| Midterm Exam  *\_\_\_\_\_ \_\_\_\_\_ \_\_\_\_ \_\_\_\_ \_\_\_\_\_\_\_\_\_\_ \_\_\_*  *| \_\_\_\_| \_\_\_\_/ \_\_\_/ \_\_\_| |\_\_\_ /\_\_\_ / \_ \*  *| \_| | \_|| | \\_\_\_ \ |\_ \ / / | | |*  *| |\_\_\_| |\_\_| |\_\_\_ \_\_\_) | \_\_\_) |/ /| |\_| |*  *|\_\_\_\_\_|\_\_\_\_\_\\_\_\_\_|\_\_\_\_/ |\_\_\_\_//\_/ \\_\_\_/*  EECS 370 Fall 2024: 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: | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* | |
| Uniqname of person sitting to your ***Right***  **(**Write 丄 if you are at the end of the row) | | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
| Uniqname of person sitting to your ***Left***  **(**Write 丄 if you are at the end of the row) | | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |

**Exam Directions:**

* You have **120 minutes** to complete the exam. There are 7questions in the exam on 16 pages (double-sided). **Please flip through your exam to ensure you have all pages.**
* 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.

| *Problem* | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| --- | --- | --- | --- | --- | --- | --- | --- |
| *Point Value* | 14 | 15 | 9 | 8 | 14 | 20 | 20 |

# THIS PAGE INTENTIONALLY LEFT BLANK

# 

# Problem 1: Multiple Choice 14 points

Completely shade in the boxes with the best answer. Select only 1 answer unless specified otherwise. *[1 point each]*

1. While \_\_\_\_\_\_\_ has largely stopped since the mid-2000s, \_\_\_\_\_\_\_ there have been ongoing efforts to keep \_\_\_\_\_\_\_ alive.
   * Moore's Law / Von Neuman's Law
   * Von Neumann's Law / Moore's Law
   * Dennard Scaling / Von Neuman's Law
   * Von Neumann's Law / Dennard Scaling
   * Moore's Law / Dennard Scaling
   * Dennard Scaling / Moore's Law
2. LC2K is an example of a \_\_\_\_\_\_ architecture. (**FILL IN ALL THAT APPLY)**
   * RISC
   * CISC
   * Von Neumann
   * Single-instruction
3. Which of the following is run at any time earlier than the linker in the build process? (**FILL IN ALL THAT APPLY)**
   * Assembler
   * Debugger
   * Compiler
   * Loader
4. Which of the following values is stored in **binary** when the program is run?

int x = 107;

int y = 0x13A;

* + Neither x nor y
  + x, but not y
  + y, but not x
  + x and y

1. Recalling that a float uses 8 bits for the exponent and 23 bits for the significand, the difference between 1.25\*23 and the next largest value that can be represented exactly as a single precision floating point value is:
   * 2-23
   * 2-22
   * 2-21
   * 2-20
   * 1.25\*2-23
   * 1.25\*2-22
   * 1.25\*2-21
   * 1.25\*2-20
2. Which of the following **best** describes why beq instructions don't need to go in the relocation table for project 2a?
   * Branches do not access anything in memory
   * The address of a branch's target can be resolved before the linking stage
   * The distance between a branch and its target will not change during linking
   * Branch targets can use hard coded offsets rather than labels
3. Instructions are placed in the \_\_\_\_\_\_\_ section of memory
   * Stack
   * Heap
   * Static
   * Executive
   * Text
4. When compiling C to assembly, the first element of a struct is aligned in memory based off: (**FILL IN ALL THAT APPLY)**
   * the size of the smallest primitive in the struct
   * the size of the largest primitive in the struct
   * the total size of the struct, not including arrays or nested structs
   * the total size of the struct, including arrays or nested structs
5. Compared to the number of instructions listed in an assembly file, the number of instructions executed when run is guaranteed to be:
   * smaller
   * smaller or equal
   * larger
   * larger or equal
   * none of the above (could be smaller, greater or equal)
6. Which of the following elements below are universal (i.e. can be used to implement any other logical operation) (**FILL IN ALL THAT APPLY)**
   * NOT
   * AND
   * OR
   * NAND
   * NOR
   * XOR

