A7(a) (30 marks + 20 bonus marks)

Focus: datapath design, performance, ALU multiplication

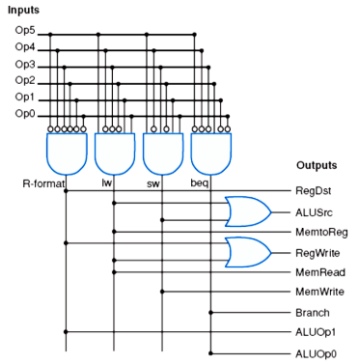
Q1. Assume we want to support the instruction `addi` in the design of the single-cycle datapath discussed in class. What changes need to be made to the control unit design in order to support that instruction? Justify your answers and draw the new control unit design. Assume ALUOp = 00. You can copy the main control unit circuit diagram and truth table from the lecture notes and apply any changes to them. [15 marks]

Where to start?

a) Identify the proper control signal (e.g. RegDst, RegWrt, etc).
b) Update the truth table used for designing the CU by adding an extra entry for addi. i.e. update this truth table:

	inputs I[31-25]						Outputs								
Instruction type	op5	op4	op3	op2	op1	op0	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-format	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0
lw	1	0	0	0	1	1	0	1	1	1	1	0	0	0	0
sw	1	0	1	0	1	1	x	1	x	0	0	1	0	0	0
beq	0	0	0	1	0	0	x	0	x	0	0	0	1	0	1

c) Update the circuit design below for the CU by considering the extra entry from step (b) above.



Q2. Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs given in the following table. [15 marks]

Processor	Clock Rate	CPI
P1	3 GHz	1.5
P2	2.5 GHz	1.0
P3	4 GHz	2.2

Answer the following and justify your answers (show steps).

i) Which processor has the highest performance expressed in “instructions-per-second”?
ii) If the processors can execute a program in 10 seconds, find the number of cycles and the number of instructions for each processor.
iii) We are trying to reduce the time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

Q3. [**Bonus: 20 marks**] Write a MIPS program that calculates the product of two unsinged integers. In your implementation, you are **not** allowed to use the MIPS instructions `mul` or `mult` for multiplication. Instead, implement the "third version of the multiplication algorithm discussed in class. Use two 32-bit general registers for storing the ’64-bit product’, e.g., `$s1` for product’s left half and `$s0 for right half`. Use the attached starter code for your program. Assume any values for the two numbers for testing your code. You don’t need to print the product on the screen.

Hints:

- To logically shift-right the 64-bit product represented by (`$s1,$s0`), you need to: (1) shift-right the right-half `$s0` by 1 bit, (2) copy the least-significant-bit (LSB) of $s1 to the MSB of `$s0`, and then (3) shift-right the left-half `$s1` by 1 bit.
- To read, set, or reset a single bit in a register, you may use `andi` and `ori` instructions.

Submission Instructions: Compress all your files (including the assembly files) into one zip file and submit it to Canvas. You can resubmit an assignment, but the new submission overwrites the old one and receives a new timestamp.