Midterm Exam Solution

EECS 370 Winter 2023: Introduction to Computer Organization

You are to abide by the University of Michigan College of Engineering Honor Code. Please sign below to signify that you have kept the honor code pledge:

I have neither given nor received aid on this exam,
nor have I concealed any violations of the Honor Code.

Signature: _____

Name: _____

Uniqname: _____

First/Last name of person sitting to your ***Right*** (Write ⊥ if you are at the end of the row) _____

First/Last name of person sitting to your ***Left*** (Write ⊥ if you are at the end of the row) _____

Exam Directions:

- You have **120 minutes** to complete the exam. There are **8** questions in the exam on **14** pages (double-sided). **Please flip through your exam to ensure you have all 14 pages.**
- You must show your work to be eligible for partial credit!
- Write legibly and dark enough for the scanners to read your answers.
- **Write your uniqname on the line provided at the top of each page.**

Exam Materials:

- You are allotted **one 8.5 x 11 double-sided** note sheet to bring into the exam room.
- You are allowed to use calculators that do not have an internet connection. All other electronic devices, such as cell phones or anything or calculators with an internet connection, are strictly forbidden.

Unique name: _____

1. **Multiple choice/fill in the blank** [16 points, 2 each]

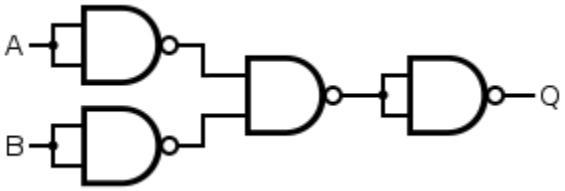
Write the capital letter of the best answer in the box or fill in the blank where indicated.

a. Dennard Scaling roughly states that:

A

A. As transistors get smaller, their power per area stays constant.
B. As transistors get smaller, their power density goes down.
C. About every two years the number of transistors on a chip doubles.
D. As transistors get smaller, their power density doubles about every two years.
E. Processors tend to get unusably hot due to power per area not staying constant.

b. Which of the following formulas is equivalent to the circuit below?



B

A. Q=A and B
B. Q=A nor B
C. Q=A xor B
D. Q=A or B
E. Q=not (A and B)

c. Which two of the following are the primary reasons we prefer not to use latches in our computer designs for things such as the MEM/WB pipeline registers?

1. If the Gate signal is set high for too short of a time there may not be enough time for feedback to stabilize in the latch.
2. If the Gate signal is set high for too long of a time, signals from the EX stage may have time to impact our output before we Gate is set low.
3. If the Gate signal is set low for too long a time, the pipeline register will lose its data.
4. If the Data signal is changing as the Gate goes high, the latch's output will be unstable.

__1__ and __2__ are the two best answers. (Note: these are 1 point each)

d. Write -6 as a 5-bit 2's complement binary number ____0b11010_____.

e. ____31_____ is the largest (closest to positive infinity) number that can be represented in a 6-bit 2's complement representation. Provide your answer in decimal.

Unique name: _____

f. Which one of the following is true about the data hazards and control hazards in a pipelined processor?

B

A. Data hazards only occur when instructions depend on the results of previous instructions, while control hazards only occur when instructions change the data in the pipeline.
B. Data hazards occur when instructions depend on register values written by previous instructions, while control hazards occur when instructions change the flow of execution.
C. Data hazards occur when the pipeline runs out of data, while control hazards occur when the pipeline runs out of control signals.
D. Data hazards occur when multiple instructions access the same control signals, while control hazards occur when instructions change the flow of data.

g. When executing a large program that has no hazards and where every stage always takes 1 cycle, the LC2K pipeline taught in class would expect to achieve a CPI of *about*:

A

A. 1.0
B. 3.5
C. 4.0
D. 5.0
E. It depends on the instruction mix

h. What is the purpose of having some registers be caller-save and other registers be callee-save (as opposed to doing one or the other) when making subroutine calls?

D

A. To ensure that the caller-saved registers are not overwritten by the callee.
B. To ensure that the callee-saved registers are not overwritten by the caller.
C. To prevent data corruption in the pipeline.
D. To reduce the number of registers stored to memory.

