Question 1 - Sum-of-Products

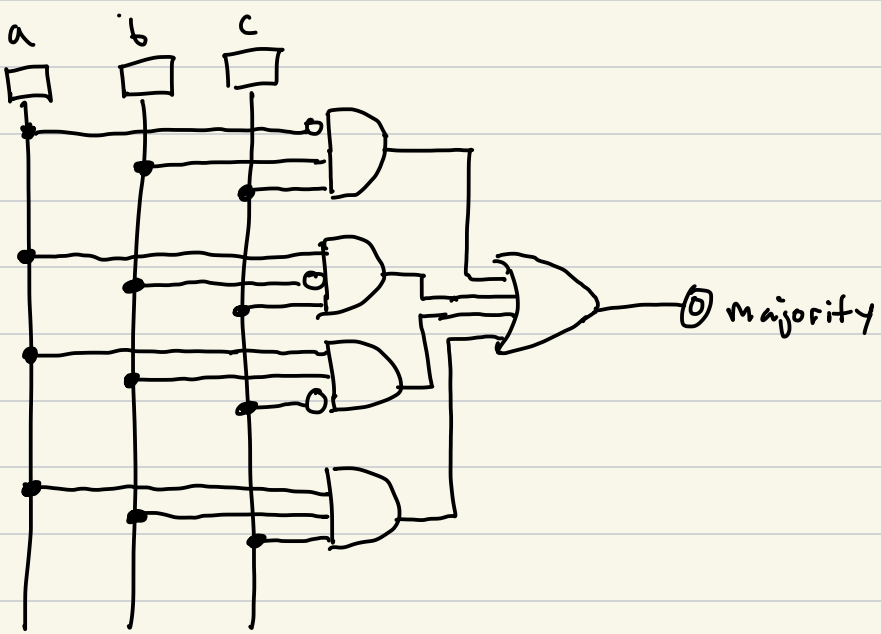
Consider implementing the Majority function of 3 1-bit inputs, `r = majority(a, b, c)`. The the value of the Majority function is true if and only if the majority (at least two) inputs are true. For example:

```
majority(1, 0, 1) = 1
majority(0, 1, 0) = 0
```

Build a truth table for this function. Next, derive the sum-of-products boolean algebra equation for this function. Finally, draw a circuit that implements this function in terms of AND, OR, and NOT gates.

a	b	c	majority
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$\text{majority}(a,b,c) = (\bar{a} \cdot b \cdot c) + (a \cdot \bar{b} \cdot c) + (a \cdot b \cdot \bar{c}) + (a \cdot b \cdot c)$$

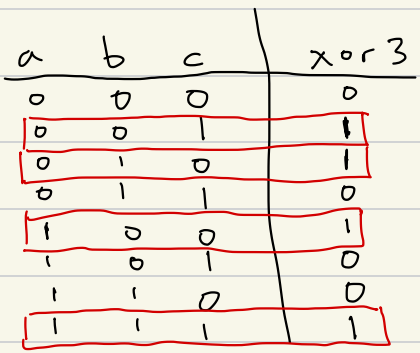


Question 2 - Sum-of-Products (Variation)

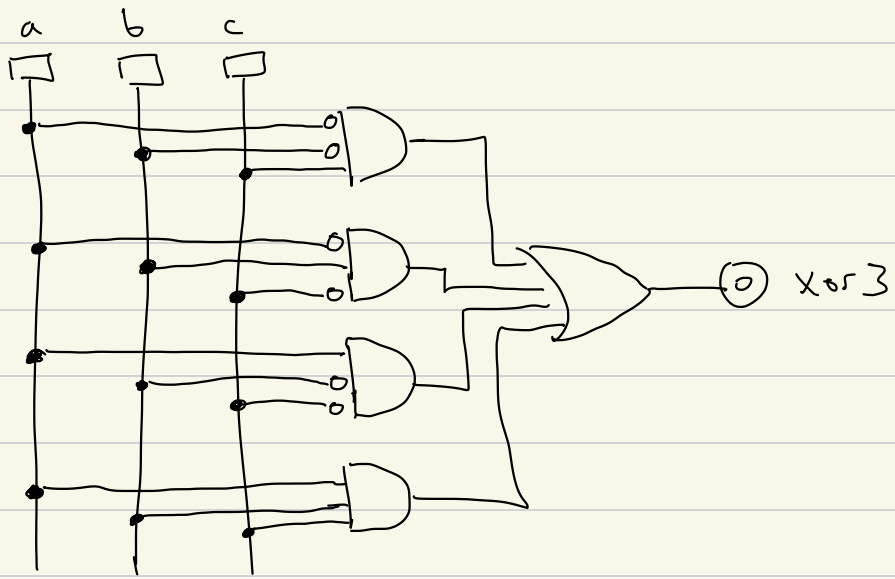
Consider implementing the 3-input XOR function, `r = xor3(a, b, c)`. The XOR function outputs true if an odd number of inputs are true. For example:

```
xor3(1, 0, 1) = 0   (even number of 1s: two)
xor3(1, 1, 1) = 1   (odd number of 1s: three)
xor3(0, 1, 0) = 1   (odd number of 1s: one)
xor3(1, 1, 0) = 0   (even number of 1s: two)
```

