

EXAM 2

Tuesday, November 22, 2005

Instructions

1. This is an open books/notes exam.
2. It comprises:
 - 7 multiple choice questions, each worth 5 points (35 points total).
 - 3 design questions, worth a total of 65 points.
3. Each question has only one correct answer.
4. For each question, indicate your answer directly on the exam.
5. You have 80 minutes to complete the exam.
6. Please make sure you enter your name and UMID in the space below AND on the answer sheet.
7. By signing below, you certify that your conduct throughout the exam has been in accordance with the College of Engineering Honor Code.
8. At the end of the exam, please turn in this exam booklet.

Name _____

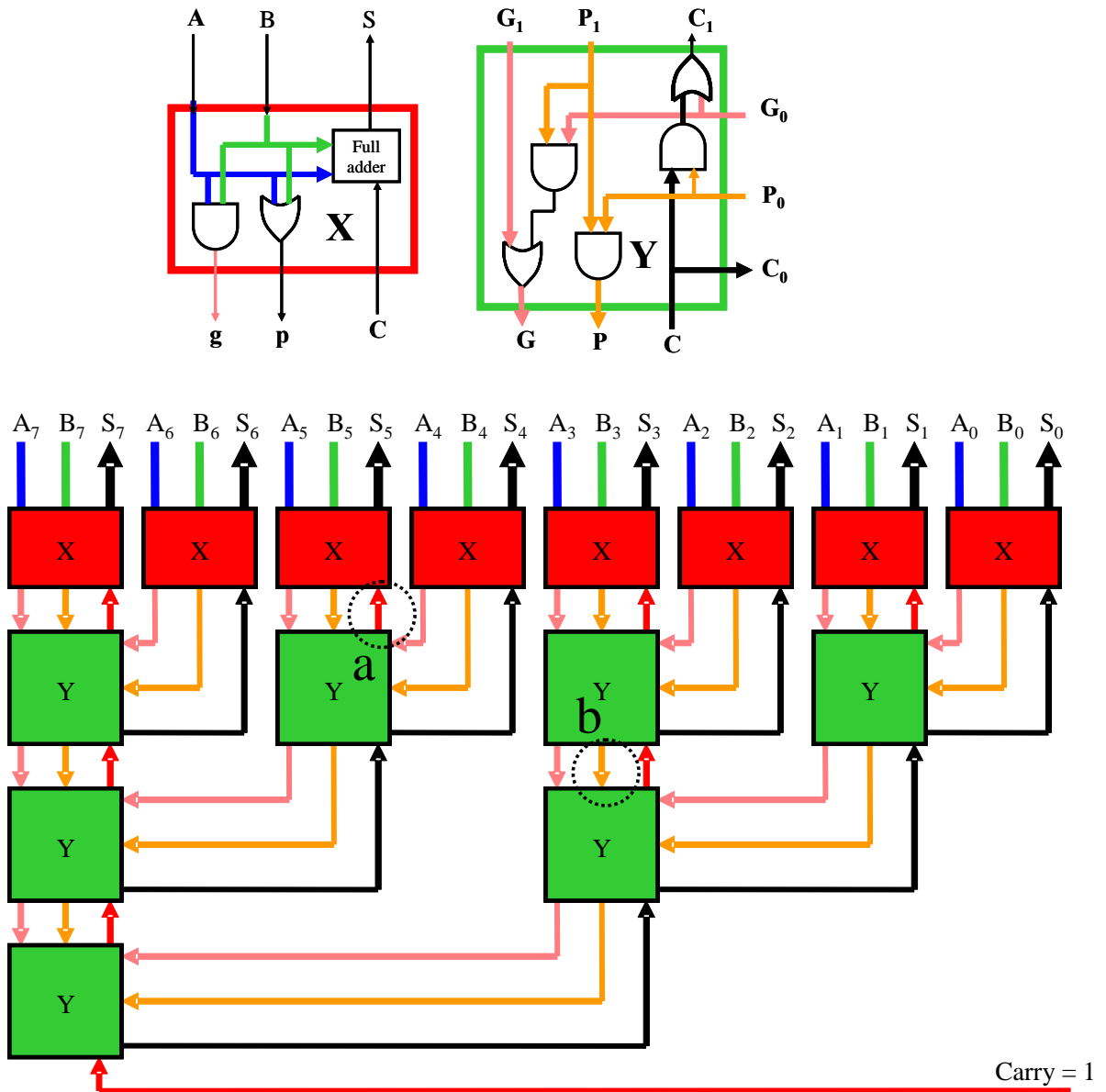
UMID _____

Signature _____

Section I: Multiple Choice [35 points]