Unique name: _____

2. Caller/Callee [8 points]

The code below uses two new-to-us ARM instructions, **push** and **pop**. The **push** instruction stores the register value listed onto the stack. The **pop** instruction loads the data from the stack and places it in the listed register. Both modify the stack pointer as appropriate to add to the stack (push) or remove from the stack (pop). Note: "lr" is the link register (X30) which is where the return address is put by a bl instruction.

```
main:
    movz r1, #0x13
    movz r2, #0x62
    movz r3, #0x83
    movz r4, #0x1
loop:
    cmpi r4, #0x5 ; 4
iterations
    b.eq end
    push {r4}
    bl   bob
    pop  {r4}
    push {r4}
    bl   tom
    pop  {r4}
    push {r4}
    bl   tom
    pop  {r4}
    add  r3, r2, r1
    addi r4, r4, #0x1
    b    loop
tom:
    push {r1}
    push {r2}
    add  r1, r1, r1
    mov  r4, r1
    add  r2, r1, r4
    pop  {r2}
    pop  {r1}
    bl   lr
bob:
    push {r2}
    push {r3}
    movz r2, #0x66
    movz r3, r2
    add  r3, r3, r3
    pop  {r3}
    pop  {r2}
    bl   lr
end:
```

Each register is being used as either callee or caller save. Identify which registers are being used as caller save and which are being used as callee save. Then, count the number of load/store pairs that occur per register when the program "main" is run until the label "end" is reached. **[8]**

Register	Caller or Callee	# load/stores
r1	Callee	2 * 4 = 8
r2	Callee	3 * 4 = 12
r3	Callee	1 * 4 = 4
r4	Caller	3 * 4 = 12

Unique name: _____

3. Short answer [11 points]

a. Consider the following LC2K program.

```
add 1 2 3
lw  1 3 4
add 3 3 3
nor 2 3 4
```

- How many *stall cycles* does the above code incur on the five-stage pipeline discussed in class if we are using "**detect and stall**"? **[3]**

  Line up dependent ID with source WB
  No hazard between add 1 and lw
  2 noops between lw and add 3
  2 noops between add 3 and nor

  4

- How many *stall cycles* does the above code incur on the five-stage pipeline discussed in class if we are using "**detect and forward**"? **[3]**

  Only add 1 for lw followed by dependent

  1

b. Say you have an ISA where all instructions are 32-bits and which has 32 general-purpose registers and all immediate values are 12-bits. If your instruction set consisted of nothing other than instructions that used two general-purpose registers and one immediate as arguments, at most how many opcodes could your ISA support? Place your final answer in the provided box and clearly show your work. **[5]**

Bits to encode each reg = ceil(log_2(#regs)) = ceil(log_2(32)) = 5

32 bits total - 12 bits for immediate - 5 bits for rA - 5 bits for rB = 10 left

We can encode 2^10 values in 10 bits, so there are 2^10 = 1024 opcodes.

1024

Unique name: _____

4. A short float [13 points]

Consider an 8-bit floating point format based on the IEEE standard where the most significant is used for the sign, the next 3 bits are used for the exponent and the last 4 bits are used for the mantissa. The scheme uses "biased 3" to represent the exponent (rather than biased 127 used for a 32-bit IEEE floating point number) and has an implicit one just like the IEEE format. This scheme is called "VSF" (very short float).

Sign	Exponent			Mantissa			
7	6	5	4	3	2	1	0

a. Write the *binary* encoding of 1.5 as a VSF number. **[4]**

0b_00111000_ $1.5 = 1.1000_2 * 2^0$. Add 3 to 0 to get exponent: 0b011
Sign bit is 0. Mantissa is everything after the radix

b. What is the largest (closest to positive infinity) number that can be exactly represented as a VSF? Provide your answer in decimal in the box. **[4]**

Keep it positive, fill everything else with 1's: 0b01111111 -> $2^4 * 1.1111_2 = 31$

31

c. What is the gap between 2.5 and the smallest number larger than 2.5 that can be represented as a VSF? Provide your answer in decimal in the box. **[5]**

$2.5 = 2^1 + 2^{-1} = 1.0100_2 * 2^1$

The last mantissa bit is the smallest amount we can add.

