Question 1: Single-cycle CPU implementation (20 points)

You are asked to implement **jr rs** instruction (jump to register rs) in the single-cycle datapath. The format of **jr** instruction is shown below.

Field	op = 0	rs	0	func = 8
Bits	31-26	25-21	20-6	5-0

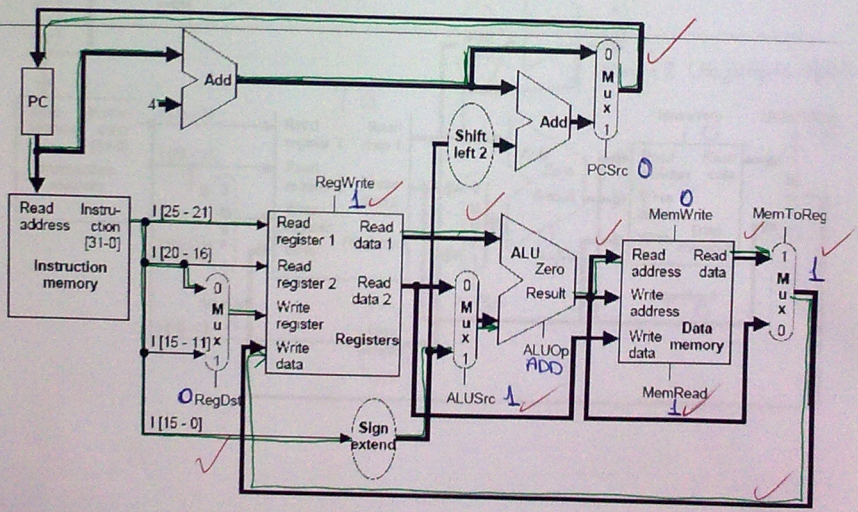
"Before implementing the **jr** instruction, do parts (a) and (b) as a small warm-up exercise."

Part (a)

The single-cycle datapath from lecture appears below. Clearly mark all wires that are active during the execution of **lw instruction**. (10 points) marked with green.

Part (b)

On the diagram below, write (next to the signal's name) values of **all** non-0 control signals required for the **lw** instruction. (5 points)



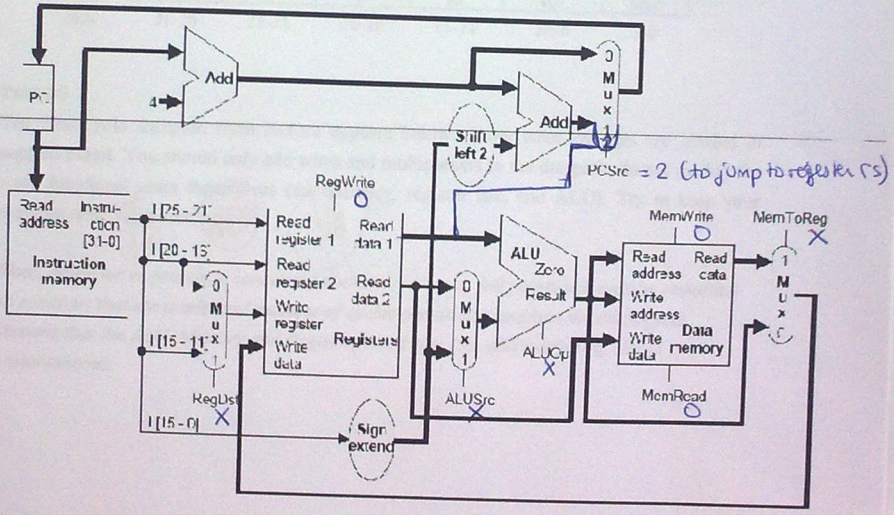
Part (c)

The single-cycle datapath from lecture (same as on the previous page) appears below. Show what changes are needed to support **jr** instruction. You should only add wires and multiplexers to the datapath; do not modify the main functional units themselves (the memory, register file and ALU). Try to keep your diagram neat! (10 points)

Note: While we're primarily concerned about correctness, full points will only be rewarded to solutions that do not lengthen the clock cycle. Assume that the ALU, Memory and Register file all take 2ns, and everything else is instantaneous.

Part (d)

On the diagram below, write (next to the signal's name) values of **all** non-0 control signals required for the **jr** instruction. (5 points)



Question 2: Multi-cycle CPU implementation and its performance (20 points)

Assume that the ALU can perform the max2 operation (*i.e.*, return the greater of 2 inputs):

```
alu_result = (A_input > B_input) ? A_input : B_input;
```

ALUOp for this instruction is MAX2.