# Problem 2: Short Answers 15 points

1. Convert each of the 8-bit hexadecimal numbers into its binary form and decimal form (both for treating the original number as a signed (two's-complement) and an unsigned value). *[3 points]*

| **Hexadecimal** | **Binary** | **Decimal (signed)** | **Decimal (unsigned)** |
| --- | --- | --- | --- |
| 0x7A |  |  |  |
| 0x9E |  |  |  |

1. For each element of the struct below, list out the starting and ending (inclusive) addresses in hex, assuming address 0x1000 is the first address available and it is run on a 64-bit architecture with datatype sizes included in the reference material: *[5 points]*

struct {

int a;

long int b[3];

char c;

struct {

char d\_1;

int d\_2;

char d\_3;

} d;

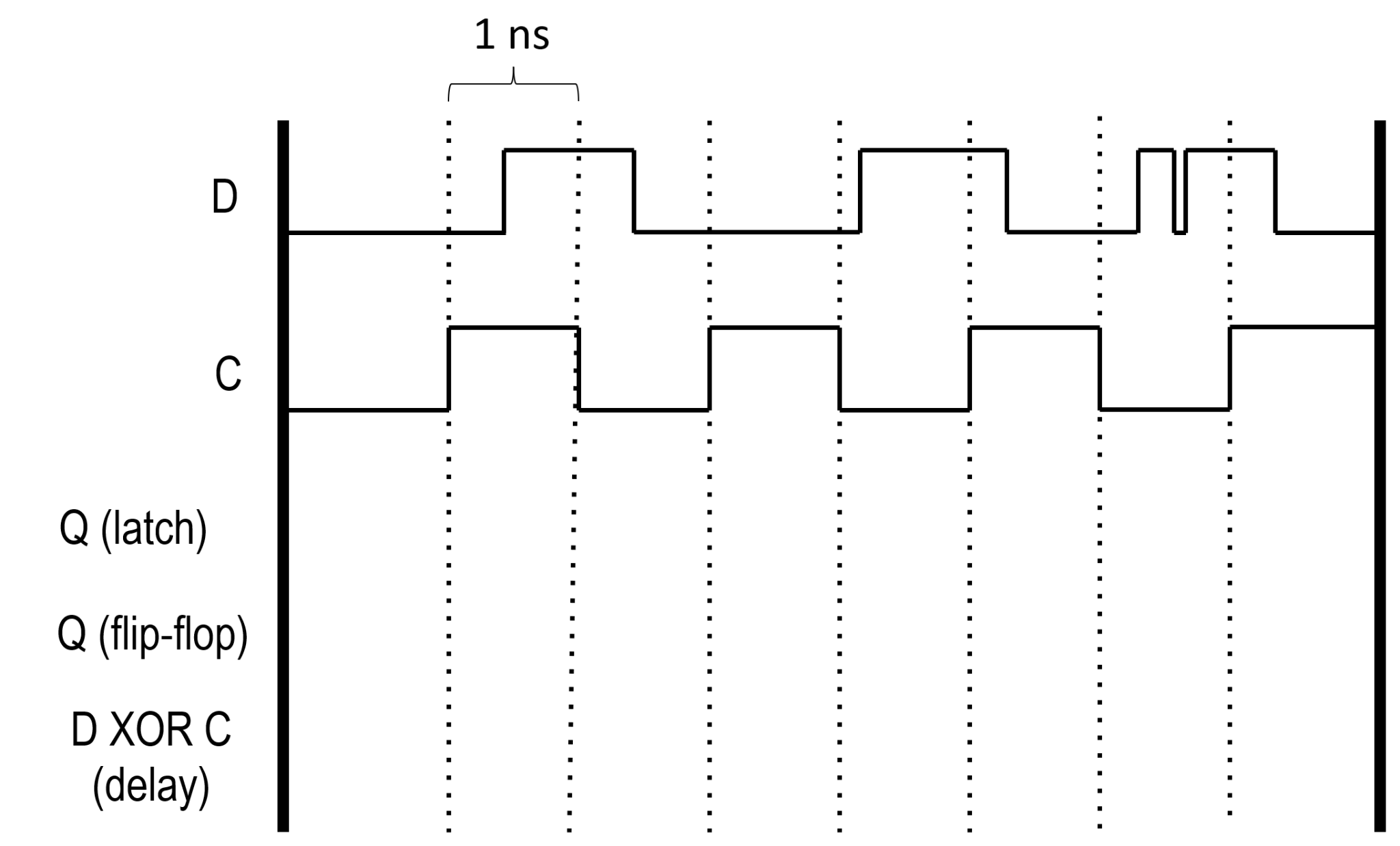
short\* e;

