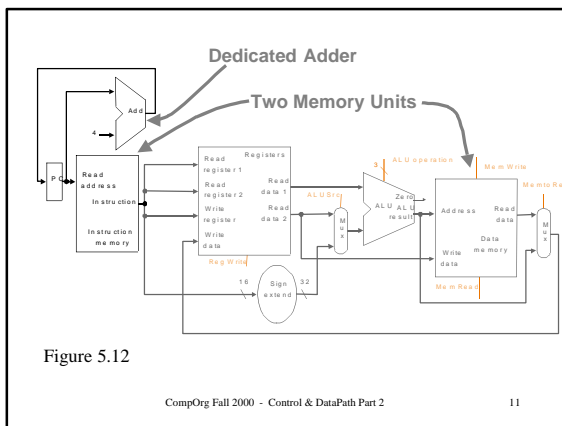


Figure 5.12

---

---

---

---

---

---

---

---

## Need to add datapath for **beq**

- Register comparison (requires ALU).
- Another adder to compute target address.
  - One input to adder is sign extended offset, shifted by 2 bits.
  - Other input to adder is PC+4

---

---

---

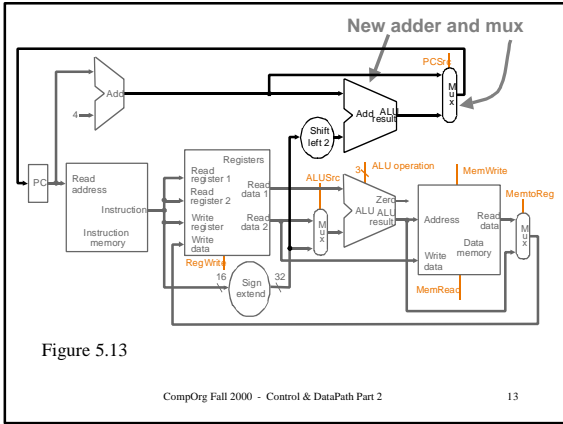
---

---

---

---

---




---

---

---

---

---

---

---

---

## Whew!

- Keep in mind that the datapath we now have supports just a few MIPS instructions!
- Things get worse (more complex) as we support other instructions:  

**j      jal      jr      addi**
- We won't worry about them now...

CompOrg Fall 2000 - Control & DataPath Part 2 14

---

---

---

---

---

---

---

---

## Control Unit

- We need something that can generate the *controls* in the datapath.
- Depending on what kind of instruction we are executing, different controls should be turned on (*asserted*) and off (*deasserted*).
- We need to treat each control individually (as a separate boolean function).

CompOrg Fall 2000 - Control & DataPath Part 2 15

---

---

---

---

---

---

---

---

## Controls

- Our datapath includes a bunch of *controls*:
  - *ALU operation* (3 bits)
  - *RegWrite*
  - *ALUSrc*
  - *MemWrite*
  - *MemtoReg*
  - *MemRead*
  - *PCSrc*

---

---

---

---

---

---

---

---

## ALU Operation Control

- A 3 bit control (assumes the ALU designed in chapter 4):

ALU Control Input	Operation
000	AND
001	OR
010	add
110	subtract
111	slt

---

---

---

---

---

---

---

---

## ALU Functions for our Instructions

**lw**, **sw** (load/store): addition

**beq**: subtraction

**add**, **sub**, **and**, **or**, **slt** (arithmetic/logical):  
All R-format instructions

---

---

---

---

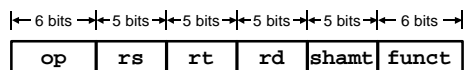
---

---

---

---

## R-Format Instructions



Operation is specified by some bits in the **funct** field in the instruction.

---

---

---

---

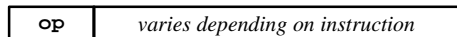
---

---

---

---

## MIPS Instruction OPCODEs



- The MS 6 bits are an OPCODE that identifies the instruction.
- R-Format: always **000000**
  - (funct identifies the operation)

<b>lw</b>	<b>sw</b>	<b>beq</b>
<b>100011</b>	<b>101011</b>	<b>000100</b>

---

---

---

---

---

---

---

---

## Generating ALU Controls

We can view the 3 bit ALU control as 3 boolean functions. Inputs are:

- the **op** field (OPCODE)
- **funct** field (for R-format instructions only)

---

---

---

---

---

---

---

---

## Simplifying The Opcode

For building the ALU Operation Controls, we are interested in only 4 different opcodes.

