

## EE30342 – Digital Design with HDL (II)

### Lectures 10-11

Dr. Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

1

## In This Session

- Finite-State Control for Multicycle Datapath
- Exception Handling

2

## Review – Multicycle Datapath

- Each instruction has multiple stages
- Each stage takes one cycle

1. Instruction fetch
2. Instruction decode / Data fetch
3. Execution, address computation
4. Memory access / R-format completion
5. Memory read completion

*All instructions use these*

☺ Each stage can re-use hardware from previous stage

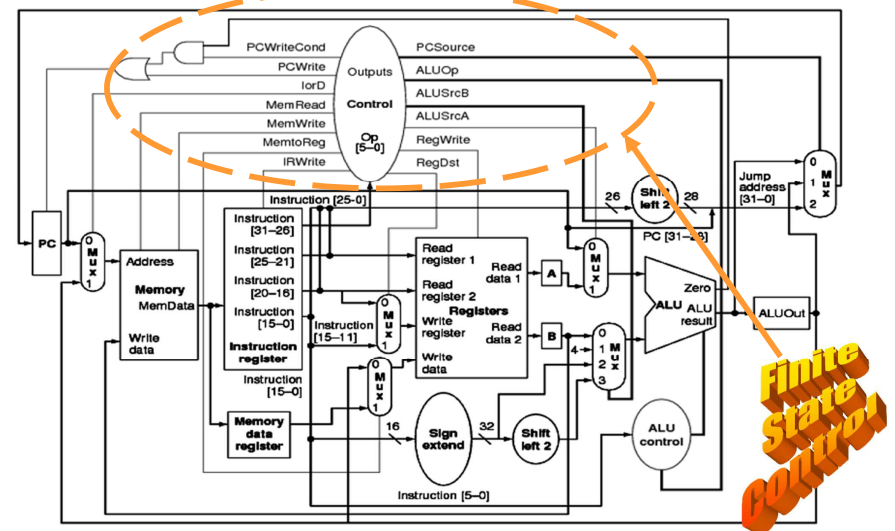
☺ More efficient use of hardware and time

>> **New Hardware + Muxes** required for buffering, control

>> **Expanded Control** for new hardware

3

## Today: Multicycle DP Control



## Multicycle DP: 1-bit Ctl. Signals

Signal Name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register number destination number for the Write register comes from the rd field.
RegWrite	None	The general-purpose register selected by the Write register input is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC.	The first ALU operand comes from the A register.
MemRead	None	Content of memory at the location specified by the Address input is put on the Memory data output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by the value on Write data input.
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
lorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None.	The output of memory is written into the IR.
PCWrite	None.	The PC is written; the source is controlled by the PCSrc.
PCWriteCond	None.	The PC is written if the Zero output from the ALU is also active.

5

## Multicycle DP: 2-bit Ctl. Signals

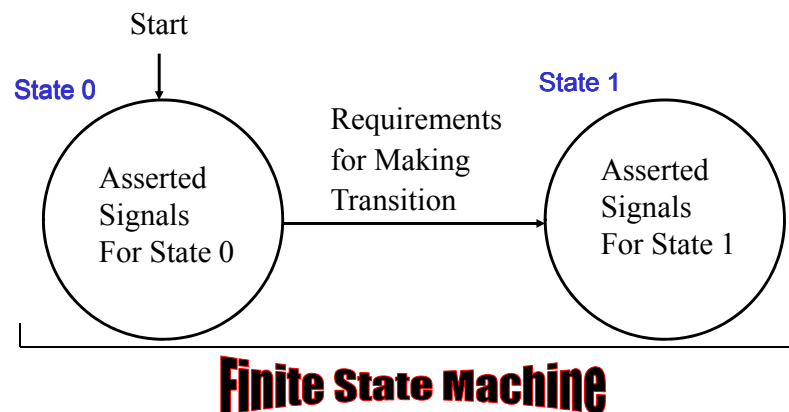
Signal Name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC+4) is sent to the PC for writing.
	01	The contents of the ALUOut ( the branch target address) are sent to the PC for writing.
	10	The jump target address ( IR[25:0]) shifted left 2 bits and concatenated with PC+4[31:28] is sent to the PC for writing.