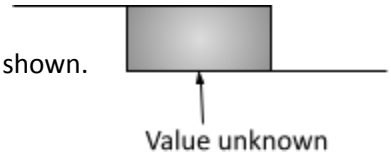
That bit's place value is $2^{-3} = ⅛ = 0.125$

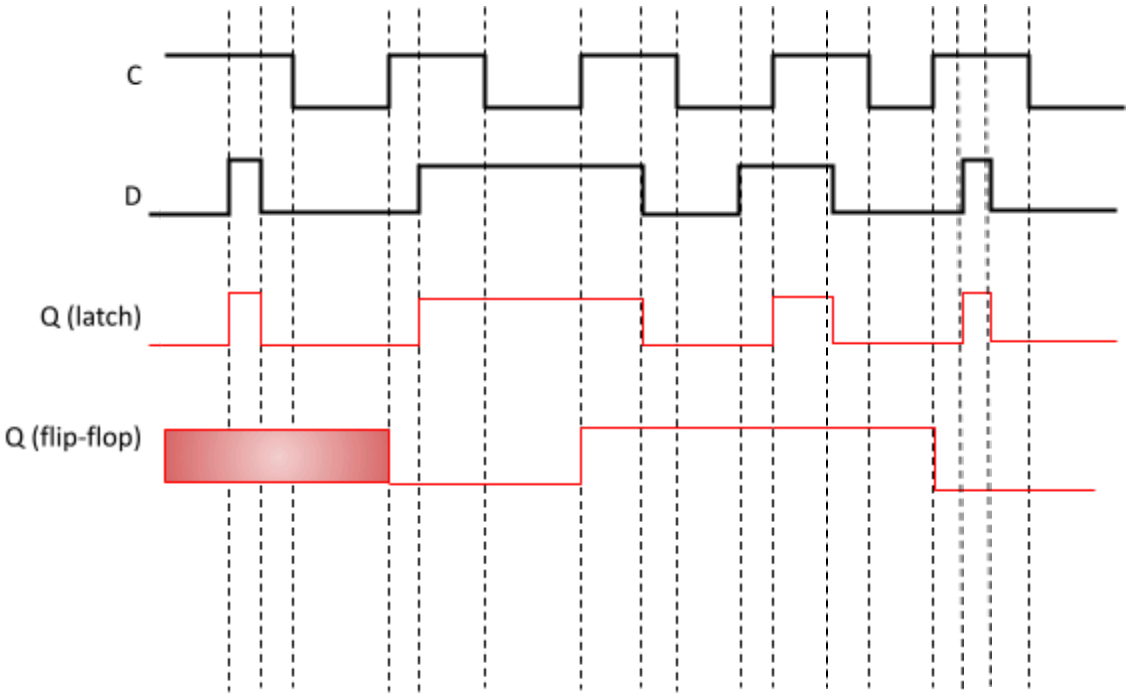
0.125

Unique name: _____

5. Latches and Flip-Flops [6 points]

Complete the timing diagram below for both a D latch and a rising-edge triggered D flip-flop. If a value is unknown, indicate that clearly using the notation shown. Assume there is no meaningful delay. In the case of the latch, "C" is the Gate input. **[6]**





Unique name: _____

6. LC2K linking [13 points]

We've tried to link and assemble our two LC-2K assembly files, but it looks like some of the data got corrupted in one of them. Fill in the blank of the object file. Recall that the 0x notation indicates that the answer is expected in *hexadecimal* (that only applies to the blanks that start with 0x, all other answers should be in the format expected in project 2a).

Line 1: Opcode 0b010, rA 0b000, rB 0b011, offset 0x0009

PC	mary.as	mary.obj
		4 6 5 3
0	lw 0 3 Bob	0x830009
1	lw 0 2 Arr	8519684
2	jalr 3 7	23003136
3	Done halt	25165824
4	Arr .fill 1	1
5	.fill 5	5
6	.fill 2	2
7	.fill 1	1
8	Size .fill 4	4
9	Bob .fill Inc	0x0
		Inc U 0
		Done T 3
		Bob D 5
		Arr D 0
		Size D 4
		0 lw Bob
		1 lw Arr
		5 .fill Inc

Unique name: _____

7. LEGv8 [15 points]

Place your answers in the appropriate box.