} my\_struct;

| Element | Starting address (hex) | Ending address (hex) |
| --- | --- | --- |
| a |  |  |
| b |  |  |
| c |  |  |
| d |  |  |
| e |  |  |

1. Complete the timing diagram below for (1) a D latch, (2) a rising-edge triggered D flip-flop, and (3) an XOR gate whose output is 1 ns delayed from any input changes. If a value is unknown given the information, indicate that clearly using the notation shown. Assume there is no meaningful delay within the components of the D latch or flip-flop. In the case of the latch, “C” is the Gate input. *[3 points]*





1. 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 points]*

# 

| Memory location | Initial value | Final value (in hex) |
| --- | --- | --- |
| 100 | 0x13 |  |
| 101 | 0x0F |  |
| 102 | 0xF2 |  |
| 103 | 0xCA |  |
| 104 | 0x90 |  |
| 105 | 0x33 |  |
| 106 | 0x7B |  |
| 107 | 0x20 |  |

# **LDURSW X0, [XZR, #100]**

**STURW X1, [XZR, #104]**

**STURH X0, [XZR, #106]**

**LDURSW X1, [XZR, #104]**

| **Reg** | **Value** |
| --- | --- |
| **X0** |  |
| **X1** |  |

# Problem 3: Bench(mark) Warmer 9 points

Consider two implementations of an ISA.

Single cycle design:

* Clock period of 15ns
* Cycles per instruction:

| **Instruction** | **Cycles** |
| --- | --- |
| All instructions | 1 cycle |

Multi-cycle design:

* Clock period of 1ns
* Cycles per instruction:

| **Instruction** | **Cycles** |
| --- | --- |
| Add | 5 cycles |
| Branch | 6 cycles |
| Load and store | 7 cycles |

The a 1 million instruction program is run with the following distribution:

| **Instruction** | **Percentage** |
| --- | --- |
| Add | 30% |
| Branch | 20% |
| Load and store | 50% |

1. What is the execution time (in milliseconds) of the program on the single-cycle design? Write your final answer in the box and show all work. Recall 1 second = 1000 milliseconds = 1 billion nanoseconds *[4 points]*



1. What is the execution time (in milliseconds) of the program on the multi-cycle design? Write your final answer in the box and show all work *[5 points]*



# Problem 4: Caller/Callee Save 8 points

Count the number caller/callee-save pairs **executed** for each variable when driver() is called once. Assume the compiler checks for liveness across function calls **but does not make any other optimizations.**

| void driver( void ){  int x=2, y=3;  printf("Starting"\n);    x = x + 1;  y = x – 3;  for(int i=0;i<4;i++){  buddy();  }  return;  } | void buddy( void ){  int j=0;  printf("In function\n");  if(j != j){  j++;  printf("Uh-oh\n"); }  j++;  return;  } |
| --- | --- |

| Variables | Caller load/store pairs | Callee load/stores pairs |
| --- | --- | --- |
| x |  |  |
| y |  |  |
| i |  |  |
| j |  |  |