We can simplify things by first reducing the 6 bit **op** field to a 2 bit value we will call **ALUOp**

---

---

---

---

---

---

---

---

Instruction	ALUOp	funct	ALU action	ALU controls
lw	00	??????	add	010
sw	00	??????	add	010
beq	01	??????	subtract	110
add	10	100000	add	010
sub	10	100010	subtract	110
and	10	100100	and	000
or	10	100101	or	001
slt	10	101010	slt	111

---

---

---

---

---

---

---

---

## Build a Truth Table

- We can now build a truth table for the 3 bit ALU control.
- Inputs are:
  - 2 bit **ALUOp**
  - 6 bit **funct** field
- Abbreviated Truth Table: only show the rows we care about!

---

---

---

---

---

---

---

---

ALUOp	func							ALU Control
0	0	x	x	x	x	x	x	010
x	1	x	x	x	x	x	x	110
1	x	x	x	0	0	0	0	010
1	x	x	x	0	0	1	0	110
1	x	x	x	0	1	0	0	000
1	x	x	x	0	1	0	1	001
1	x	x	x	1	0	1	0	111

x means "don't care"

CompOrg Fall 2000 - Control & DataPath Part 2 25

---

---

---

---

---

---

---

---

---

---

---

---

### Adding the ALU Control

- We can now add the ALU control to the datapath:
  - inputs to this control come from the instruction and from ALUOp
- If we try to show all the details the picture becomes too complex:
  - just plop in an "ALU Control" box.

CompOrg Fall 2000 - Control & DataPath Part 2 26

---

---

---

---

---

---

---

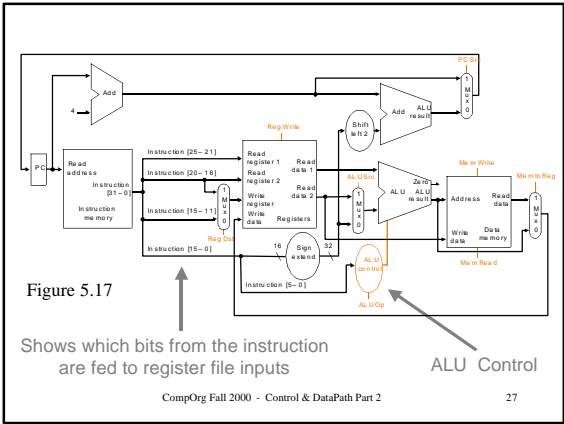
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

---

---

## Implementing Other Controls

- The other controls in our datapath must also be specified as functions.
- We need to determine the inputs to all the functions.
  - primarily the inputs are part of the instructions, but there are exceptions.
- Need to define precisely what conditions should turn on each control.

CompOrg Fall 2000 - Control & DataPath Part 2

28

---

---

---

---

---

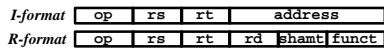
---

---

---

## RegDst Control Line

- Controls a multiplexor that selects one of the fields **rt** or **rd** from an R-format or I-format instruction.
- I-Format is used for load and store.
- **sw** needs to *write* to the register **rt**.



CompOrg Fall 2000 - Control & DataPath Part 2

29

---

---

---

---

---

---

---

---

## RegDst usage

- **RegDst** should be
  - 0 to send **rt** to the write register # input.
  - 1 to send **rd** to the write register # input.
- **RegDst** is a function of the opcode field:
  - If instruction is **sw**, **RegDst** should be 0
  - For all other instructions **RegDst** should be 1

CompOrg Fall 2000 - Control & DataPath Part 2

30

---

---

---

---

---

---

---

---

## RegWrite Control

- a **1** tells the register file to write a register.
  - whatever register is specified by the write register # input is written with the data on the write register data inputs.
- Should be a **1** for arithmetic/logical instructions and for a store.
- Should be a **0** for load or **beq**.

CompOrg Fall 2000 - Control & DataPath Part 2

31

---

---

---

---

---

---

---

---

## ALUSrc Control

- MUX that selects the source for the second ALU operand.
  - 1 means select the second register file output (read data 2).
  - 0 means select the sign-extended 16 bit offset (part of the instruction).
- Should be a **1** for load and store.
- Should be a **0** for everything else.

CompOrg Fall 2000 - Control & DataPath Part 2

32

---

---

---

---

---

---

---

---

## MemRead Control

- A **1** tells the memory to put the contents of the memory location (specified by the address lines) on the Read data output.
- Should be a **1** for load.
- Should be a **0** for everything else.