Variable to register mappings

```
int32_t t - X2        int32_t v - X4        uint64_t x - X7
int32_t u - X3        int32_t w - X5        uint64_t y - X8
```

a. Convert the following C code to no more than 7 lines of equivalent LEGv8 assembly, assume `int32_t foo[]`'s base address is held in X6. If you need a temporary register, use X1. You must use CBZ or CBNZ rather than any other branch. **[9]**

```
if(t==0)
   w = foo[x+7];
else
   w=3;
```

```
      CBNZ   X2  else ; or #5
if:   LSL    X1, X7,  #2
      ADD    X1, X1,  X6
      LDURSW X5, [X1, #28]
      CBZ    XZR end  ; or #2
else: MOVZ   X5, #3
end:
```

```
      CBZ    X2  if   ; or #3
else: MOVZ   X5, #3
      CBZ    XZR end ; or #4
if:   LSL    X1, X7,  #2
      ADD    X1, X1,  X6
      LDURSW X5, [X1, #28]
end:
```

*For the branches, offsets multiplied by 4 will be accepted due to incorrect answers given on piazza
Using B instead of CBZ XZR should be accepted, even though it said to "use CBZ and CBNZ rather than any other branch"

b. Show the final value of the memory and registers listed after the following LEGv8 code runs. You are to assume that all registers and any memory value not shown are initialized to be zero. Assume we are in little endian mode. Put all answers in hex. **[4]**

```
LDURSW    X4, [X31, #100]
STURW     X3, [X31, #104]
STURB     X4, [X31, #107]
LDURSW    X3, [X31, #104]
```

X3: 0xFFFFFFFF80000000
(Sign extension **IS** required)

X4: 0x55F90880

(Leading zeros are fine)

Memory location	Initial value	Final value (in hex)
100	0x80	0x80
101	0x08	0x08
102	0xF9	0xF9
103	0x55	0x55
104	0x44	0x00
105	0xCC	0x00
106	0xBB	0x00
107	0x22	0x80

Unique name: _____

8. Stack it Up [18 points]

Consider a 8-bit stack-based ISA named "HOB" (identical to the ISA from homework 3). There are 3 instruction formats (bit 0 is the least-significant bit).

R-type instructions (add, nor):	bits 7-5: opcode	bits 4-0: unused (should all be 0)
I-type instructions (pushi):	bits 7-5: opcode	bits 4-0: value (2's complement)
A-type instructions (push, pop, beq)	bits 7-5: opcode	bits 4-0: address (unsigned)

Assembly language Instruction	**Opcode in binary (bits 7, 6, 5)**	**Action**
add (R-type)	000	Pop the top two values off of the stack and push on the sum of those two values back on the stack.
nor (R-type)	001	Pop the top two values off of the stack and push on the bitwise NOR of those two values back on the stack.
push (A-type)	010	Take the value stored at memory location specified by the address field and push it onto the stack
pop (A-type)	011	Take the value at the top of the stack and pop it. Place that value into the memory location specified by the address field.
pushi (I-type)	100	Push the sign-extended value stored in the value field.
beq (A-type)	101	If the two values on the top of the stack are equal, pop them and branch to the location specified by the address field. Otherwise do nothing (don't change anything on the stack.)
halt (R-type)	110	Increment the PC (as with all instructions), then halt.

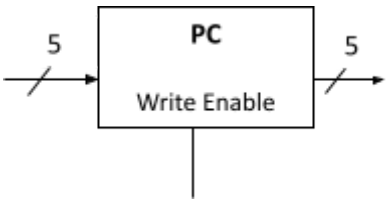
a. Translate the following instructions into *hexadecimal*. **[3]**

Instruction	**Hex**
nor	0x20
pushi -1	0x9F
beq 4	0xA4

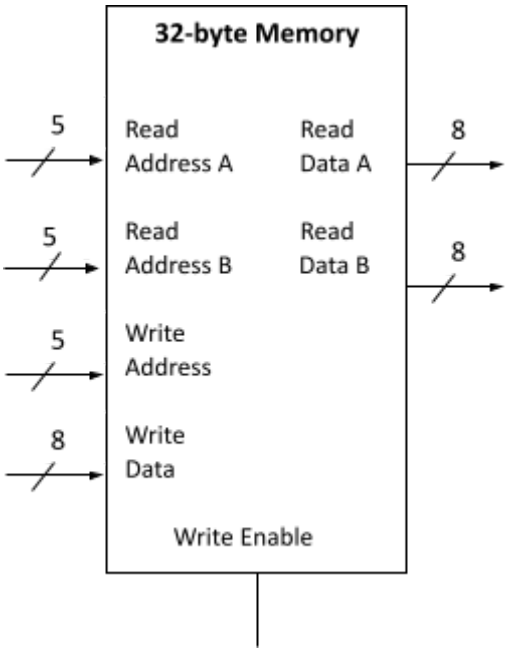
Unique name: _____

b. Say you have the following devices:

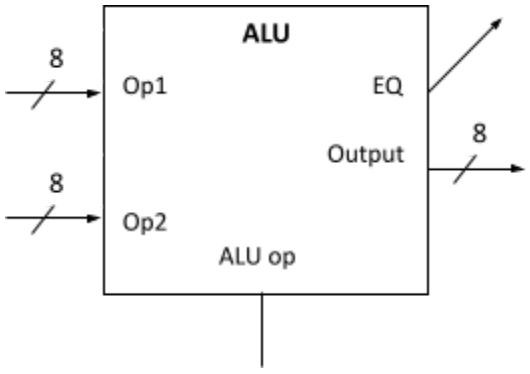
- **PC**: This is a 5-bit register with a write-enable control. When first powered up it initializes itself to zero. When the write-enable is 1, the device will store the input value on the next rising edge of the clock. (That is, at the end of the clock cycle.)



- **32-byte memory**: When an address is presented to the Read Address A input, the data stored in that address is immediately written to Read Data A. When an address is presented to the Read Address B input, the data stored in that address is immediately written to Read Data B. When Write Enable is a 1, Write Data is written to the address specified by Write Address on the rising edge of the system clock. (That is, at the end of the clock cycle)



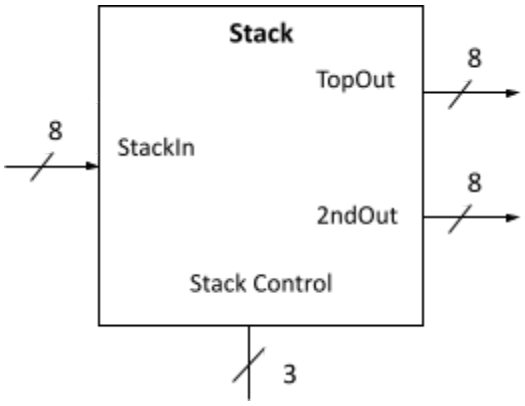
- **ALU**: When ALUop is 1, the value Op1 + Op2 is placed on Output. When ALUop is 0, the bitwise NOR of Op1 and Op2 is placed on Output. Also, if Op1 and Op2 are equal then EQ=1, else EQ=0.



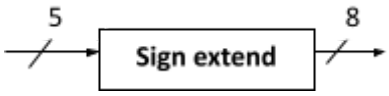
Unique name: _____

- **Stack:** The top value of the stack is placed on TopOut, the second highest value is placed on 2ndOut. Further the control signal has the following options:
    - 0: do nothing
    - 1: pop the top value
    - 2: push the value StackIn onto the top of the stack.
    - 3: pop the top two values from the stack
    - 4: pop the top two values and then push the value StackIn onto the top of the stack.

  As with the other registered devices, any changes to the stack don't occur until the end of the clock cycle.



- **Sign extend:** This device sign extends a 5-bit 2's complement number into an 8-bit 2's complement number.

