CMSC611: Advanced Computer Architecture
Homework 2 Solution

Question 1: (25 points)

a) Using the conditional load LDZ Rd, x(Rsl), Rs2, the new code is

```
        LD      R1, 0(R2)
        stall   (RAW hazard on R1)
        DADDI   R4, R1, #8
        LDZ     R4, 0(R3), R1
        SD      R4, 0(R2)
```

One stall is needed to resolve the RAW hazard.

b) For the original code,

```
        LD      R1, 0(R2)
        stall   (RAW hazard on R1)
        stall   (RAW hazard on R1)
        BNEZ    R1, L1
        stall   (control hazard)
        LD      R1, 0(R3)
        J       L2
        stall   (control hazard)
L1:     DADDI   R1, R1, #8
L2:     SD      R1, 0(R2)
```

Four stalls are needed to resolve the RAW and control hazards.
Execute the original code in the pipeline: if the branch "BNEZ R1, L1" is taken, 11 clock cycles are needed; otherwise, 13 clock cycles are needed.
Execute the new code in the pipeline: 9 cycles are needed.
Since implementing the new conditional load increases the clock cycle by 10%, if the original clock cycle time is t ns, the new clock cycle time will be (1+10%)×t ns. We compute the speedup in two cases as:

$$\text{Speedup}_{\text{branch taken}} = \frac{\text{ExecutionTime}_{\text{old}}}{\text{ExecutionTime}_{\text{new}}} = \frac{\text{ClockCycles}_{\text{old}} \times \text{ClockCycleTime}_{\text{old}}}{\text{ClockCycles}_{\text{new}} \times \text{ClockCycleTime}_{\text{new}}} = \frac{11t}{9\times(1+10\%)t} \approx 1.11$$

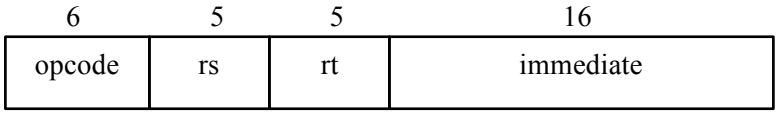
$$\text{Speedup}_{\text{branch not taken}} = \frac{\text{ExecutionTime}_{\text{old}}}{\text{ExecutionTime}_{\text{new}}} = \frac{\text{ClockCycles}_{\text{old}} \times \text{ClockCycleTime}_{\text{old}}}{\text{ClockCycles}_{\text{new}} \times \text{ClockCycleTime}_{\text{new}}} = \frac{13t}{9\times(1+10\%)t} \approx 1.31$$

Therefore, it is worth making the change because the new code is faster than the original code by 1.11x when the branch is taken in the original code and 1.31x when the branch is not taken in the original code.

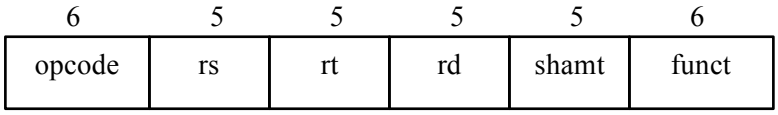
Question 2: (15 points)

a) For the instruction set architecture of MIPS supporting 32 registers, we know that

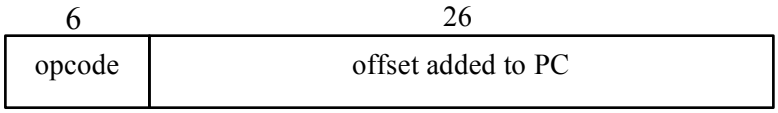
I-type instruction:



R-type instruction:



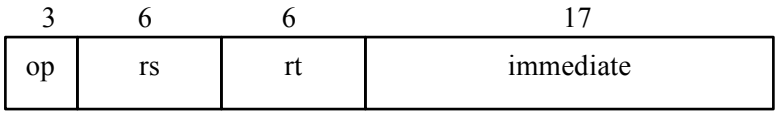
J-type instruction:



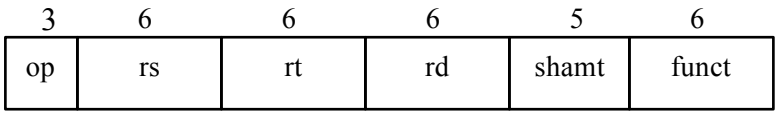
If we want to change the number of registers from 32 to 64, we need 6 bits (2^6=64) to assign each register field. Therefore, we need 2×6=12bits (2 register fields) for I-type instruction, 3×6=18bits (3 register fields) for R-type instruction, and 0bits (0 register fields) for J-type instructions totally.

b) The new bit layouts for different instruction types are

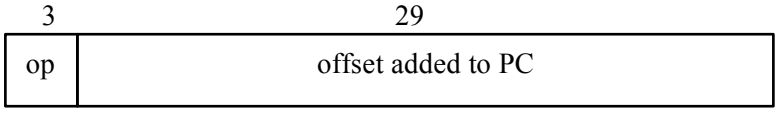
I-type instruction:



R-type instruction:



J-type instruction:



This new instruction set architecture can only support 2^3=8 opcodes.

Question 3: (30 points)

a) See "Original code" on the attached pipeline chart for details.

b) See "Schedule code 1" and "Schedule code 2" on the attached pipeline chart for details. Both schedule codes are acceptable as correct answers. Since R1 can only be changed by the ADDI instruction, R1 is accumulated by 8 in each iteration of Loop. With the given initial values of R1 and R8, Loop will keep iterating until BNE is false. It indicates that the total number of iterations of Loop is 40 / 8 = 5.
For the original code, 5 iterations of Loop take 4×13+16=68 clock cycles. For the schedule code 1, 5 iterations of Loop take 4×9+12=48 clock cycles. For the schedule code 2, 5 iterations of Loop take 4×10+13=53 clock cycles. Either the schedule code 1 or 2 is better than the original code in clock cycles. Notice that the BNE instruction is resolved in the ID stage. Because of the control hazard and there is no delayed branching, the instruction following BNE only starts from a new clock cycle right after the ID stage of BNE.

Question 4: (30 points)

a) The pipeline chart with a 6-stage pipeline for 6 instructions is:

IF	RF	ALU1	MEM	ALU2	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	ALU1	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

b) Adders are needed for three cases:
1) Increment program counter for new instructions at IF stage.
2) Calculate the effective memory address or branch at ALU1 stage.
3) Execute ALU operation at ALU2 stage.

Three adders are needed maximally in this case:

IF	RF	ALU1	MEM	**ALU2**	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	**ALU1**	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

c) Register read and write ports are needed for two cases:
1) Read register at RF stage.
2) Write register at WB stage.

Memory read and write ports are needed for three cases:
1) Read instruction from memory at IF stage.
2) Read data from memory at MEM stage.

3) Write data to memory at MEM stage.

One register write port, one memory read/write port (if the third instruction is Load/Store), and one register read port are needed in the following case:

IF	RF	ALU1	MEM	ALU2	**WB**					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	RF	ALU1	**MEM**	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	**RF**	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB

Two memory read ports and one register read port are needed in the following case, if the first instruction is Load. Or, one memory write port, one register read port and one memory read port are needed in the following case, if the first instruction is Store.

IF	RF	ALU1	**MEM**	ALU2	WB					
	IF	RF	ALU1	MEM	ALU2	WB				
		IF	**RF**	ALU1	MEM	ALU2	WB			
			IF	RF	ALU1	MEM	ALU2	WB		
				IF	RF	ALU1	MEM	ALU2	WB	
					IF	RF	ALU1	MEM	ALU2	WB