CompOrg Fall 2000 - Control & DataPath Part 2

33

---

---

---

---

---

---

---

---

### MemWrite Control

- 1 means that memory location (specified by memory address lines) should get the value specified on the memory Write Data input.
- Should be a **1** for store.
- Should be a **0** for everything else.

CompOrg Fall 2000 - Control & DataPath Part 2

34

---

---

---

---

---

---

---

---

### MemToReg Control

- MUX that selects the value to be stored in a register (that goes to register write data input).
  - 1 means select the value coming from the memory data output.
  - 0 means select value coming from the ALU output.
- Should be a **1** for load and any arithmetic/logical instructions.
- Should be a **0** for everything else (**sw, beq**).

CompOrg Fall 2000 - Control & DataPath Part 2

35

---

---

---

---

---

---

---

---

### PCSrc Control

- MUX that selects the source for the value written in to the PC register.
  - 1 means select the output of the Adder used to compute the relative address for a branch.
  - 0 means select the output of the PC+4 adder.
- Should be a **1** for beq if registers are equal!
- Should be a **0** for other instructions or if registers are different.

CompOrg Fall 2000 - Control & DataPath Part 2

36

---

---

---

---

---

---

---

---

## PCSrc depends on result of ALU operation!

- This control line can't be simply a function of the instruction (all the others can).
- **PCSrc** should be a 1 only when:
  - **beq** AND **ALU zero** output is a 1
- We will generate a signal called “branch” that we can AND with the ALU zero output.

---

---

---

---

---

---

---

---

## Truth Table for Control

Instruction	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp
R-format	1	0	0	1	0	0	0	10
lw	0	1	1	1	1	0	0	00
sw	x	1	x	0	0	1	0	00
beq	x	0	x	0	0	0	1	01

---

---

---

---

---

---

---

---

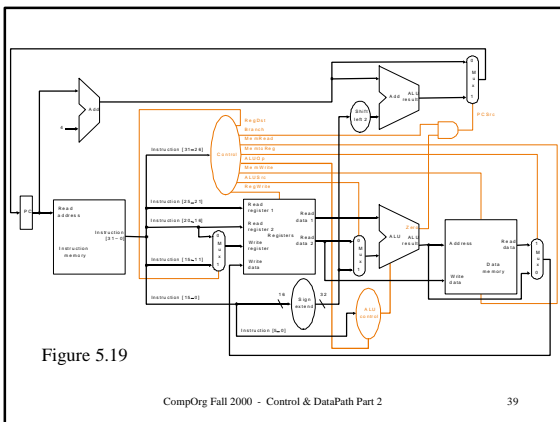


Figure 5.19

---

---

---

---

---

---

---

---

## Single Cycle Instructions

- View the entire datapath as a combinational circuit.
- We can follow the *flow* of an instruction through the datapath.
  - single cycle instruction means that there are not really any *steps* – everything just happens and becomes finalized when the clock cycle is over.

CompOrg Fall 2000 - Control & DataPath Part 2

40

---

---

---

---

---

---

---

---

## `add $t1,$t2,$t3`

- Control Lines:
  - ALU Controls specify an ALU add operation.
  - RegWrite will be a 1 so that when the clock cycle ends the value on the Register Write Input lines will be written to a register.
  - all other control lines are 0.

CompOrg Fall 2000 - Control & DataPath Part 2

41

---

---

---

---

---

---

---

---

## `lw $t1,offset($t2)`

- Control Lines:
  - ALU Control set for an add operation.
  - ALUSrc is set to 1 to indicate the second operand is sign extended offset.
  - MemRead would be a 1.
  - RegDst would select the correct bits from the instruction to specify the dest. register.
  - RegWrite would be a 1.

CompOrg Fall 2000 - Control & DataPath Part 2

42

---

---

---

---

---

---

---

---

## Disadvantage of single cycle operation

If we have instructions execute in a single cycle, then the cycle time must be long enough for the slowest instruction.

- all instructions take the same time as the slowest.

CompOrg Fall 2000 - Control & DataPath Part 2

43

---

---

---

---

---

---

---

---

## Multicycle Implementation

- Chop up the processing of instructions in to discrete stages.
- Each stage takes one clock cycle.
  - we can implement each stage as a big combinational circuit (like we just did for the whole thing).
  - provide some way to sequence through the stages.