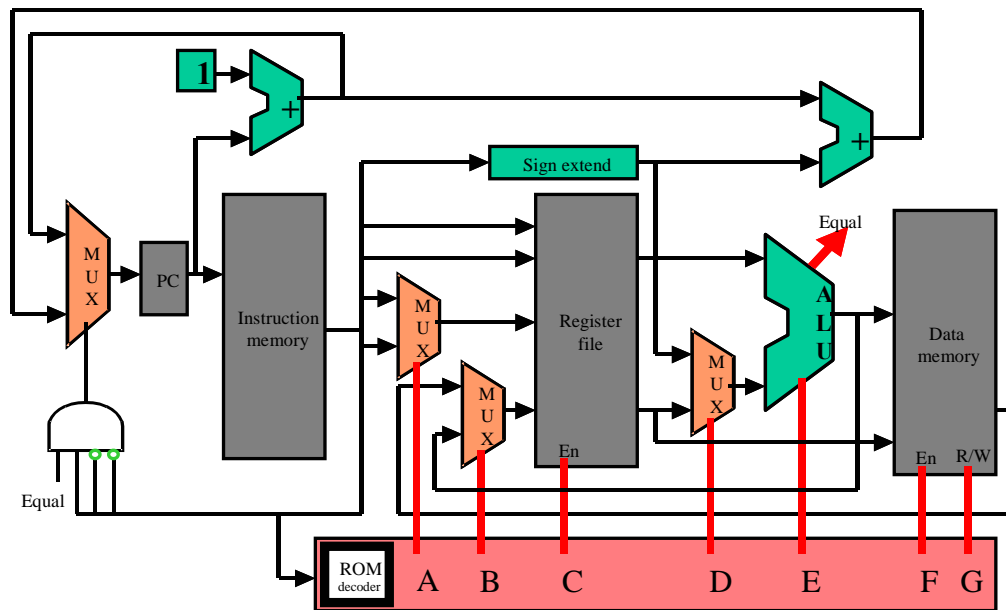
Circle the correct answer

1. [5 points] Using the following 8-bit Carry Look-ahead Adder to add 188 (10111100) and 41 (00101001) with **input Carry = 1**, what are the values of a and b as labeled by dotted circles?



- $ab = 00$
- $ab = 01$
- $ab = 10$
- $ab = 11$ (ANS)
- I don't have a clue how carry lookahead works.

Questions 2 - 3 refer to the following single-cycle datapath implementation of LC-2K5 that was covered in lecture and shown below.



2. [5 points] Assume the following delays for the datapath components

Memory	4 ns
Register file	1 ns
ALU	3 ns
Adder	3 ns

All other components have 0 delays. What is the minimum clock period possible for this datapath?

- a. 6 ns
- b. 8 ns
- c. 11 ns
- d. 12 ns
- e. 13 ns (ANS)

3. [5 points] Among the 7 control signals A-G, pick all the signals that are safe to be set to “don’t care” for both SW and BEQ instructions.

