

CSE 675.02: Introduction to Computer Architecture

Designing MIPS Processor (Multi-Cycle)

Presentation H

Reading Assignment: 5.5,5.6

Multi-Cycle Design Principles

- Break up execution of each instruction into steps.
- The number of steps and the tasks in each step are instruction dependent.
- Each step takes one clock cycle.
- Balance the amount of work to be done in each clock cycle.
- Restrict each cycle to use only one major functional unit in the data path, or if more than one major functional unit used they should be used in parallel.
- Major units are memory, register file and ALU, since we assume that they introduce the most significant delays during execution of instructions.
- We assume all other delays in the wiring is negligible.

Multi-Cycle Design Principles (cont.)

- During execution of any instruction, we may be reusing functional units, but in different steps (clock cycles), e.g.
 - Single memory can be used for instruction and data,
 - ALU will be used to compute not only tasks it performed in the single-cycle design (e.g. lw & sw addresses and R-type instruction calculations), but it will be used to increment PC (by 4) and to calculate branch target address.
- Control signals will not be determined solely by the instruction in execution (i.e. its op-code and/or function code) but also by the particular clock cycle the instruction is being executed in.
- At the end of each cycle during instruction execution **store intermediate values for use in later cycles.**
- For that purpose, introduce **additional “internal” registers.**

g. babic

Presentation H

3

Elaboration on Work Balance in Each Step

- During any given step **it is not allowed** to have a serial combination of usage of the major functional units; for example:
 - **It is not allowed** that in one step contents of registers are read from the register file and then those contents are used as operands for ALU in the same step, or
 - **It is not allowed** that in one step ALU performs a function on some operands and its result is used as an address for memory read or write in the same step.
- **This principle is introduced to avoid that any step requires too much time, implying that clock cycles have to be of that unnecessary length.**
- Notice that two of the major functional units **are allowed** to be used in parallel, e.g. reading contents from a register file and the ALU performing a function on unrelated data at the same time.

g. babic

Presentation H

4

Five Steps In Instruction Execution

- Major steps in execution of an instruction are:
 - Instruction Fetch
 - Instruction Decode and Register Fetch
 - Execution, Memory Address Computation, or Branch Completion
 - Memory Access or R-type instruction completion
 - Write-back step
- Not every instruction will have all those steps
- Our instructions will take 3-5 steps, i.e. 3-5 clock cycles.
- The first two steps are common to all instructions.

g. babic

Presentation H

5

Multi-Cycle Datapath High Level View

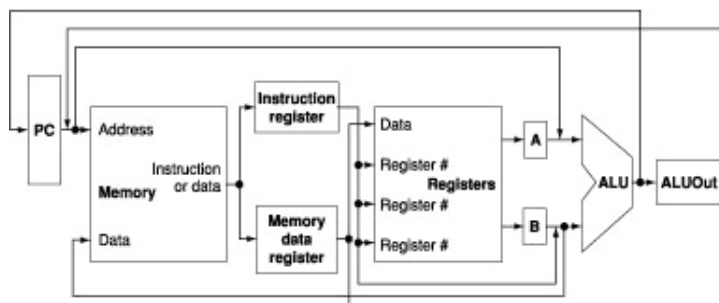


Figure 5.25

- The use of shared functional units requires new temporary registers that hold data between clock cycles of the same instruction.
- The additional registers are:
 - Instruction register (IR),
 - Memory data register (MDR),
 - A and B registers,
 - ALUOut register.

6

Step 1: Instruction Fetch

- Use PC to get instruction and put it in the Instruction Register, i.e. $IR \leftarrow \text{Memory}[PC]$; *lorD=0, MemRead, IRWrite*
- Increment the PC by 4 and put the result back in the PC, i.e. $PC \leftarrow [PC] + 4$; *ALUSrcA=0, ALUSrcB=01, ALUOp=00, PCSource=0, PCWrite*
- Here are rules for signals that are omitted:
 - If signal for mux is not stated, it is *don't care*
 - If ALU signals are not stated, they are *don't care*
 - If MemRead, MemWrite, RegWrite, IRWrite, PCWrite or PCWriteCond is not stated, it is *unasserted*, i.e. *logical 0*.

g. babic

Presentation H

9

Step 2: Instruction Decode & Register Fetch

- We aren't setting any control lines based on the instruction type, since we are busy "decoding" it in our control logic.
- Read registers rs and rt in case we need them:
 $A \leftarrow \text{Reg}[IR[25-21]]$;
 $B \leftarrow \text{Reg}[IR[20-16]]$;
Done automatically
- Compute the branch address in case the instruction is a branch:
 $\text{ALUOut} \leftarrow PC + (\text{sign-extend}(IR[15-0]) \ll 2)$;
ALUSrcA=0, ALUSrcB=11, ALUOp=00

g. babic

Presentation H

10

Step 3: Execute, Mem Addr, Branch

- ALU is performing one of three functions, based on instruction type
- Memory Reference (lw or sw):
 $ALUOut \leftarrow A + \text{sign-extend}(IR[15-0]);$
 $ALUSrcA=1, ALUSrcB=10, ALUOp=00$
- R-type:
 $ALUOut \leftarrow A \text{ op } B;$
 $ALUSrcA=1, ALUSrcB=00, ALUOp=10$
- Branch on Equal:
if $(A==B)$ $PC \leftarrow ALUOut;$
 $ALUSrcA=1, ALUSrcB=00, ALUOp=01$
 $PCSource=01, PCWriteCond$

