Lab# 12 THE SINGLE CYCLE DATAPATH

Instructor: I Putu Danu Raharja.

Objectives:
Learn how to implement instructions for a CPU.

Method:

Learn to implement the single cycle datapath for a subset of 16-bit MIPS-like processor.

Preparation:

Read the slides.

File To Use:

12.1 OVERVIEW:

Suppose we would like to design a simple 16-bit MIPS-like processor with seven 16-bit general-purpose registers: R1 through R7. R0 is hardwired to zero and cannot be written, so we are left with seven registers. There is also one special-purpose 16-bit register, which is the program counter (PC). All instructions are also 16 bits. There are three instruction formats, R-type, I-type, and J-type as shown below:

R-type format:
4-bit opcode (Op), 3-bit register numbers (Rs, Rt, and Rd), and 3-bit function field (funct)

Op(4)	Rs(3)	Rt(3)	Rd(3)	funct(3)

I-type format:
4-bit opcode (Op), 3-bit register number (Rs and Rt), and 6-bit immediate constant

Op(4)	Rs(3)	Rt(3)	Imm(6)

J-type format:
4-bit opcode (Op) and 12-bit immediate constant

Op(4)	Imm(12)

For R-type instructions, Rs and Rt specify the two source register numbers, and Rd specifies the destination register number. The function field can specify at most eight functions for a given opcode. We will reserve opcode 0 and opcode 1 for R-type instructions.

For I-type instructions, Rs specifies a source register number, and Rt can be a second source or a destination register number. The immediate constant is only 6 bits because of

the fixed size nature of the instruction. The size of the immediate constant is suitable for our uses. The 6-bit immediate constant is signed (and sign-extended) for all I-type instructions.

For J-type, a 12-bit immediate constant is used for instructions such as J (jump), JAL (jump-and-link), and LUI (load upper immediate) instructions.

Instruction Encoding:

Eight R-type instructions, six I-type instructions, and three J-type instructions are defined. These instructions, their meanings, and their encodings are shown below:

Instr	Meaning	Encoding				
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 001
NOR	Reg(Rd) = ~(Reg(Rs) | Reg(Rt))	Op = 0000	Rs	Rt	Rd	f = 010
XOR	Reg(Rd) = Reg(Rs) ^ Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 011
ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
SLT	Reg(Rd) = Reg(Rs) < Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 110
JR	Jump register: PC = Reg(Rs)	Op = 0000	Rs	000	000	f = 111
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 0100	Rs	Rt	Immediate^6	
SLTI	Reg(Rt) = Reg(Rs) < ext(im^6)	Op = 0110	Rs	Rt	Immediate^6	
LW	Reg(Rt) = Mem(Reg(Rs) + ext(im^6))	Op = 1000	Rs	Rt	Immediate^6	
SW	Mem(Reg(Rs) + ext(im^6)) = Reg(Rt)	Op = 1001	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 1010	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1011	Rs	Rt	Immediate^6	
J	PC = PC + 1 + ext(im^{12})	Op = 1100	Immediate^{12}			
JAL	R7 = PC + 1, PC = PC+1+ext(im^{12})	Op = 1101	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

12.2 LAB EXERCISE

Based on the above requirement, implement only the datapath.