Build a truth table for this function. Next, derive the sum-of-products boolean algebra equation for this function. Finally, draw a circuit that implements this function in terms of AND, OR, and NOT gates.

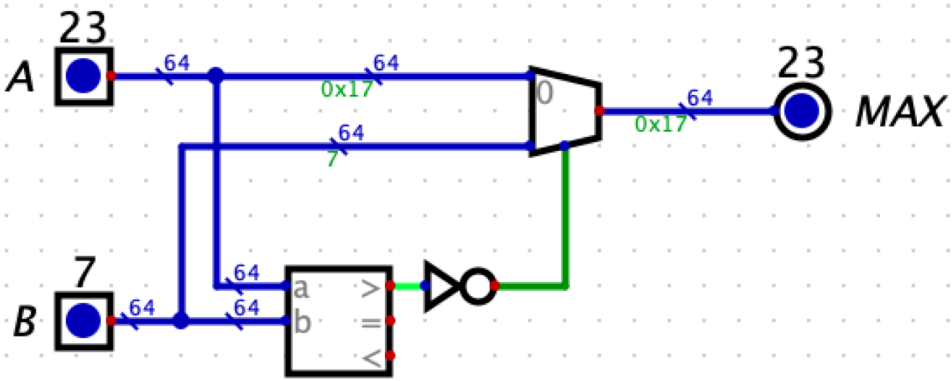


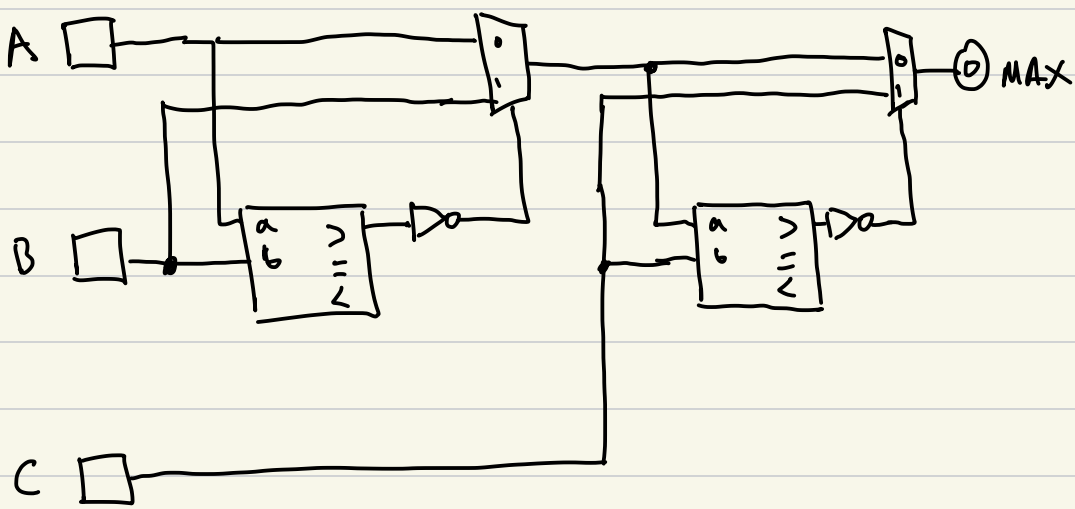
$xor3 = (\bar{a} \cdot \bar{b} \cdot c) + (\bar{a} \cdot b \cdot \bar{c}) + (a \cdot \bar{b} \cdot \bar{c}) + (a \cdot b \cdot c)$



Question 3 - Digital Design

Consider the following circuit called Max2, that determines the maximum of two 64-bit inputs. That is, the circuit take inputs A and B and the output is A if A > B, otherwise the output is B.