Note: beq instruction is done, thus this instruction requires 3 clock cycles to execute.

g. babic

Presentation H

11

Steps 4 and 5: Instruction Dependent

- Step 4: R-type and Memory Access
 - Loads and stores access memory
 $MDR \leftarrow \text{Memory}[ALUOut]$ (load); $lorD=1, MemRead$
or
 $\text{Memory}[ALUOut] \leftarrow B$ (store); $lorD=1, MemWrite$
 - R-type instructions finish
 $\text{Reg}[IR[15-11]] \leftarrow ALUOut;$ $\text{RegDst}=1, \text{MemToReg}=0,$
 RegWrite
Register write actually takes place at the end of the cycle on the falling edge
 - Store and R-type instructions are done in 4 clock cycles
- Step 5: Write back (load only) $\text{RegDst}=0, \text{MemToReg}=1,$
– $\text{Reg}[IR[20-16]] \leftarrow MDR$ RegWrite

g. babic

Presentation H

12

Summary of Instruction Executions

Figure 5.30

Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps
Instruction fetch	$IR \leftarrow \text{Memory}[PC]$ $PC \leftarrow PC + 4$			
Instruction decode/register fetch	$A \leftarrow \text{Reg}[IR[25-21]]$ $B \leftarrow \text{Reg}[IR[20-16]]$ $ALUOut \leftarrow PC + (\text{sign-extend}(IR[15-0]) \ll 2)$			
Execution, address computation, branch/jump completion	$ALUOut \leftarrow A \text{ op } B$	$ALUOut \leftarrow A + \text{sign-extend}(IR[15-0])$	if $(A == B)$ then $PC \leftarrow ALUOut$	$PC \leftarrow PC[31-28] \parallel (IR[25-0] \ll 2)$
Memory access or R-type completion	$\text{Reg}[IR[15-11]] \leftarrow ALUOut$	Load: $MDR \leftarrow \text{Memory}[ALUOut]$ or Store: $\text{Memory}[ALUOut] \leftarrow B$		
Memory read completion		Load: $\text{Reg}[IR[20-16]] \leftarrow MDR$		

Note: Jump instruction added: $PCSource=10, PCWrite$

g. babic

Presentation H

13

Implementing Control

- Values of control signals are dependent on:
 - what instruction is being executed and
 - which step (i.e. clock cycle) is being performed.
- Use the information we've accumulated to specify a finite state machine – FSM:
 - specify the finite state machine graphically, or
 - use microprogramming.
- Then, an implementation can be derived from specification.

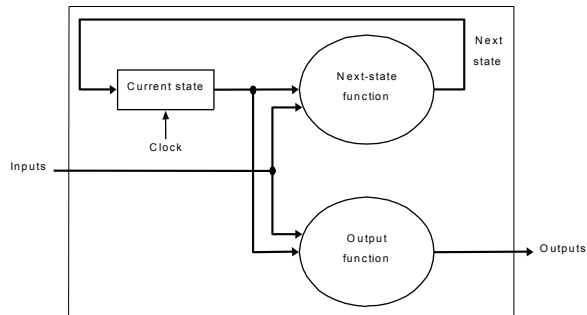
g. babic

Presentation H

14

Finite State Machines

Figure B.10.1



- A current state is kept in the Current state register.
- Next state function and Output function are determined by Current state and Inputs.
- In our case, Output function is based only on Current state.