# Problem 5: Assembling and Linking 14 points

1. 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. *(Note: the object file is shown starting a line earlier to allow the relevant parts of assembly to line up) [14 points]*

| **eecs.as** |  | **eecs.obj** |
| --- | --- | --- |
| lw 0 3 Loc          lw   0 2 Lst  beq 0 2 Done          jalr 3 7  Done    halt  Lst  .fill 6         .fill 2         .fill 1         .fill 7  Size   .fill 3  Loc  .fill Func | 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | \_\_ \_\_ \_\_ \_\_  0x0083\_\_\_\_\_  0x0082\_\_\_\_\_  0x0102\_\_\_\_\_  0x015F0000  0x01800000  0x6  0x2  0x1  0x7  0x3  0x\_\_\_\_\_\_  Done \_\_\_\_\_\_  Lst \_\_\_ \_\_\_  Size \_\_\_ \_\_\_  Loc \_\_\_ \_\_\_  \_\_\_\_ \_\_\_\_ \_\_\_\_  0 \_\_\_\_ \_\_\_\_  1 \_\_\_\_ \_\_\_\_  5 \_\_\_\_ \_\_\_\_ |

# 

# 

# Problem 6: ARM Assembly 20 points

Fill in the blanks to complete the following C-to-LEGv8 assembly conversion. You may not define any other labels. Assume we are following the ARM ABI discussed in class (see reference material), and assume that the call stack **grows downwards** in memory as items are placed on the stack (i.e. memory addresses get smaller as more items are added to the stack). Remember that LEGv8 is a 64-bit system.

| **int helper(int x);**  **int data[8]; // assume array**  **// starts at address**  **// 1000**  **void proc(int n) {**  **int j = n \* 9;**  **while(helper(j) < 10) {**  **j += 1;**  **}**  **data[j % 8] = j;**  **return;**  **}** | proc:  subi x28, x28, \_\_\_\_  stur \_\_\_\_, [x28, #0]  \_\_\_\_\_\_\_\_\_\_\_\_\_  add x0, x0, x1  while:  subi x28, x28, #4  sturw \_\_\_\_\_, [x28, #0]  \_\_\_\_ helper  cmpi \_\_\_\_\_, #10  \_\_\_\_\_ \_\_\_\_, [x28, #0]  addi x28, x28, #4  b.ge end  \_\_\_\_\_\_\_\_\_\_\_\_\_\_  b while  end:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_  lsl x1, x1, #2  sturw x0, [x1, #1000]  ldur \_\_\_\_\_, [x28, #0]  addi x28, x28, #8  br x30 |
| --- | --- |

# 

# 

# Problem 7: Big Macc 20 points

A common instruction in many ISAs is "multiply-accumulate" (MACC), in which a destination register is incremented by the product of two other registers, i.e.

destReg = destReg + regA \* regB

Assume we replace our jalr instruction with MACC, encoding it as an R-Type instruction (destReg in bits 2-0, source regs in 21-19 and 18-16). We build a multi-cycle datapath, performing the steps below for each cycle for MACC:

