ECE200 Computer Organization Lab4

Due Apr. 27th 11:59pm

Report

- Abstract (less than 100 words)
- Introduction (function description, design assumptions, etc.)
- Documentation (a list of source code files and descriptions of each of them, please also write a **readme file** to describe how to test/run your code)
- Verification (test all implemented MIPS instructions, for bonus2 show assembly code and test result)
- Comments (if any problems, difficulties, suggestions, etc.)

Basic Assignment (100 points)

Write Verilog code to implement a non-pipelined *single-cycle* 16-bit RISC processor (16-bit data and address, 32-bit instruction), which can support all the 18 instructions in Table 1. An illustrated datapath is shown in Figure 1.

	6	5	5	5	5	6
add	0x00	rs	rt	rd	0x00	0x20
addi	0x08	rs	rt	immediate		
sub	0x00	rs	rt	rd	0x00	0x22
and	0x00	rs	rt	rd	0x00	0x24
andi	0x0c	rs	rt	immediate		
nor	0x00	rs	rt	rd	0x00	0x27
or	0x00	rs	rt	rd	0x00	0x25
ori	0x0d	rs	rt	immediate		
lui*	0x0f	0x00	rt	immediate		
slt	0x00	rs	rt	rd	0x00	0x2a
slti	0x0a	rs	rt	immediate		
beq	0x04	rs	rt	offset		
bgez	0x01	rs	0x01	offset		
j	0x02	target				
lb	0x20	rs	rt	offset		
lh	0x21	rs	rt	offset		
sb	0x28	rs	rt	offset		
sh	0x29	rs	rt	offset		

Table 1: Instruction set for basic assignment.

*lui will be load immediate: directly write immediate to destination register

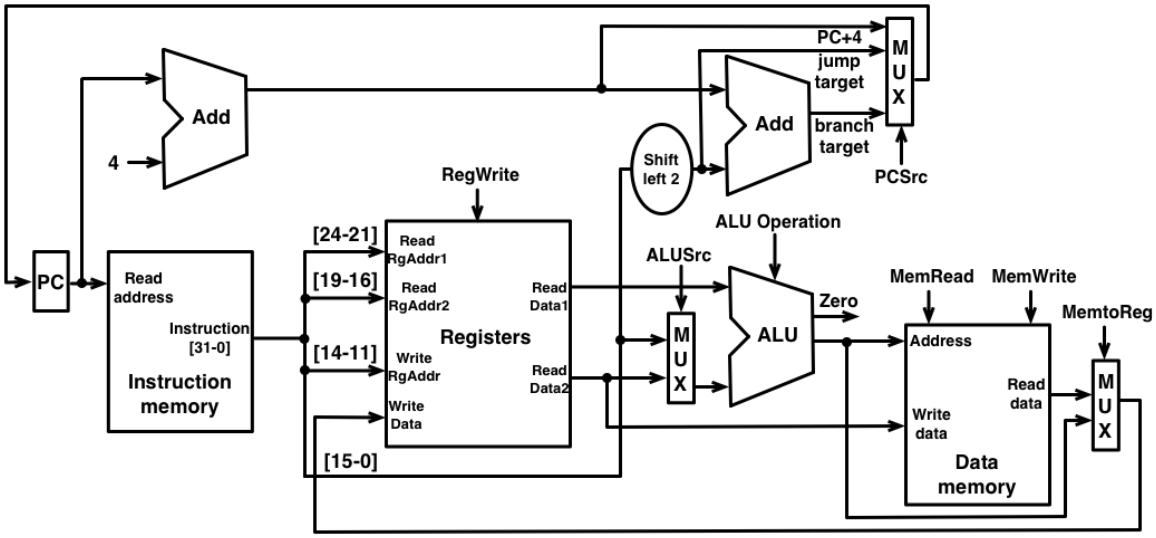

Figure 1: Illustration of a 16-bit RISC datapath.

Memory is byte addressable (8-bit per unit). Big-endian (specify address of the most significant byte) is recommended for byte ordering. Memory sizes are self-defined. Use $readmemh to initiate content in instruction memory.

MIPS reg name	Convert name	Number	Usage
$zero	$zero	0	Constant 0
$at	$at	1	Reserved for assembler
$v0	$v0	2	Expression evaluation and function results
$v1	$v1	3	
$a0	$a0	4	Argument 1 (caller saves)
$a1	$a1	5	Argument 2 (caller saves)
$a2	$a2	6	Argument 3 (caller saves)
$a3	$a3	7	Argument 4 (caller saves)
$t0	$t0	8	Temporary (caller saves)
$t1	$t1	9	Temporary (caller saves)
$t2	$s0	10	Temporary (callee saves)
$t3	$s1	11	Temporary (callee saves)
$t4	$gp	12	Global pointer
$t5	$sp	13	Stack pointer
$t6	$fp	14	Frame pointer
$t7	$ra	15	Return address (HW access)

Table 2: Registers usage converted from MIPS register convention.

Due to the limited number (16, where MIPS is 32 registers) of registers in this processor a conversion table is provided in Table 2 (might not be used for the basic assignment).

Simulate the design and to test each instruction. Submit source code and report through Black Board.

Bonus 1: optional instructions (30 points)

Edit the design to support the following instructions.

	6	5	5	5	5	6
div	0x00	rs	rt	0x00	0x00	0x1a
mult	0x00	rs	rt	0x00	0x00	0x18
mfhi	0x00	0x00	0x00	rd	0x00	0x10
mflo	0x00	0x00	0x00	rd	0x00	0x12
sll	0x00	0x00	rt	rd	shamt	0x00
srl	0x00	0x00	rt	rd	shamt	0x02
jr	0x00	rs	0x00	0x00	0x00	0x08
jal	0x03	target				

Hint: mult and div need Hi and Lo registers.

Bonus 2: code generation (30 points)

Prerequisite: jal and jr instruction need to be implemented.

Translate the following c code to assembly code using the converted register names (second column in Table 2) and then generate binary instructions. Initiate the instruction memory using generated instructions; and run the code on the processor you designed.

```
int fact (int n) {
        If (n<=1) return 1;
        else return (n * fact(n-1));
};
int a = fact(8);
```

Online assembler: http://alanhogan.com/asu/assembler.php

When use the online assembler, register names need to be converted back to MIPS register name. Check whether the jump target address and branch offset are desired.

Bonus 3: 5-stage pipeline (80 points)

Modify the design to a 5-stage pipeline.

Step 1: divide building blocks appropriately to each stage: IF, ID, EXE, MEM, and WB; and add pipeline registers as needed. Add logic to insert pipeline bubbles as needed. (30 points)

Step 2: add quick comparison and address computation for branch in decode stage. (20 points)

Step 3: add bypassing (forwarding) logic appropriately. (30 points)