This circuit first selects the maximum value of a and b, call it max(a,b) then it selects the maximum value of max(a,b) and c

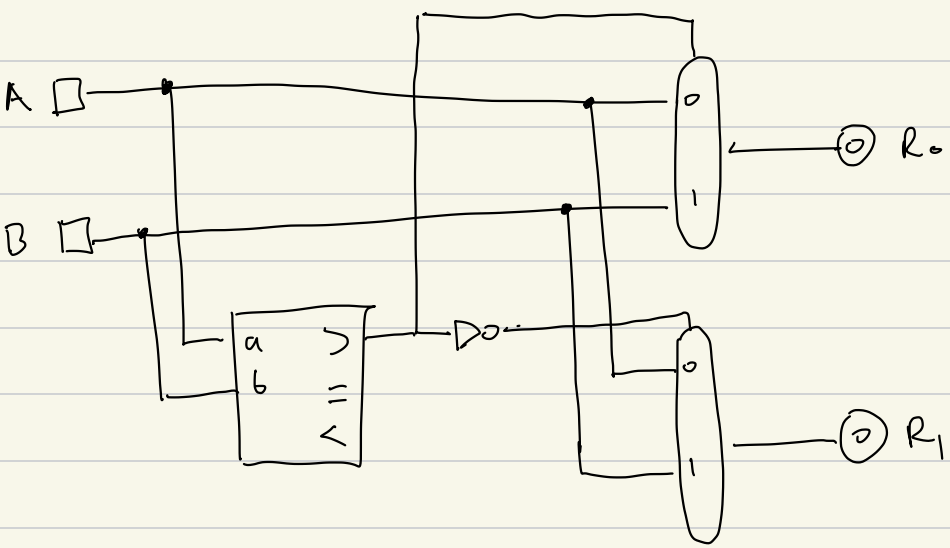
Question 4 - Digital Design (Variation)

Consider the Max2 circuit from Question 3, which determines the maximum of two 64-bit inputs using a comparator and multiplexor.

Build a new circuit called **Sort2** that takes two 64-bit inputs A and B and produces two outputs: OUT1 and OUT2, where OUT1 contains the smaller value and OUT2 contains the larger value. For example:

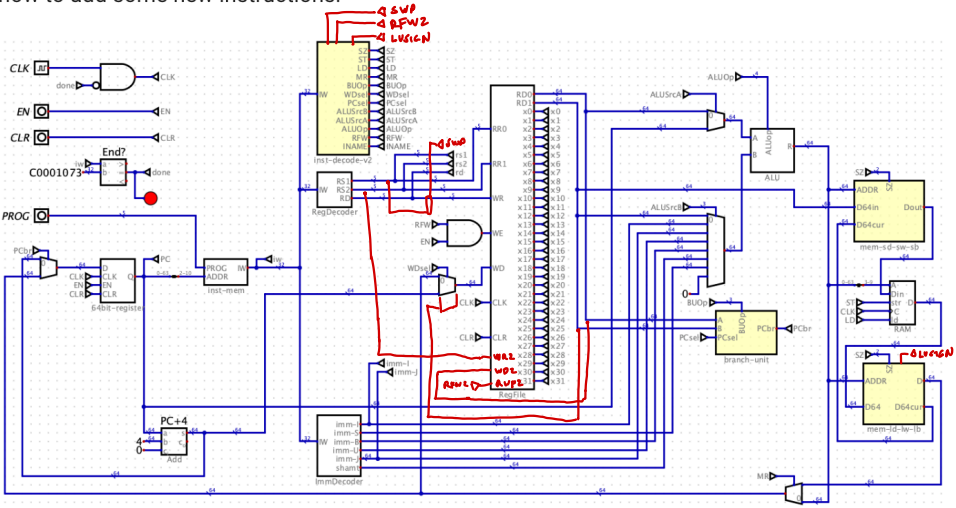
- If inputs are A=5, B=12, then OUT1=5 and OUT2=12
- If inputs are A=20, B=8, then OUT1=8 and OUT2=20
- If inputs are A=7, B=7, then OUT1=7 and OUT2=7

You can use the library circuits from above (comparators that output 1 if A > B, and multiplexors) and any additional gates you may need. Give a brief explanation in English how your circuit works. Draw neatly.