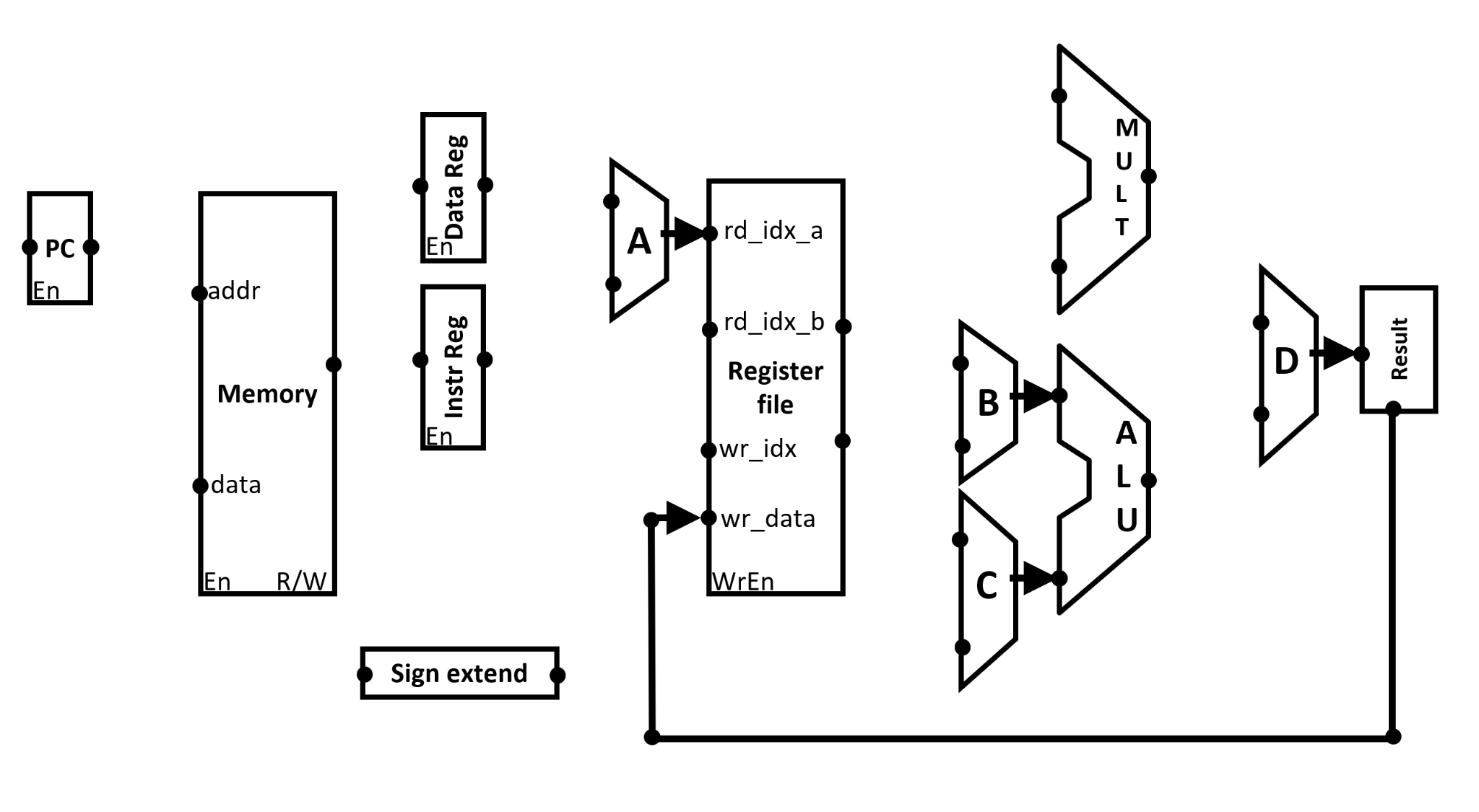
| **Cycle** | **MACC Actions** |
| --- | --- |
| 1 | Fetch instruction, calculate next PC |
| 2 | Start calculation, update PC |
| 3 | Finish calculation |
| 4 | Write back result |

