CSE 140 Lab/HW#3 – Due: in 2 weeks (see below)

MIPS Decoder (80pts)

Form a group of two members and do the following tasks.

* Note that this programming assignment is closely related to the project so work with the team member who you want to do the project together.

Write a C program that decodes an input machine code. Your program should be able to decode the R, I, and J type instructions listed below. We will only test the following instructions. You can find these instructions from the first page of the MIPS Reference Data sheet (download from CatCourse).

MIPS Reference Data

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	$0 / 20_{hex}$
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		$0 / 21_{hex}$
And	and	R	R[rd] = R[rs] & R[rt]		$0 / 24_{hex}$
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		$0 / 08_{hex}$
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs] +SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs] +SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		$0 / 27_{hex}$
Or	or	R	R[rd] = R[rs] | R[rt]		$0 / 25_{hex}$
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		$0 / 2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	$0 / 2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		$0 / 00_{hex}$
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		$0 / 02_{hex}$
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	$0 / 22_{hex}$
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		$0 / 23_{hex}$

When the program is started, your program will print "**Enter an instruction:**" and receive one 32-bit machine code from user. Then, the program will decode the machine code and print the information (type and values of individual fields of the instruction format) of the instruction. The

followings are the example execution of the program. Your program must produce the following sample results in the exact same format.

Sample results (inputs are presented in blue bold font):

Enter an instruction:

00000001000010011000100000100000

Instruction Type: R
Operation: add
Rs: $8
Rt: $9
Rd: $17
Shamt: 0
Funct: 32 (or 0x20)

Enter an instruction:

00110100001001000000000000000000

Instruction Type : I
Operation: ori
Rs: $1
Rt: $4
Immediate: 0 (or 0x0)

Enter an instruction:

00001100000000100000000000000000

Instruction Type : J
Operation: jal
Immediate: 131072 (or 0x0020000)

As a group, discuss how to read and interpret the instructions from user. Once this step is done, split the task among your group so that each member will implement different instructions individually.

Assume that your program will always receive a machine code in correct formats. So, you do not need to implement invalid input handling function. All test cases will follow the correct machine code format.

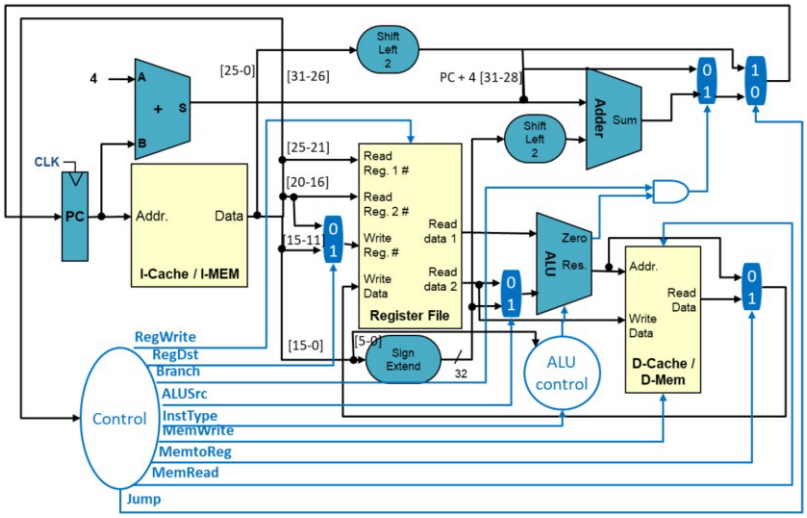
Demo : You will have two weeks to finish this lab. However, as an intermediate check, you need to show a demo to your TA one instruction per type (R, I, and J) at the next lab time. Therefore, your program needs to be able to decode at least one instruction per instruction type by next lab. And then you will complete the program and submit the final program by the deadline. When demoing and submitting your program, indicate which portion of the code was your implementation. "Working on the code together" means someone did not implement individually.

Single-cycle MIPS Architecture (20pts)

1. Assume that core components of single-cycle processor (shown below) have the following latencies:

I-Mem	Adder	Mux	ALU	Regs Rd/Wr	D-Mem	Sign-Extend	Shift-Left-2
30ps	40ps	20ps	60ps	15ps/15ps	200ps	20ps	20ps

Single-cycle processor data path:



Suppose that this data path executes only two types of instruction:

`add $rd, $rs, $rt`

The given instruction proceeds as follows:

1. The instruction is fetched from the I-Mem.
2. The input is given to the register file, which outputs read data 1 and read data 2
3. The outputs are given as input to the ALU via a mux.
4. The output of the ALU is written into the register file via a mux.

So `add $rd, $rs, $rt = I_mem+reg_read+ Mux+ALU + mux+ Reg_write`

`= 30ps + 15ps+20ps+60ps+20ps+15ps`

`Latency for this instruction = 160ps`

`sw  $rt, offset($rs)`

1. The instruction is fetched from the I-Mem.
2. The input is given to the register file, which outputs read data 1
3. The outputs are given as input to the ALU via a mux.
4. The output of the ALU is given as an address to the D-Mem (information is store in the cache)
5. Since sw just store information to cache, we don't need to do the (reg write +mux) because there are no register to write to.

```
= I_mem+reg_read+ Mux+ALU + D.mem

= 30ps + 15ps + 20ps+60ps+200ps

The latency of this instruction = 325ps
```

What would be the clock period to correctly execute the two instructions on the above single-cycle processor? Assume that PC register doesn't take any latency (i.e. Propagating a new PC value to I-Cache/I-Mem doesn't take any cycle).

2. We learned that the control unit feeds the components in the processor with control signals. The table on the page 5 of CSE140_Lecture-4_Processor-3 shows the summary of the control signals.
    a. In the table, there are a few 'X (don't care)' signals. Explain what they mean and why they do not need to have a solid value.
    There're few X signals that don't need to have a solid value in the lecture slide because the data/critical path will never reach/use those signals. Therefore, it's irrelevant data, thus the reason why they don't need a solid value.

    b. Explain what will happen if MemWrite is 1 for R-type instruction.
    The address that data memory receives will be store/write in the data memory/cache; instruction = mem[address]. The only operation that has Mem Write is sw that stores information into the cache for future. If all R-type instruction have MemWrite =1 and there's different multiple R-type instructions, it can overload the cache because it'll ran out of memory space.

Submission Guideline

- Submit your code under a name "first-member-name_second-member-name.c", and your solutions for the single-cycle datapath problems in a separate MS Word or a pdf format to the CatCourse.
- Deadline: **Before the lab in two weeks** (If this lab is assigned on 2/15 7:30AM, the deadline is 3/1 7:30AM)