Given this improved ALU, implement the **max4** instruction that writes into register rd the largest value of 4 registers:

```
max4 rs, rt, rd, rm # rd = max(rs, rt, rd, rm)
```

Note that register **rd** is both an *input* and an *output*. Instruction **max4** has the following format:

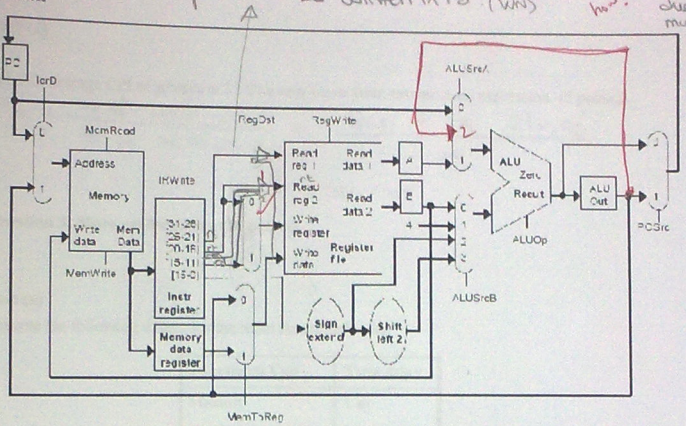
Field	op	rs	rt	rd	rm	func
Bits	31-26	25-21	20-16	15-11	10-6	5-0

Part (a)

The multicycle datapath from lecture appears below. Show what changes are needed to support **max4**. You should only add wires and multiplexers to the datapath; do not modify the main functional units themselves (the memory, register file, and ALU). Try to keep your diagram neat! (10 points)

*Note: While we're primarily concerned about correctness, full points will only be rewarded to solutions that use a **minimal number of cycles** and do not lengthen the clock cycle. Assume that the ALU, Memory and Registerfile all take 2ns, and everything else is instantaneous.*

rs & rt are 1st compared, their result goes back to be saved in rs & it will be compared with rm, their result saved in rs will now be compared with rd (found in rb) due to multiplexer. The max of 4 will be written in rd (WB)



Part (b)

The **max4** instruction can be used in place of three branch instructions, reducing the number of instructions that need to be executed. Below are two functionally equivalent programs; the second of which uses the **max4** instruction:

a0 a1 v0 t1 t2
rt rm rs 0(rm)

	Program 1		**Program 2**
	lw v0, 0(a0)	a0 → rs	
	add a1, a0, 12	a0+4 → rt	lw t0, 4(a0)
label:	add a0, a0, 4	a0+8 → rd	lw t1, 8(a0)
	lw t1, 0(a0)	a0+12 → rm	lw t2, 12(a0)
	slt t2, t1, v0		max4 v0, t0, t1, t2
	bne t2, $zero, skip		jr $ra
	move v0, t1	always taken	lw v0, 0(a0)
skip:	bne a0, a1, label		
	jr $ra		

7+7+7+7

Assume the datapath and control that you implemented in parts (a) and (b); *assume* **slt** *and* **move** *can be considered as R-type instructions and* **jr** *and* **bne** *take the same amount of time as* **beq**. Also assume that the first element in the input array is the largest (so that the first branch of program 1 is always taken). How much faster (or slower) is program 2 than program 1? You may leave your answer as a fraction. (15 points) (7 points)

For program 1:
total # of instrs: 7+7+7+7+1 = 29
assume 5 cc / inst.
cc. time = 1sec
⇒ Exec time = 29 × 5 = 145

For program 2:
6 instructions, assume 5 c.c. / instruction
and c.c. time = 1sec.
⇒ Exec. time = 6 × 5 × 1 = 30 sec.

ratio = Exec time(1) / Exec time(2) = 145/30 ⇒ (2) is faster by 80/145 than (1) ⇒ 2 is of higher performance

Part (c)

What is average CPI of **program 1**? You may leave your answer as an expression. (5 points)

CPI = # of clock cycles / # of instructions = 5cc. x 29 instr. / 29

= 5cc./instr.

Question 3: More performance (20 points)

18

Part (a)

Assume the following delays for the main functional units:

Functional Unit	Time delay
Memory	5 ns
ALU	4 ns
Register File	3 ns

Given the following instructions: **lw**, **sw**, **add**, **beq,** calculate:

- minimum time to perform each instruction
- time required on a single-cycle datapath (from q. 1)
- time required on multi-cycle datapath (from q. 2).