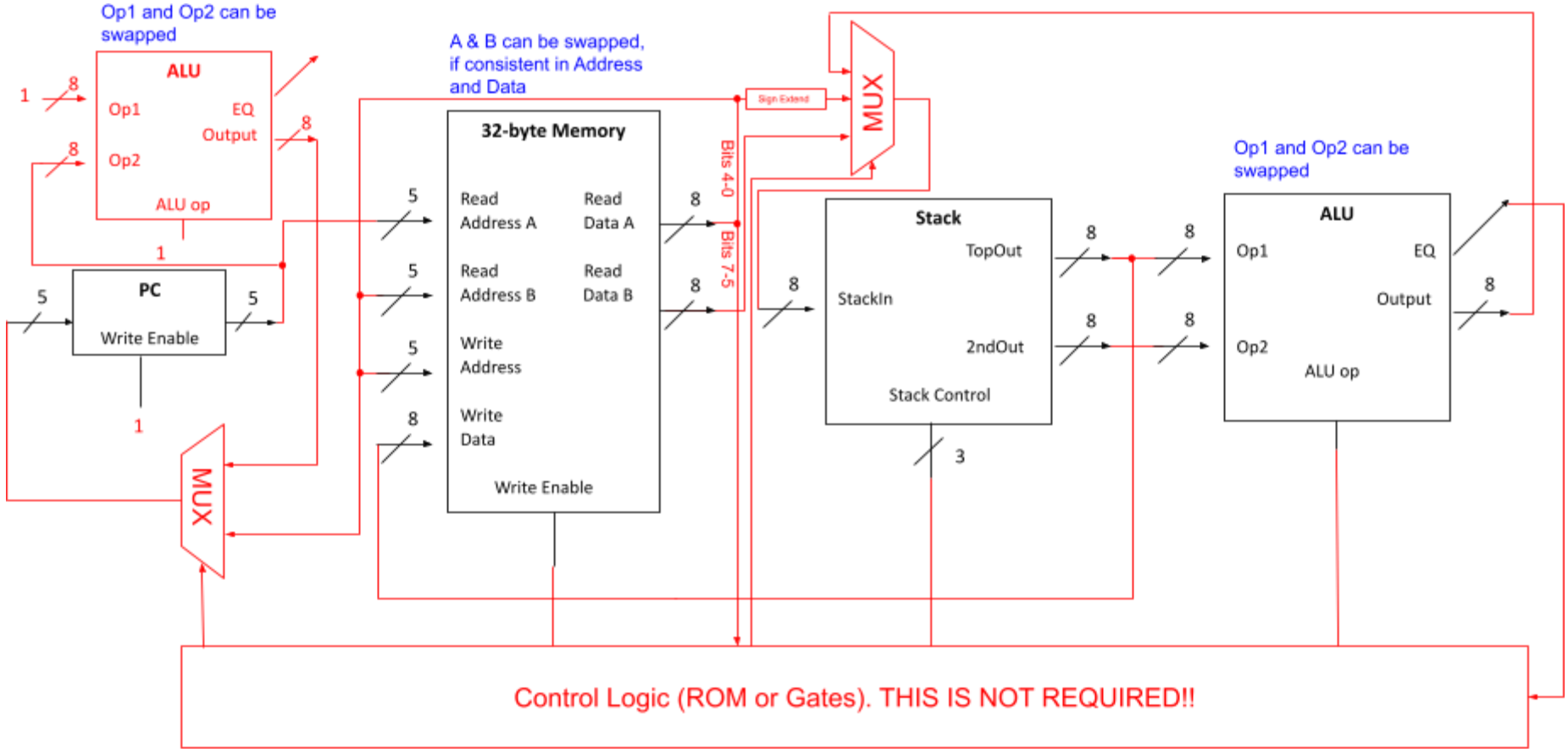


Using the above devices plus wires and Muxes, create a *single-cycle data path* for the HOB ISA. You may drive constant values onto the wires if needed and should use as little hardware as possible. We have supplied two copies of a template for your answer. ***You may need to add additional devices from the above list.***

Note: we are asking for the data path only, you aren't implementing the control part (no ROM, etc.). Thus, you will leave control points (e.g. MUX inputs, Stack Control, Write Enable) unconnected.

The next two pages have identical templates for this question (in case you make mistakes). *You must clearly indicate which copy of the template you want us to grade (by clearly crossing out the one you don't want graded), otherwise we will grade the first one.* **[15 points]**

Unique name: _____



Datapath: PC + 1 --> PC	1.5
Datapath: Offset --> PC	1.5
Datapath: ALU output --> Stack	1.5
Datapath: Sign-Extended Offset --> Stack	1.5
Datapath: Memory Data Value --> Stack	1.5
Datapath: PC --> Memory Read Address	1.5
Datapath: Offset --> Memory Read Address	1.5
Datapath: Offset --> Memory Write Address	1.5
Datapath: Stack TopOut --> Memory Write Data	1.5
Datapath: Stack Outputs --> ALU ops	1.5