6

## Basics of Finite State Control (FSC)

**State:** Snapshot of machine (map of ones and zeroes)

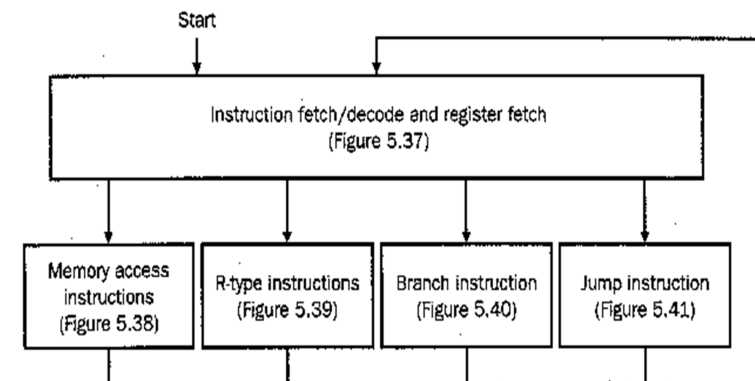
**Transition:** Going from one state to another



7

## FSC for Multicycle Datapath

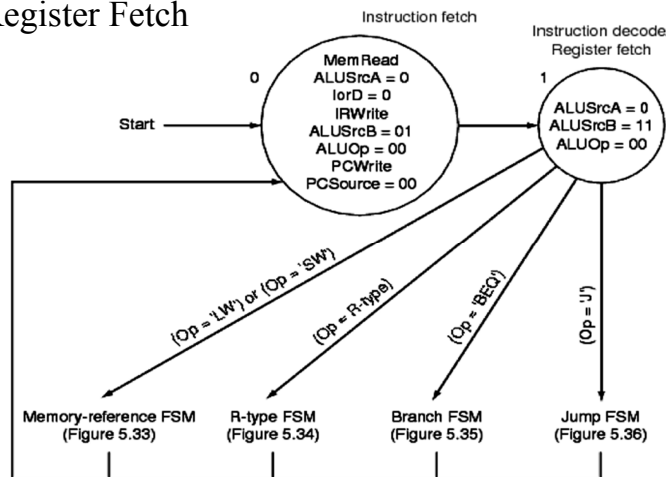
### High-Level View – Useful for Simplifying Abstractions



8

## FSC for Instr Fetch & Decode

**Recall:** All instr's. use Instruction Fetch/Decode, Register Fetch



9

## Multicycle DP: Store Word (sw)

**Step 1:** Fetch instr. // Store in IR // Compute PC + 4

**Step 2:** Decode instruction: **opcode, rs, rt, offset** fields

Data fetch: Apply **rt** to Register File => Base address

Data Read into **A** buffer register (Base)

SignExt, Shift **offset** field into **B** buffer register

**Step 3:** ALU operation : Base + Offset

ALU output goes into **ALUout** register

**Step 4:** **ALUout** register contents applied as Memory Address

Assert: **MemWrite**

**CPI for Store = 4 cycles**

10

## Multicycle DP: Load Word (lw)

**Step 1:** Fetch instr. // Store in IR // Compute PC + 4

**Step 2:** Decode instruction: **opcode, rd, rt, offset** fields

Data fetch: Apply **rt** to Register File => Base address

Data Read into **A** buffer register (Base)

SignExt, Shift **offset** field into **B** buffer register

**Step 3:** ALU operation: Base + Offset

ALU output goes into **ALUout** register

**Step 4:** **ALUout** register contents applied as Memory Address

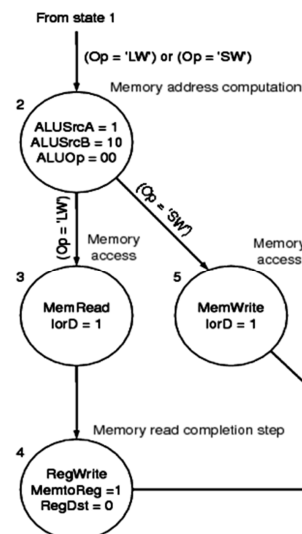
Assert: **MemRead**

**Step 5:** Memory Data Out is written into **rd**.

**CPI for Load = 5 cycles**

11

## FSC for Load/Store Instructions



Assume Fetch/Decode completed

Set ALU inputs from A, SignExt  
offset field  
Perform ALU op to get MemAddr

Perform Memory Access (read  
/write)