- a. A
- b. AB (ANS)
- c. ABE
- d. ABG
- e. ABEG

4. [5 points] Consider a block of code with 100 assembly instructions in the format of “lw, nand, lw, nand, ...”. Each nand instruction depends and only depends on the lw instruction right before it. All the lw instructions do not depend on any previous instructions. The code is executed on a 20-stage pipelined datapath, where data memory accesses occur on the 15th stage, execute is in the 10th stage, and register file reads occur on the 9th stage. There are full data forwarding paths to the execute stage. Hazards that cannot be resolved by data forwarding are avoided using the “detect and stall” method as discussed in lecture. How many cycles does this block of code take to run on this pipelined datapath? Make sure you include cycles to fill and to drain the pipeline.
- 300
 - 350
 - 369 (ANS)
 - 400
 - 419
5. [5 points] Consider running the following assembly program on the 5-stage, LC-2K5 pipelined datapath covered in lecture. **The register file correctly resolves writes/reads to the same register in same cycle, as given in lecture.** Assuming there is no hazard detection, no data forwarding paths, and no branch prediction, what is the minimal number of noops that need to be inserted into the program by the programmer or the compiler to guarantee correct execution?

```

                lw    0    1    neg1
                lw    1    3    neg1
begin         beq    0    3    exit
                add   1    3    3
                lw    2    4    val
                nand  3    4    4
                sw    2    4    loc
                beq   0    0    begin
exit         halt
val         .fill 10
neg1       .fill -1
loc        .fill 0

```

- 8
- 11
- 12
- 14 (ANS)
- 16

6. [5 points] What is the IEEE 754 representation of -315.4375?

- a. 0 10000111 001110110011000000000000
- b. 1 10000111 001110110110000000000000
- c. 1 10000111 001110110011100000000000
- d. 1 10000111 001110110111000000000000 (ANS)
- e. 1 10000111 001110111111000000000000

7. Consider the execution of a program with the following instruction mix on the pipelined LC-2K5.

- R-type: 40%
- Stores: 20%
- Loads: 30%
- Branches: 10%

Assume full data forwarding is in place. 50% of the loads stall, with a total penalty of 1 cycle. 10% of the branches are mis-predicted with a total penalty of 2 cycles. I-cache miss rate is 2%, and D-cache miss rate is 4%. For each cache, the total miss penalty is 20 cycles. What is the CPI of the program? (The term “penalty” captures the total overhead in addition to the normal number of cycles required for the operation when there is no cache miss, branch mis-prediction, or load hazard.)

- a. 1.17
- b. 1.57
- c. 1.82
- d. 1.97 (ANS)
- e. 2.07

Section II: Design questions [65 points]

Part A. Caches [25 points]

You are given the following cache configuration information:

Cache size = 16 bytes
 Block size = 2 bytes
 Direct mapped

Write allocate
 LRU replacement policy
 Address size = 16 bits, byte addressable memory

1. [4 points] Given the address 0xF19C, what are the tag and index values (specify in binary or hex)?

Tag = **0xF19 (1111 0001 1001)**

Set index = **0x6 (110)**

2. [9 points] Given the following sequence of addresses, label each reference as a hit (H) or miss (M). For the misses, indicate which are conflict misses. Each reference is a single byte of data.

Address	4	7	21	22	23	20	5	7	36	6	4
H/M	M	M	M	M	H	H	M	M	M	H	M
Conflict?							C	C			C

3. [4 points] What is the **average** memory latency (measured in cycles) for the previous reference stream if hits take 1 cycle and the miss penalty is 10 cycles (note: miss penalty excludes bringing the data from the cache to the register)?

$$\begin{aligned}
 \text{Avg latency} &= ((\# \text{hits} * \text{hit latency}) + (\# \text{misses} * \text{miss latency})) / \# \text{accesses} \\
 &= ((3 * 1) + (8 * (10 + 1))) / 11 \\
 &= 91 / 11 \\
 &\approx 8.27
 \end{aligned}$$

4. [8 points] Given that you want the reference stream hit/miss behavior to be the following:

Address	4	7	21	22	23	20	5	7	36	6	4
H/M	M	M	M	M	H	H	H	H	M	H	H

What is the minimum cache associativity which will result in this behavior (all other parameters, e.g. size, block size, LRU are unchanged)? Show your work. (Guesses will be given 0 credit.)