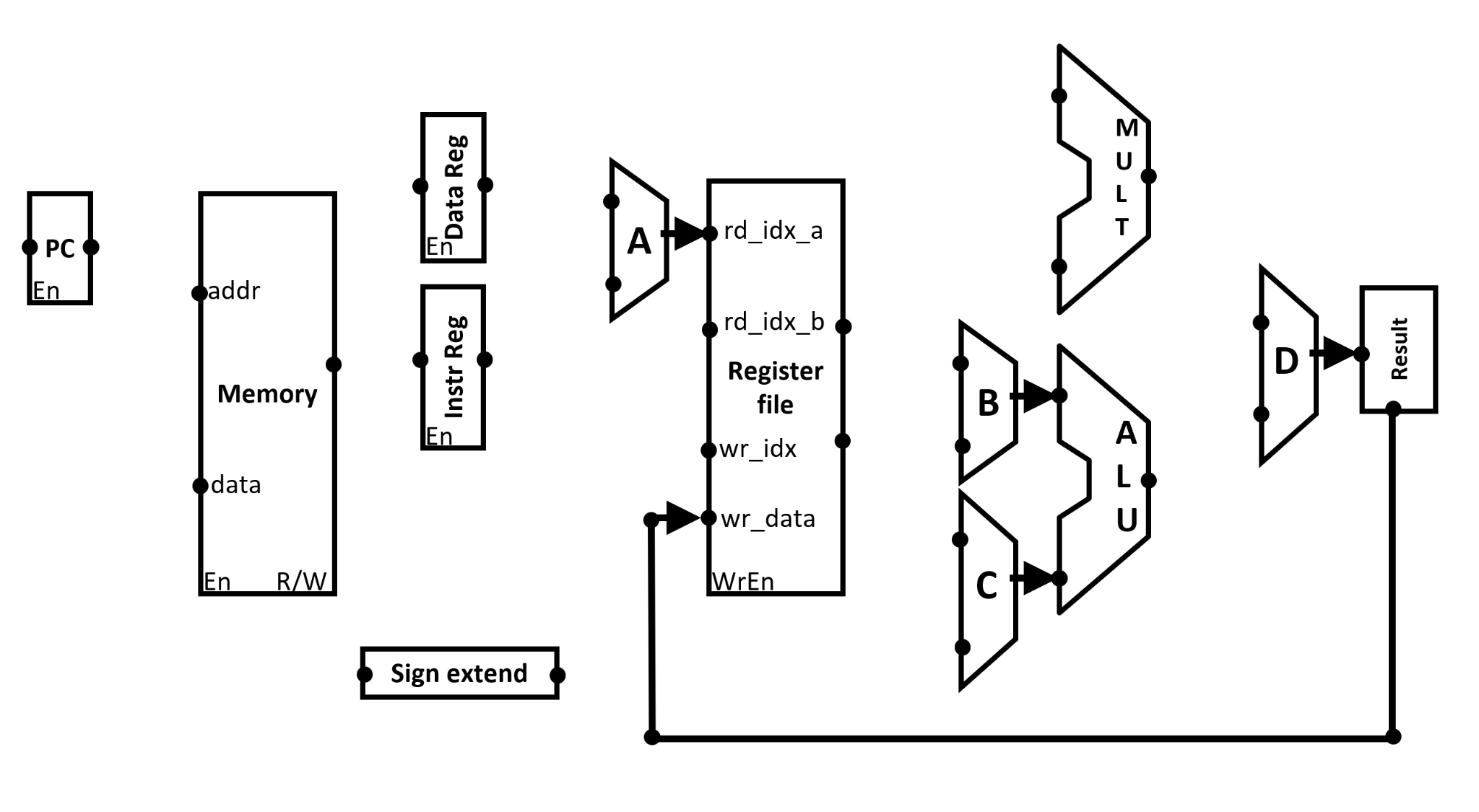
Assume a full memory cycle is needed to read an instruction from memory, **but unlike in lecture, a register value can be read and used on the same cycle**.

1. On the datapath below, add the connections, along with any hardcoded 1s or 0s, needed to support MACC. **Any connections needed for other LC2K instructions but not MACC will lose points if included.** You may not add any other muxes or other components, and you **do not need to show control signals.** *[10 points]*

Notes:

* All muxes (labeled A-D) are 2-inputs
* "MULT" calculates the product of its two inputs using combinational logic
* All other components are as described in lecture
* Place a clear dot on any crossing lines which should be connected. Crossing lines without a dot will be assumed to be unconnected.
* In any instance where a multi-bit signal is split into different groups of signals, you **must label which bits are which** by placing an inclusive number range above each line
* Two copies of the datapath are provided. Place a large 'X' through the copy you do not want graded.





1. List the control signals needed to implement each of the 4 cycles for MACC below. Any control signals whose values don't matter must be listed as "X". The description of each cycle is copied below for reference *[10 points]*

| **Cycle** | **MACC Actions** |
| --- | --- |
| 1 | Fetch instruction, calculate next PC |
| 2 | Start calculation, update PC |
| 3 | Finish calculation |
| 4 | Write back result |

| Cycle | PC en | Mem en | Mem R/W | DataReg en | InstReg en | muxA sel | RegFile WrEn | muxB sel | muxC sel | ALU op | muxD sel |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** |  |  |  |  |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |  |  |  |  |  |
| **3** |  |  |  |  |  |  |  |  |  |  |  |
| **4** |  |  |  |  |  |  |  |  |  |  |  |

THIS PAGE INTENTIONALLY LEFT BLANK