Computer Architecture – Project #2

MIPS Multi-Cycle CPU Implementation

1. Introduction

Although the **single-cycle design** will work correctly, it would not be used in modern designs because it is inefficient. To see why this is so, notice that the clock cycle must have the same length (Critical Path) for every instruction in this single-cycle design, and the CPI = 1.

As a solution about this problem, We can broke each instruction into a series of steps corresponding to the functional unit operations. We can use these steps to create a multiple clock cycle (or a multi-cycle) implementation. In a **multi-cycle implementation**, each step in the execution will take 1 clock cycle. The multi-cycle implementation **allows a functional unit to be used more than once** per instruction, as long as it is **used on different clock cycles**. This sharing can help **reduce** the amount of **hardware required**. The ability to allow instructions to take different numbers of clock cycles and the ability to share functional units within the execution of a single instruction are the major advantages of a multi-cycle design**.** The following figure shows an example of the similar multicycle CPU implementation.

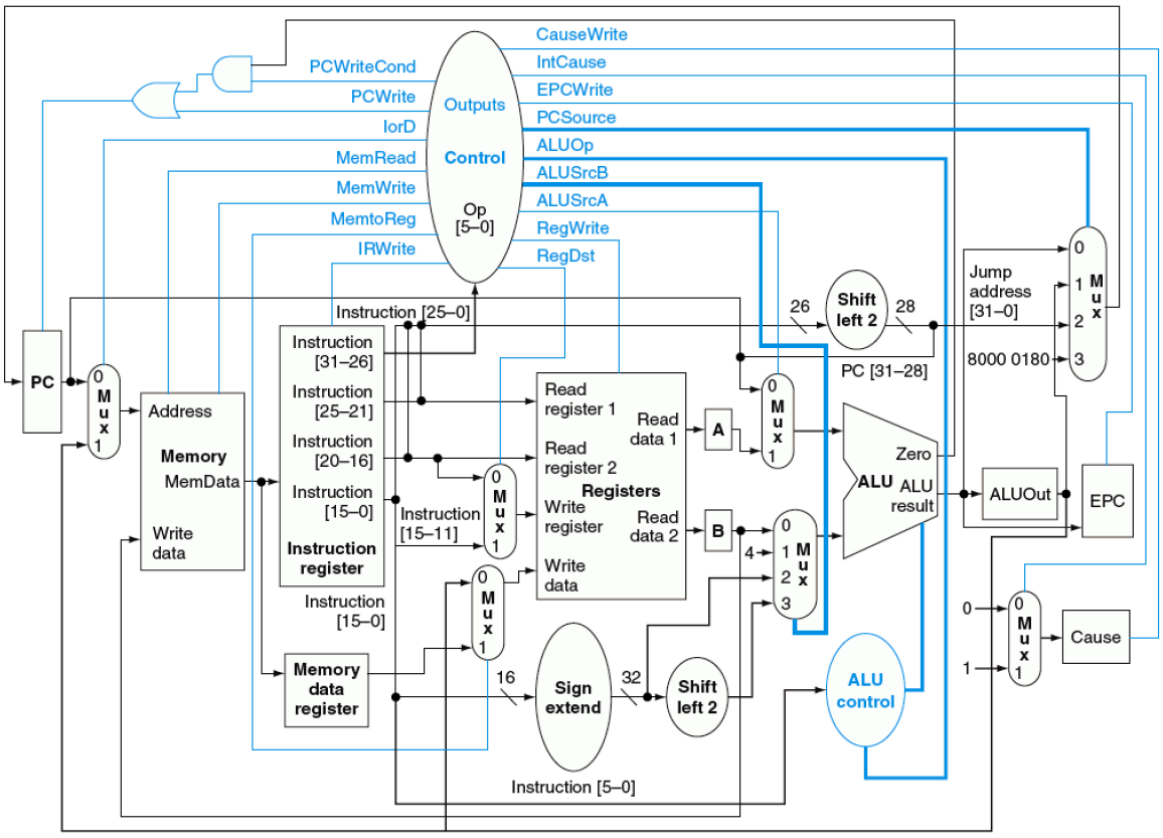