If Load, then do Register Write  
Go back for another instruction

12

## Multicycle DP: R-format

**Step 1:** Fetch instr. // Store in IR // Compute PC + 4

**Step 2:** Decode instruction: **opcode**, **rd**, **rs**, **rt**, **funct** fields

Data fetch: Apply **rs**, **rt** to Register File

Data Read into **A,B** buffer registers

**Step 3:** ALU operation

ALU output goes into **ALUout** register

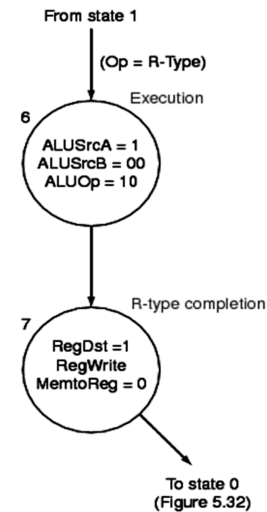
**Step 4:** **ALUout** register contents is written into **rd**

Assert: **RegWrite,RegDst**

**CPI for R-format = 4 cycles**

13

## FSC for R-format Instructions



Assume Fetch/Decode completed

Set ALU inputs from A,B buffers

Perform ALU op

Select **rd** as Destination Register

Write **ALUout** buffer to Register File

Go back for another instruction

14

## Multicycle DP: Cond. Branch

**Step 1:** Fetch instr. // Store in IR // Compute PC + 4

**Step 2:** Decode instruction: **opcode**, **rs**, **rt**, **offset** fields

Data fetch: Apply **rs**, **rt** to Register File

BTA calc: SignExt,Shift **offset** field into **B** buffer register

**ALU** compose PC, offset => BTA

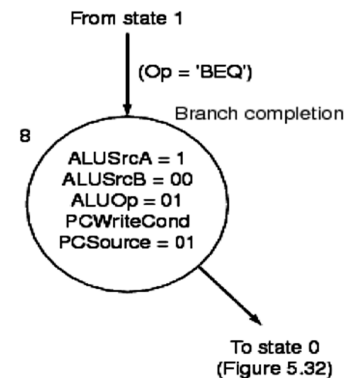
**Step 3:** **ALU** operation: compare **rs**, **rt**

ALU output present at **Zero** register causes Control to select BTA or PC+4

**CPI for Conditional Branch = 3 cycles**

15

## FSC for Branch Instruction



Assume Fetch/Decode completed

Set ALU inputs from A,B buffers

Perform ALU op: compare

Apply BTA or PC+4 to PC

Go back for another instruction

16

## Multicycle DP: Jump

**Step 1:** Fetch instr. // Store in IR // Compute PC + 4

**Step 2:** Decode instruction: **opcode**, **address** fields

JTA calc: SignExt, Shift **offset** field [Bits 27-0]

**Concatenate** with PC [Bits 31-28] => JTA

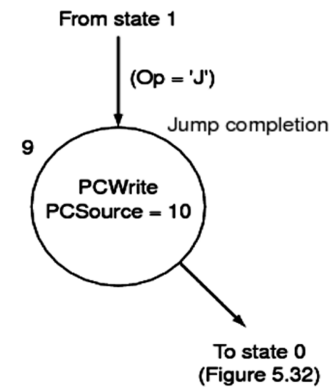
**Step 3:** PC replaced by the Jump Target Address (JTA)