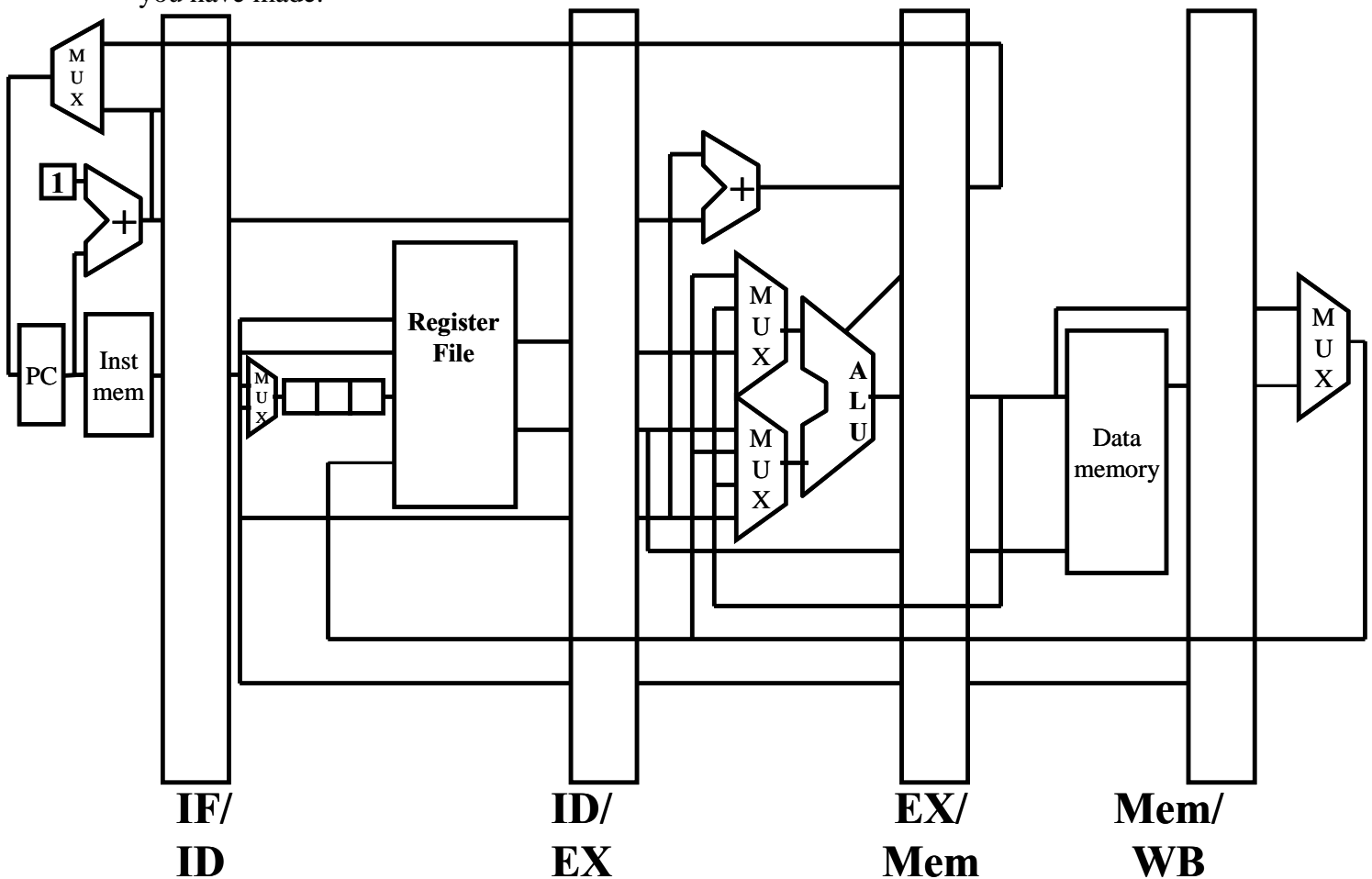
2-way associative.