CompOrg Fall 2000 - Control & DataPath Part 2

44

---

---

---

---

---

---

---

---

## Advantages of Multicycle

- Only need those stages required by an instruction.
  - the control unit is more complex, but instructions only take as long as necessary.
- We can share components
  - perhaps 2 different stages can use the same ALU.
  - We don't need to duplicate resources.

CompOrg Fall 2000 - Control & DataPath Part 2

45

---

---

---

---

---

---

---

---

## Additional Resources for Multicycle

- To implement a multicycle implementation we need some additional registers that can be used to hold intermediate values.
  - instruction
  - computed address
  - result of ALU operation
  - ...

CompOrg Fall 2000 - Control & DataPath Part 2

46

---

---

---

---

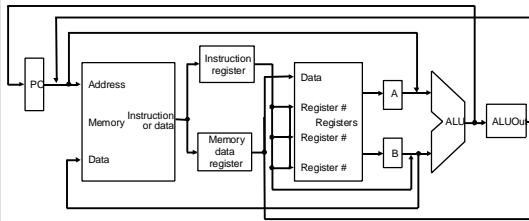
---

---

---

---

## Multicycle Datapath



CompOrg Fall 2000 - Control & DataPath Part 2

47

---

---

---

---

---

---

---

---

## Instruction Stages

- Instruction Fetch
- Instruction decode/register fetch
- ALU operation/address computation
- Memory Access
- Register Write

CompOrg Fall 2000 - Control & DataPath Part 2

48

---

---

---

---

---

---

---

---

## Control for Multicycle

- Need to define the controls
- Need to come up with some way to sequence the controls
  - Two techniques
    - finite state machine (we will skip)
    - microprogramming

CompOrg Fall 2000 - Control & DataPath Part 2

49

---

---

---

---

---

---

---

---

## MicroProgramming

- The idea is to build a (very small) processor to generate the controls signals at the right time.
- At each stage (cycle) one *microinstruction* is executed – the result changes the value of the control signals.
- Somebody writes the *microinstructions* that make up each MIPS instruction.

CompOrg Fall 2000 - Control & DataPath Part 2

50

---

---

---

---

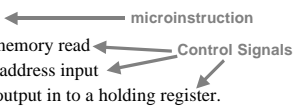
---

---

---

---

## Example microinstructions

Fetch next instruction: 

- turn on instruction memory read
- feed PC to memory address input
- write memory data output in to a holding register.

Compute Address:

- route contents of base register to ALU
- route sign-extended offset to ALU
- perform ALU add
- write ALU output in to a holding register.

CompOrg Fall 2000 - Control & DataPath Part 2

51

---

---

---

---

---

---

---

---

## Sequencing

- In addition to setting some control signals, each microinstruction must specify the next microinstruction that should be executed.
- 3 Options:
  - execute next microinstruction (default)
  - start next MIPS instruction (Fetch)
  - Dispatch (depends on control unit inputs).

CompOrg Fall 2000 - Control & DataPath Part 2

52

---

---

---

---

---

---

---

---

## Microinstruction Format

- A bunch of bits – one for each control line needed by the control unit.
  - bits specify the values of the control lines directly.
- Some bits that are used to determine the next microinstruction executed.

CompOrg Fall 2000 - Control & DataPath Part 2

53

---

---

---

---

---

---

---

---

## Dispatch Sequencing

- Can be implemented as a table lookup.
  - bits in the microinstruction tell what row in the table.
  - inputs to the control unit tell what column.
  - value stored in table determines the microaddress of the next microinstruction.
- This is a simplified description (called a *microdescription*)

CompOrg Fall 2000 - Control & DataPath Part 2

54

---

---

---

---

---

---

---

---

## nano-nano-instructions

- Embed a single bit, frequency modulated, signal based processor in a dispatch table.
- The LS bit of the indirect address is used to determine the number of iterations in the microloop.
- The pico-assembler constructs the mini-object method calls as virtual members of the Florida election commission.
- It's that simple!

*Dave is up too late, ignore this slide...*

CompOrg Fall 2000 - Control & DataPath Part 2

55

---

---

---

---

---

---

---

---

## Moving On

- We've skipped lots of chapter 5!
- I hope you understand the general idea.
  
- We will deal with multicycle implementation more when we talk about pipelining...

CompOrg Fall 2000 - Control & DataPath Part 2

56

---

---

---

---

---

---

---

---