Question 5 - Single Cycle Processor

Consider the following version of the RISC-V single-cycle processor you developed for Project06. For this problem you are going to answer some questions about the implementation then explain how to add some new instructions.



Answer the following questions below. Justify your answers. Answers without justification will receive 0.

- (1) What is the maximum number of instructions a single program can have in this processor?
- (2) How many registers can be updated in a single clock cycle?
- (3) Which instruction uses the second input to the ALUSrcA MUX?
- (4) What is the purpose of the SZ input to `mem-sd-sw-sb` and `mem-ld-lw-lb`?

(1) 512 instructions. The address splitter is 2-10, which is 9 bits. $2^9 = 512$.

(2) Only one register can be updated in a single clock cycle because the register file only has one WD input, one WR input, and one RFW input. Note that the PC can be updated separately from the one register that can be updated in the register file.

(3) JAL and the conditional branches (BEQ, BNE, BLT, BGE) all use the second input to the ALUSrcA MUX, which is the current PC. This is used to calculate the target address.

(4) SZ is used to determine if we are loading or storing a double (0), a word (1) or a byte (2).

Now you are going to explain how to add new instructions to this processor. For each instruction, describe any additions or changes needed to the data path, the control path, and components. Also, explain changes needed in the instruction decoder. Completeness and details in your answer are important. You should assume that each of these instructions are currently not supported by this processor.

- (5) LWU (load word unsigned) This instruction loads a word (32 bits) from memory and puts the value into a 64 bit register. However, the word is not sign extended, like with LW (load word). Example usage: `lwu t0, (a0)`
- (6) SWPR (swap registers). This new instruction swaps two register values. Without this instruction, to swap register values, say `a0` and `a1` you need to do the following:

```
mv t0, a0
mv a0, a1
mv a1, t0
```

With SWPR, you can just do `swpr a0, a1`

(5) For LWU we need a new control line to send to the load component (mem-ld-lw-16) to select between signed loads and unsigned loads. Call it LUSIGN. This needs to be added to the decoder spreadsheet and the instruction decoder, and as an input to the load component. It will control a MUX that picks between sign extend (0) and zero extend (1). See LUSIGN above.

(6) For SWPR, we need to modify the register file so that two registers can be updated on a single clock cycle. This means the register file needs 3 new inputs: WD2, WR2, RFW2. The register file will need to be modified to allow two registers to updated at the same time with these new control lines. They need to be added to the decoder spreadsheet and instruction decoder.

Now, for SWPR, we need to read rs1 on RD0 and rs2 on RD1. The new SWP control line selects writing to rs1 instead of rd. Whereas we connect RD0 to WD2, and RD1 to the WDsel MUX, which will need to be expanded. See the modifications to the two level circuit above.

Question 6 - Single Cycle Processor (Variation) ¶

Consider the RISC-V single-cycle processor diagram from Question 5. Answer the following questions below. Justify your answers. Answers without justification will receive 0.

- (1) What is the maximum size (in bytes) of data memory that this processor can address?
- (2) What determines the minimum clock cycle time for this processor?
- (3) Which instruction(s) use the output of the Data Memory component?
- (4) Why does the processor need both an ALU and an Adder (the one that computes PC+4)?

(1) We can determine the maximum size of data memory (RAM) from the address bits and the data bits. We know the data bits are 64, or 8 bytes. The address bits are 3-9, or 7 bits. $2^7 \times 8 = 2^7 \times 2^3 = 2^{10} = 1024$ bytes.

(2) The cycle time is determined by the longest combinational path in the circuit. (I won't ask this).