<u>Set 2 contents</u>	<u>Set 3 contents</u>	<u>Access</u>	<u>Result</u>
-	-	4	M
4 5	-	7	M
4 5	6 7	21	M
4 5, 20 21	6 7	22	M
4 5, 20 21	6 7, 22 23	23	H
4 5, 20 21	6 7, 22 23	20	H
4 5, 20 21	6 7, 22 23	5	H
4 5, 20 21	6 7, 22 23	7	H
4 5, 20 21	6 7, 22 23	36	M
4 5, 36 37	6 7, 22 23	6	H
4 5, 36 37	6 7, 22 23	4	H

Part B. Pipelined Datapath [20 points]

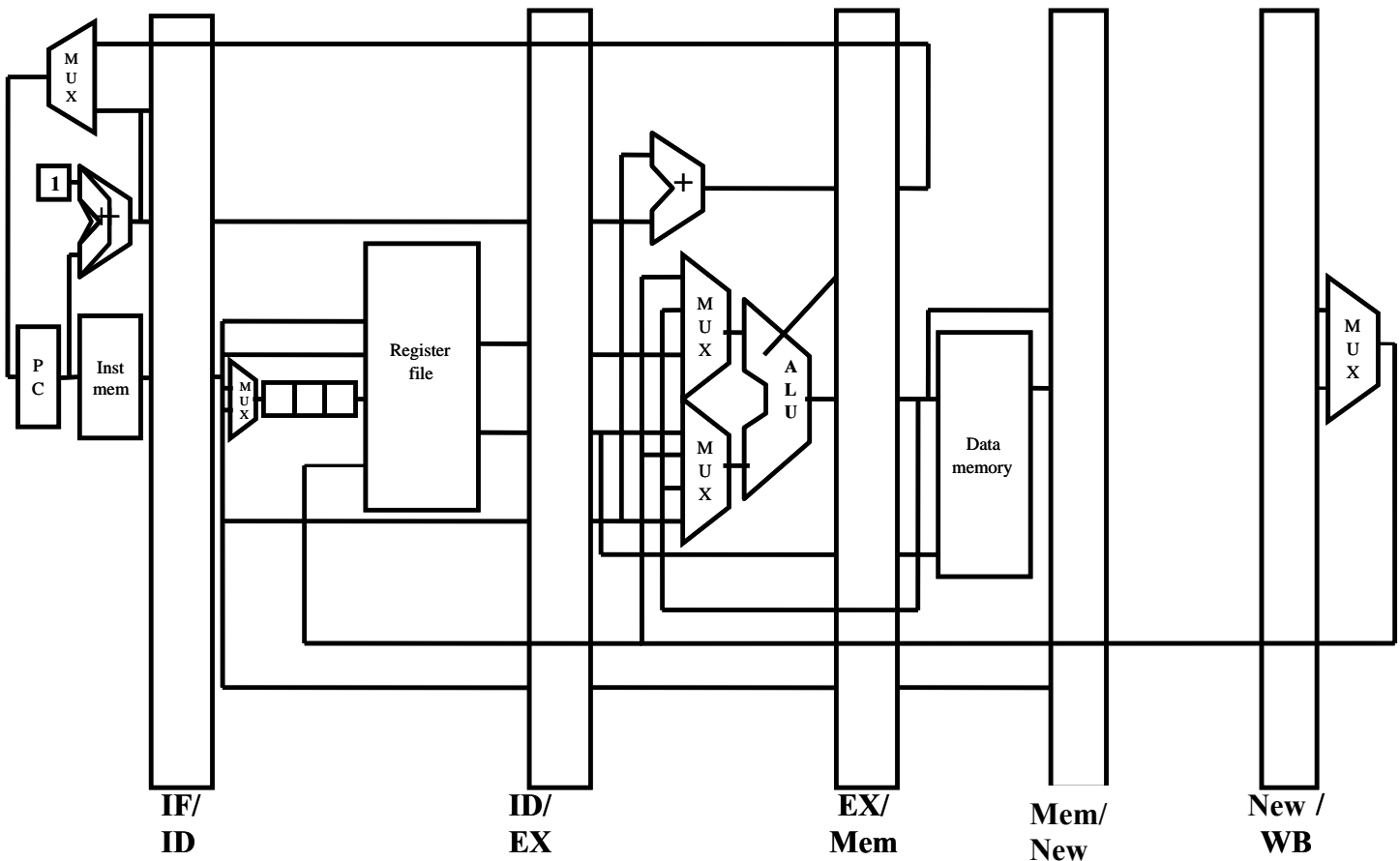
Suppose that you want to extend the LC-2K5 instruction set to support CISC instructions to increase code density. The CISC instructions that we are interested in are those that take a source operand directly from memory and those that write their destination to memory. For instance, add-to-memory, `addtm regA regB destReg`, has the following semantics: $\text{Mem}[\text{destReg}] = \text{regA} + \text{regB}$.