Figure 1 - The multi-cycle datapath together with the necessary control lines

Stage #	Action for R-type instructions	Action for Memory instructions	Action for Branches	Action for Jumps	Remarks
0	IR = Memory[PC] PC = PC + 4				Instr Fetch
1	A = Reg [IR[25:21]] B = Reg [IR[20:16]] ALUOut = PC + (Sign-Extened (IR[15:0]) << 2)				Instr Decode & Register Fetch
2	ALUOut = A op B	ALUOut = A + Sign-Extended (IR[15:0])	If (A==B) then PC = ALUOut	PC = { PC[31:28], (IR[25:0]<<2) }	Execution or Branch/Jump Completion
3	Reg [IR[15:11]] = ALUOut	Load: MDR = Memory[ALUOut] Store: Memory [ALUOut] = B			Memory Access & R-type Completion
4		Load: Reg [IR[20:16]] = MDR			Memory Read Completion

Figure 2 – Example Execution Stages for The multi-cycle datapath

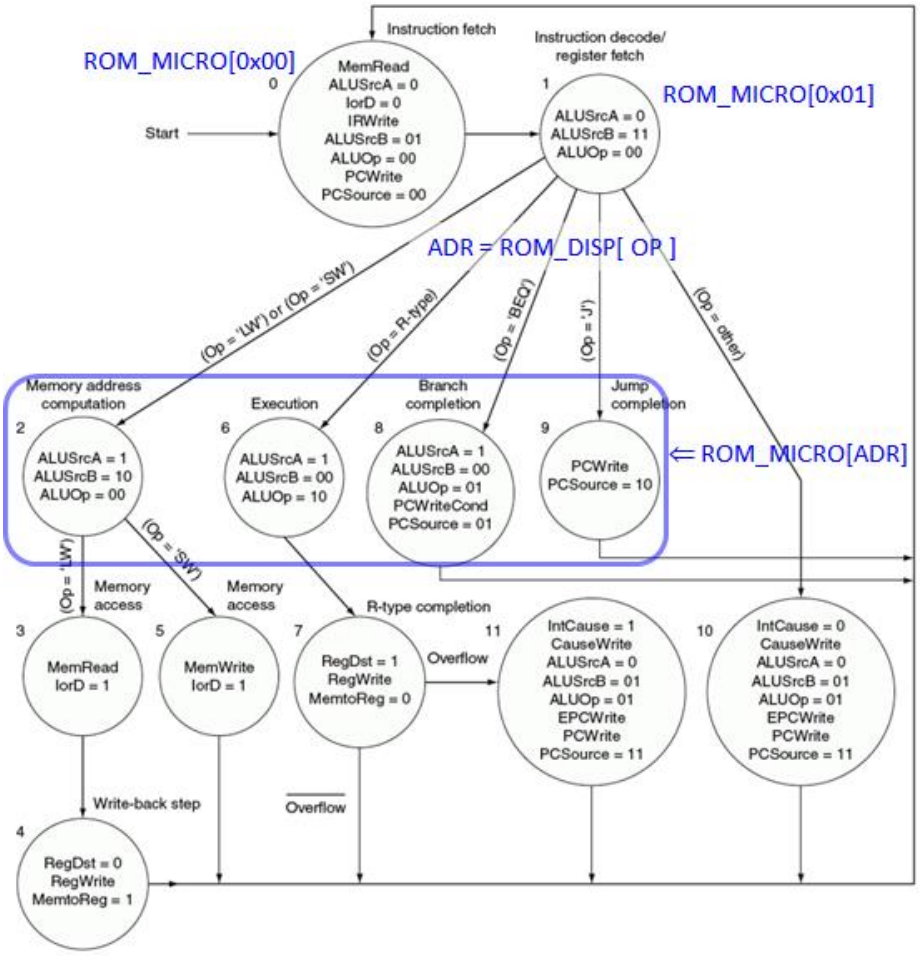

Figure 3 – An Example Finite State Machine for The multi-cycle datapath

Table 1 - Instance name of top module

Instance name	Description
U0_PC	Program counter
U1_MEM	Memory (IM+DM)
U2_IR	Instruction Register
U3_MDR	Memory data register
U4_FSM	Finite State Machine Main Controller
U5_RF	Register File
U6_A	Temporary register for Read data1
U7_B	Temporary register for Read data2
U8_SEU	Sign Extend Unit
U9_ALU	Arithmetic Logical Unit
U10_MUL	Multiplier Unit
U11_ALUO	Temporary register for ALU result