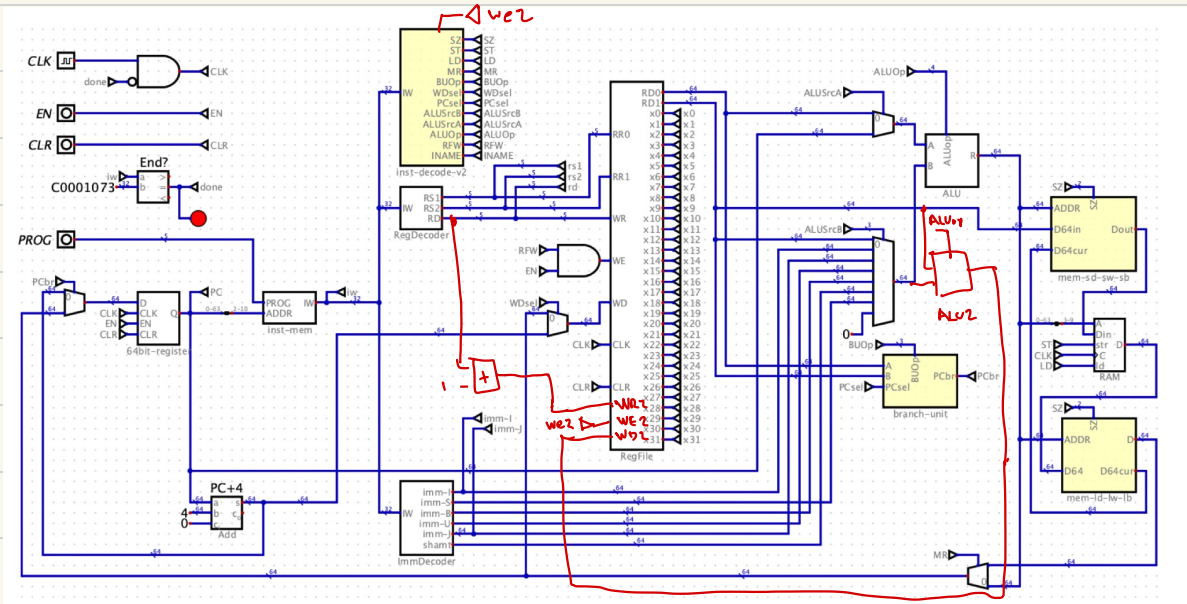
(3) The load instructions: lb, lw, ld and sb, sw for read-modify-write.

(4) Because we may need to do two additions in the same clock cycle: PC+4 and an ALU add for things like BTA calculations.

Now you are going to explain how to add new instructions to this processor. For each instruction, describe any additions or changes needed to the data path, the control path, and components. Also, explain changes needed in the instruction decoder. Completeness and details in your answer are important. You should assume that each of these instructions are currently not supported by this processor.

- (5) **LBU (load byte unsigned)**: This instruction loads a single byte (8 bits) from memory and puts the value into a 64-bit register. However, the byte is zero-extended (not sign-extended) like with LB (load byte). The upper 56 bits are filled with zeros. Example usage: `lbu t0, 0(a0)`
- (6) **ADDI2 (add immediate to 2 registers)**: This new instruction adds the same immediate value to two consecutive registers in a single instruction. For example, `addi2 a0, a0, 8` would simultaneously compute:
- `a0 = a0 + 8`
- `a1 = a1 + 8`

This instruction is useful for efficiently updating multiple loop counters or array pointers.

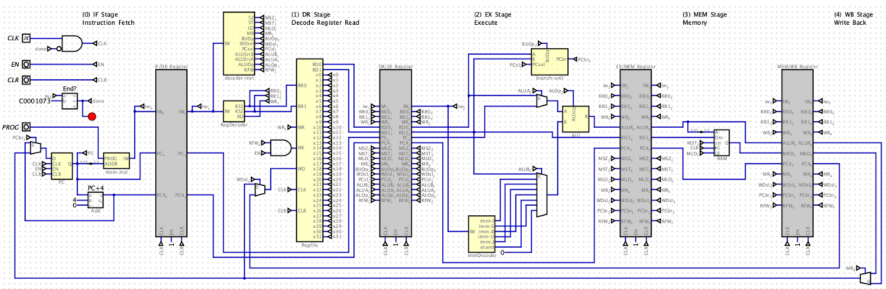


(5) lbu is similar to lwu. (see above)

(6) addi2 is similar to swpr in that we need to write to two resisters on the same clock cycle. However, in addition we need a second ALU because we need to do two adds on the same clock cycle. We also need an adder to compute a1 from a0.