**PCsource** = 10, **PCWrite** asserted

**CPI for Jump = 3 cycles**

17

## FSC for Jump Instruction



Assume Fetch/Decode completed

Set PC to be overwritten

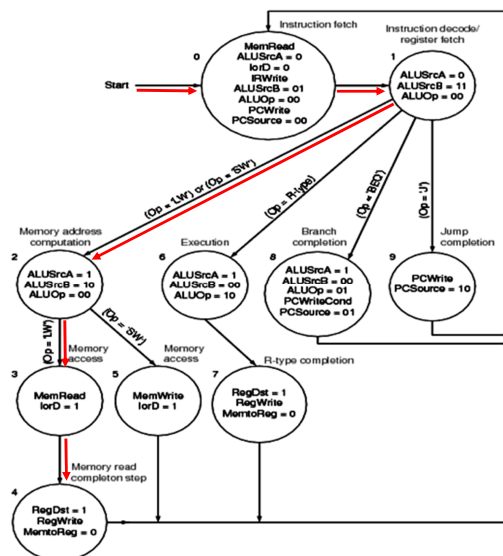
Hardware composes JTA

Apply JTA to PC

Go back for another instruction

18

## FSC for Multicycle Datapath



10 states

CPI = No. of states encountered on path for that instruction

R-format = 4 states

Store = 4 states

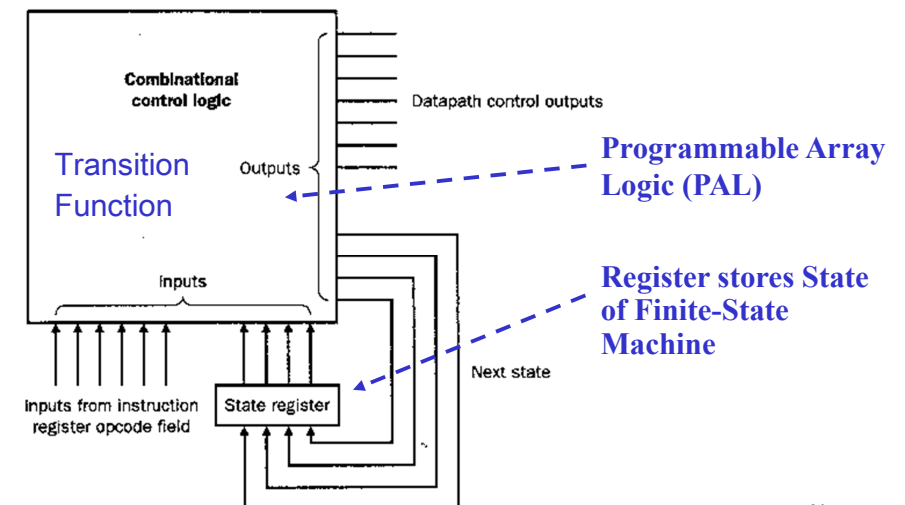
**Load = 5 states [0,1,2,3,4]**

Branch = 3 states

Jump = 3 states

19

## Hardware for Multicycle DP FSC



20

## New Topic: Exceptions

**Definition:** Event causes unexpected transfer of control

**Types:** (1) Exception [overflow] (2) Interrupt [I/O]

**Difference:** *Exception* generated inside the processor  
*Interrupt* associated with *external* event

**Challenges:**

- *Exception Detection* – How to discover exception
- *Exception Handling* – What to do

**MIPS:** 2 types – Undefined instruction, Arith. overflow

21

## Exception Detection

- **Undefined Instruction:**

1. Add State 10 [Exception] to Finite-State Control
2. Every instruction, which is not lw, sw, beq, R-format, or jump, will go to State 10

- **Arithmetic Overflow:**

1. Recall: ALU has overflow detection logic. So, create a new State 11 in FSC to handle overflow
2. When ALU output *Overflow* is asserted, then control is transferred to State 11

22

## Exception Handling

- **Two Techniques:** *EPC/Cause* and *Vectored Interrupts*

- *Vectored Interrupts:*

1. Each exception has a distinct address  $A_E$  associated with it
2. Exception detected  $\Rightarrow A_E$  for that exception written to PC

- *EPC / Cause:* (MIPS practice)

1. Exception detected  $\Rightarrow$  Address of instruction saved in **\$epc**  
**Cause** register has code for exception
2. Exception handler acts on **Cause**, tries to restart execution at instruction pointed to by **\$epc**

23

## Mods to MIPS DP for Exceptions

1. **New Registers** - **\$epc** and **Cause** (32-bit)

2. **New Control Signals** – EpcWrite, CauseWrite

3. **New Control Line** – 0 for undefined instr., 1 for ovflw

4. **New Mux Signal** for PCsource =  $11_2$

- Old PC inputs: PC+4, BTA, JTA
- Additional input:  $A_E = C0000000_{16}$  in MIPS

*Recall:* ALU overflow detection already “installed”

24

## New FSC with Exception States