8

Write your answers in the table below. State any assumptions. (10 points)

same length for each clock cycle ⇒ 5

Time to perform the instruction:

Instruction	minimum time	in single-cycle	in multi-cycle	
lw	20ns.	5+3+4+5+3 = 20 ns	5+3+4+5+3 = 20ns.	5*5
sw	17ns	5+3+4+5 = 17ns.	5+3+4+5 = 17ns	5*4
add	15ns	5+3+4+3 = 15 ns	5+3+4+3 = 15ns	5*3
beq	12ns.	5+3+4 = 12ns	12ns	

all take as long as the longest ⇒ 20

Part (b)

How much faster than a 1GHz single-cycle MIPS processor would a 3.0Ghz Pentium4 x86 processor be if it achieved a CPI of 0.5 on a workload? (6 points)

assume same nr. of instrs. I.

$$\frac{\text{Performance of single}}{\text{Perf of Pentium}} = \frac{\text{Exec of Pent}}{\text{Exec. of single}} = \frac{0.5 \times I \times \frac{1}{3\,GHz}}{1(I) \times \frac{1}{1}} = \frac{0.5}{3 \times 10^9\ Hz}$$

$\Rightarrow$ Pentium 4 is of better performance

Question 4: Pipelining and forwarding (20 points)

Part (a)

Show or list all of the dependencies in this program. For each dependency, indicate which instructions *and* register are involved. (6 points)

```
add $8, $5, $5
add $2, $5, $8
sub $3, $8, $4
add $2, $2, $3
```

105 + 105 = 210 → $8

$8 210 + $5 105 = 315 → $2

$8 210 + $4 104 = 314 → $3

$3 314 + $2 315 = 629 → $2

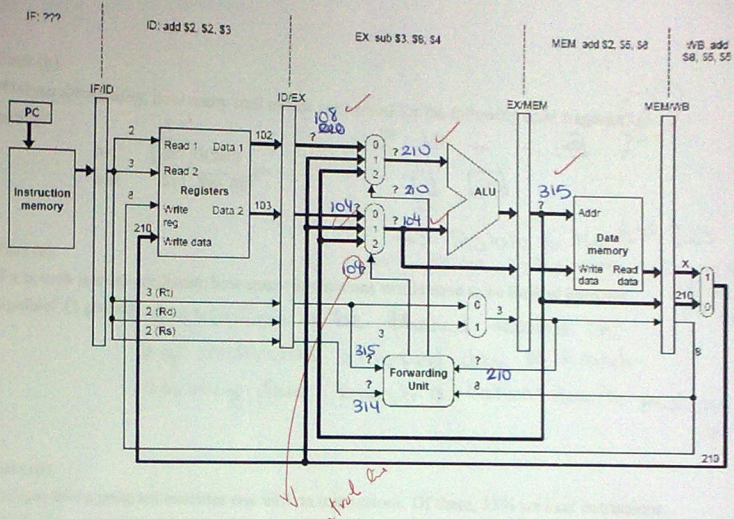
Part (b)

The pipelined datapath on the next page shows the fifth cycle of executing this program, including values for several of the stages. Fill in the ten remaining values, marked with a **?** symbol, in the EX and MEM stages. ~~(20 points; 2 points each)~~

N.B.:

- Write your answers directly on the diagram, but write clearly.
- Show decimal values.
- Assume that registers initially contain their number plus 100: $2 contains 102, $8 contains 108, etc.
- Write 'X' for any numbers that cannot be determined.

Note! Assume forwarding is done!



Question 5: Pipelining performance (20 points)

IPG	FET	ROT	EXP	REN	WLD	REG	EXE	DET	WRB
1	2	3	4	5	6	7	8	9	10

17

One CPU manufacturer has proposed the 10-stage pipeline above for a 500MHz (2ns clock cycle) machine. Here are the correspondences between this and the MIPS pipeline:

- Instructions are fetched in the FET stage.
- Register reading is performed in the REG stage.
- ALU operations *and* memory accesses are both done in the EXE stage.
- Branches are resolved in the DET stage.
- WRB is the writeback stage.

Part (a)

How much time is required to execute one million instructions on this processor, assuming there are no dependencies or branches in the code? (5 points)

Execution time = ~~(1,000,000 × 10) + (999,999 ×~~

= # of C.Cs × C.C. time

= # of inst. × CPI × C.C. time

= 1000,000 × 10 × 2ns

= 20,009,000 ns. ≈ 2,000,018 ns.

Part (b)

Without forwarding, how many stall cycles are needed for the following code fragment? (5 points)

```
lw    $t0, 0($a0)
add   $v1, $t0, $t0
```

Part (c)

If a branch is miss-predicted, how many instructions would have to be flushed from the pipeline? (5 points)

Part (d)

Assume that a program executes one million instructions. Of these, 15% are load instructions which stall, and 10% of the instructions are branches. The CPU predicts branches correctly 75% of the time. How much time will it take to execute this program? (10 points)

Performance

1. Formula for computing the CPU time of a program P running on a machine X:

$$CPU\ time_{X,P} = Number\ of\ instructions\ executed_P \times CPI_{X,P} \times Clock\ cycle\ time_X$$

2. CPI is the average number of clock cycles per instruction:

$$CPI = Number\ of\ cycles\ needed \div Number\ of\ instructions\ executed$$

3. Speedup is a metric for relative performance of 2 executions:

$$Speedup = Performance\ after\ improvement \div Performance\ before\ improvement$$
$$= Execution\ time\ before\ improvement \div Execution\ time\ after\ improvement$$