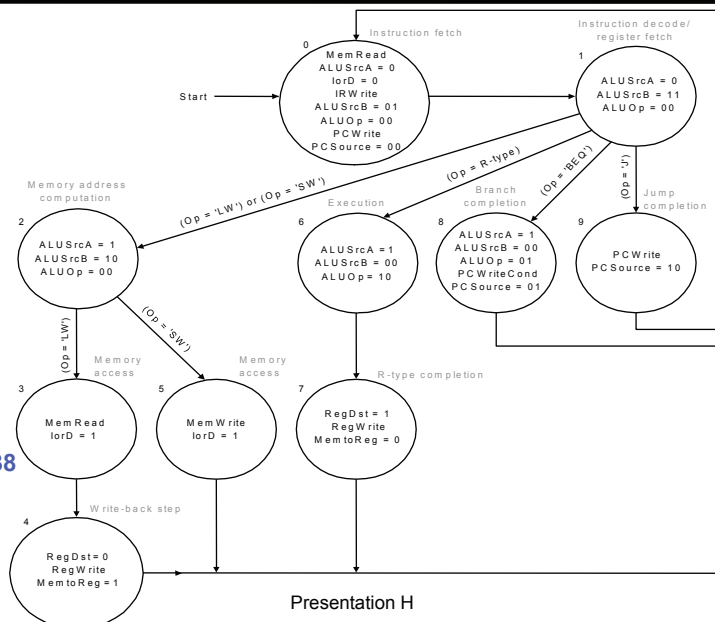
g. babic

Presentation H

15

Finite State Machine Graph for Control Unit

Figure 5.38

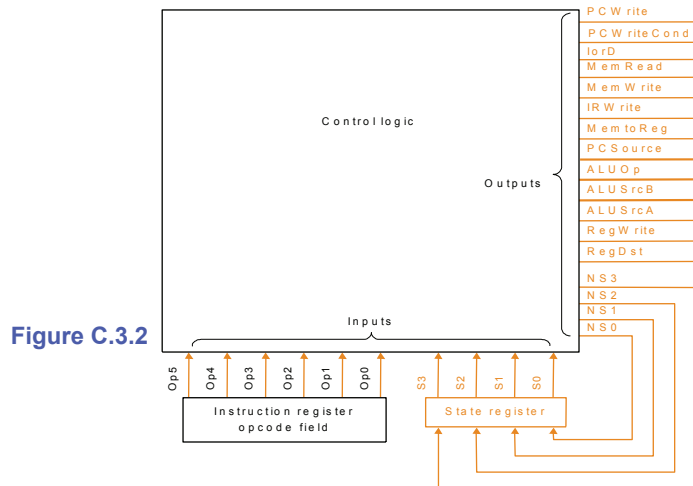


g. babic

Presentation H

16

Implementation of FSM for Control Unit



g. babic

Presentation H

17

MIPS Interrupt Processing

We are implementing processing of only two exceptions:

- illegal op- code and
- integer overflow.

When any of the exceptions occurs, MIPS processor processes the exception (as any other interrupt) in the following 3 steps:

Step 1: EPC register gets a value equal to address of a faulty instruction.

Step 2.: $PC \leftarrow 80000180_{16}$

Cause register \leftarrow a code of the exception

- illegal op-code = 10
- integer overflow = 12

Step 3. Processor is now running in Kernel mode.

Note: we are not implementing step 3.

g. babic

Presentation H

18

Multi-Cycle Datapath for Exception Handling

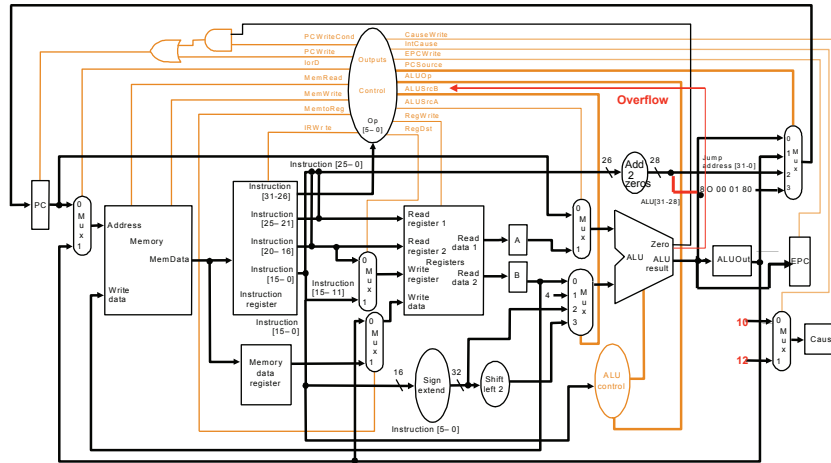


Figure 5.39
with corrections in red

g. babic

Presentation H

19

FSM Graph with Exception Handling

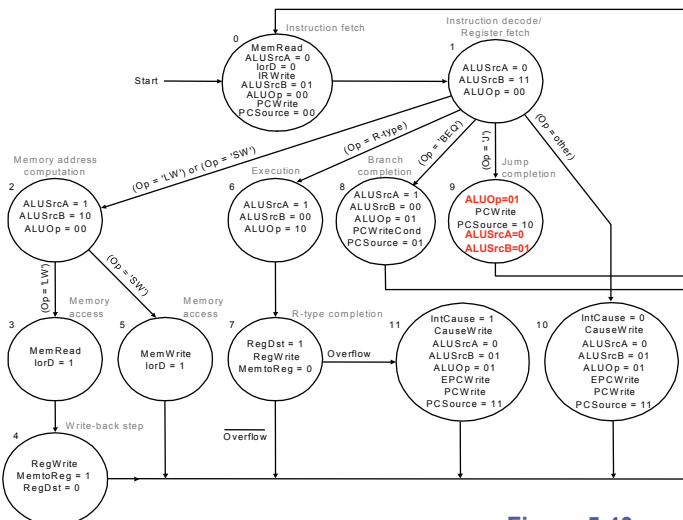


Figure 5.40
with additions in red

g. babic

Presentation H

20