2. Assignment

The Multi-Cycle CPU for the project is implemented using the microprogramming technique. MIPS Instructions LW, SW, ORI, ADD, SUB, J, LUI, LHI, LLO are already implemented. You have to add the following MIPS Instructions:

XORI, SLT, SUBU, SRA, MULTU, MFLO, SB, LHU, BLEZ, JR

*** The format of the Instruction must follow the attached datasheet.**

*** Design both DISP_ROM (Dispatch ROM) & MICRO_ROM (microprogram ROM).**

*** The endianness is little endian.**

*** You can define and use any state number below 128 as you want except the number 0 & 1, which are already defined for you.**

*** Most ALU functions take 1 cycle while multiplication and division take 2 cycles.**

2.1 Finite State Machine - Control Unit Using Micro-Programming Technique

- Dispatch ROM filename: **ROM_DISP.txt**
- Micro-Program ROM filename: **ROM_MICRO.txt**

Table 2 – Consecutive I/O port configuration of the dispatch ROM: ROM_DISP

Port name	Classification	Bit	Description
undef	Input	1-bit	Undefined Instruction
raddr	Input	8-bit	Address for Microprogram ROM

Table 3 – Consecutive I/O port configuration of the microprogram ROM: ROM_MICRO

Port name	Classification	Bit	Description
IorD	Output	1-bit	Memory Access for Instruction or Data
MemRead	Output	1-bit	Memory Read Enable Signal
MemWrite	Output	1-bit	Memory Write Enable Signal
DataWidth	Output	3-bit	Memory Data Control signal
IRwrite	Output	1-bit	Instruction Register Write Enable signal
RegDst	Output	2-bit	Register Destination Selection
RegDatSel	Output	3-bit	Register Write Data Selection
RegWrite	Output	1-bit	Register Write Enable signal
ExtMode	Output	1-bit	Extender Control signal
ALUsrcA	Output	3-bit	ALU input A Selection
ALUsrcB	Output	3-bit	ALU input B Selection
ALUop	Output	5-bit	ALU Operation Control signal
ALUctrl	Output	2-bit	ALU Control signal
Branch	Output	3-bit	Branch Address Selection signal
PCsource	Output	2-bit	Next PC Selection
PCwrite	Output	1-bit	PC Write Enable signal
StateSel	Output	2-bit	Next FSM State Number Selection

*** There are an 8-bit reserved field between PCwrite and StateSel. Set them as xxxxxxxx.**

*** The simulator stops right after it executes the "break" instruction or when the cycle time reaches the number in the file "tb_cycle_limit.txt".**

*** The "mem_dump.txt" and "reg_dump.txt" show the data in 32bit word format which means first left-most byte has the highest address.**

3. 결과 Report (표지제외 최소 2장)

■ 문제의 해석 및 해결 방향

- 실험 내용에 대한 설명
  ex) 자기가 구현한 FSM, 동작 및 특징
- 문제점 해결 방향

ex) 기존 기능을 어떻게 이용하여 명령을 구현하였는지 등

■ 설계 의도와 방법

- 구현한 Multi Cycle CPU FSM Diagram
- Micro-Instruction 의 Field 구분 및 Field 용도
- Micro-Instruction 반복된 사용에 대한 분석
- 코드 주석 필수
- 시뮬레이션 결과와 예상 결과 비교 분석

4. 결과 Report Submission

■ Soft copy

- Due data: **5월 3일(수) 23:59 까지 (딜레이 받지 않음.)**
- 결과 Report(.docx or .hwp)와 프로젝트 폴더를 압축하여 U-campus에 upload.
- **압축 파일명 양식: 학번_이름_Project_2.zip**
  - **ex) 2099722000_홍길동_ Project_2.zip**