Question 7 - Pipelined Processor

Consider the project07 starter pipelined processor below.



Answer the following questions and provide justification for answers. Answers without justification will receive 0.

Consider running the following code on the starter pipeline processor.

```
addi a1, zero, 3
addi a2, zero, 4
add a0, a1, a2
unimp
```

- (1) Explain why the code above produces the wrong result on the starter pipeline processor.
- (2) What result does this give? That is, what value will be in `a0` after executing this code?
- (3) Without modifying the processor implementation, how could you modify this code to make it run correctly?

(1) When the add instruction reaches the EX stage RD0 and RD1 will not have the correct values because the two addi instructions will not have completed their WB stages.

(2) On our processor, a0 will be 0 because all register values are initialized to 0 at startup.

(3) We can add nops:

```
addi a1, zero, 3
addi a2, zero, 4
nop
nop
nop
add a0, a1, a2
unimp
```

Now consider running this code on a complete solution to project07. That is a version of this pipeline with a proper hazard unit that supports forwarding, stalling and flushing.

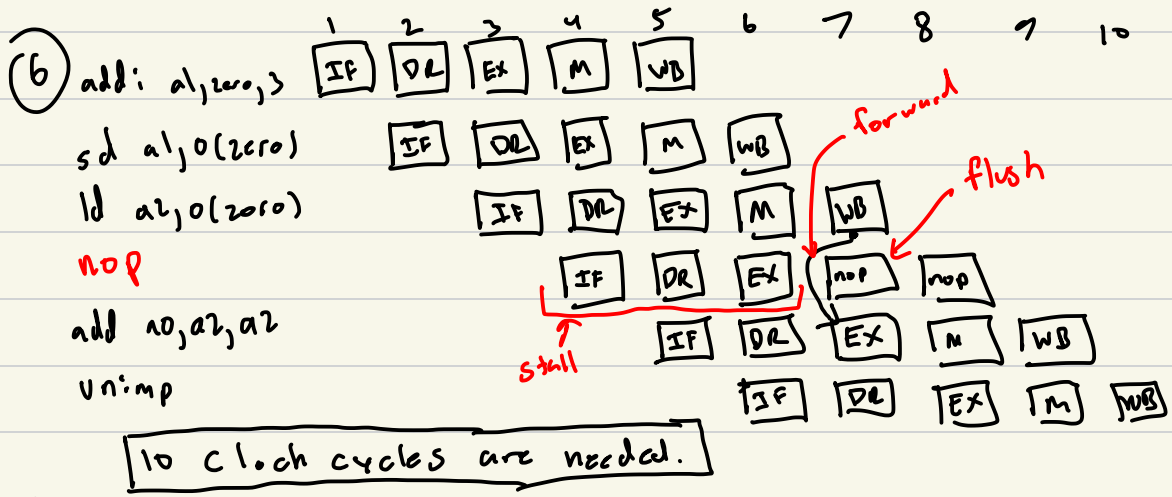
- (4) How many clock cycles will it take to complete the code given above? 8 (see below)
- (5) When the code executes, are any flushes or stalls needed? No, just forwarding is needed.

Now consider the following code on a complete solution to project07:

```
addi a1, zero, 3
sd a1, 0(zero)
ld a2, 0(zero)
add a0, a2, a2
unimp
```

- (6) How many clock cycles are needed to execute this program?
- (7) Is flushing needed when executing this code? If so, explain when it is needed.
- (8) Is stalling needed when executing this code? If so, explain when it is needed?
- (9) Is forwarding needed when excuting this code? If so, explain when it is needed.

(6)



10 clock cycles are needed.

(7) Yes, on clock cycle 6, when we need to insert a nop.

(8) Yes, on clock cycle 6, we need to stall the add

(9) Yes, in clock cycle 7, we forward from WB to RD0 and RD1 in EX.

(4)

```
addi a1, zero, 3
addi a2, zero, 4
add  a0, a1, a2
unimp
```