- [10 points] Extend the LC-2k5 pipeline datapath below to support the `addtm` instruction. You are only allowed to extend existing or add new MUXes, add wires, add new register file read or write ports, and add entries to pipeline registers. Show your modifications directly on the diagram below. Below the diagram, list the changes that you have made.



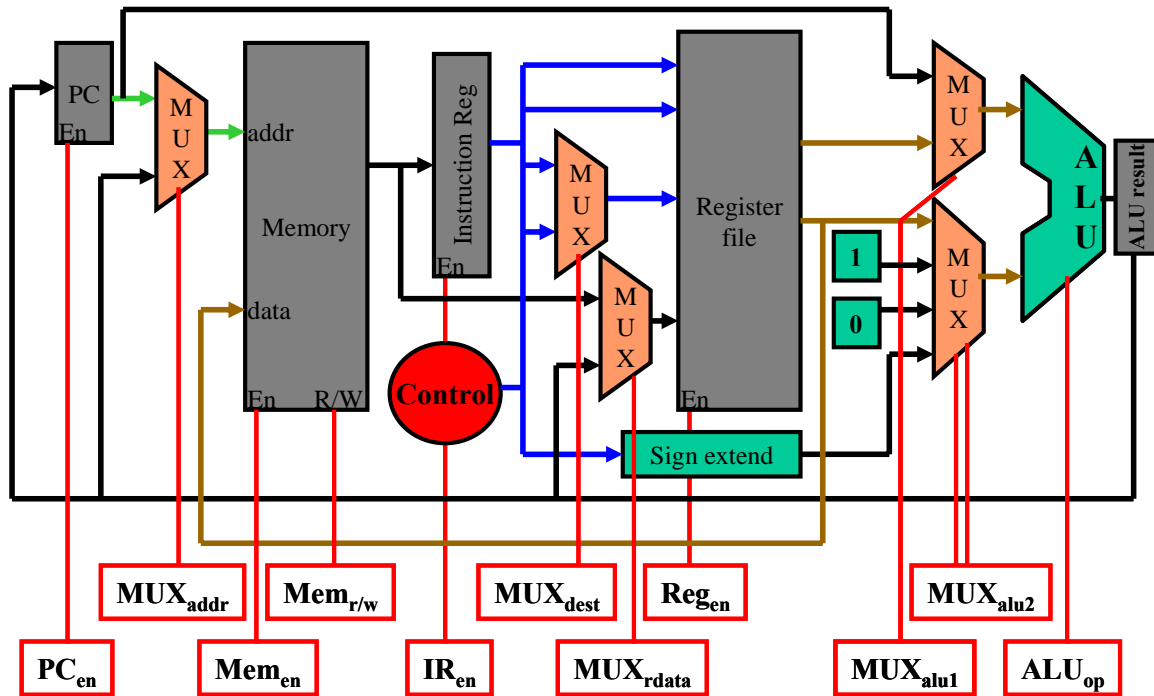
1. Add additional read port to the register file to read `destReg`
2. Pass `destReg` value to ID/EX and EX/MEM
3. Add a mux to address input of data memory, inputs are ALUResult and the `destReg` value from EX/MEM
4. Add another mux to data input of data memory, inputs are `regB` and ALUResult from the EX/MEM

2. [10 points] Another change is to allow operands to come from memory. We want to extend the baseline LC-2K5 pipeline to support add-from-memory, `addfm regA regB destReg`, which has the following semantics: $\text{destReg} = \text{Mem}[\text{regA}] + \text{regB}$. To do this **without stalling the pipeline** requires an additional pipeline stage inserted between the MEM and WB stages as shown below. Fill in the components of this stage, including any wires, MUXes, ALUs that are needed to support the `addfm` instruction. Note that you may also need to add wires and extend or add MUXes to other stages. You can ignore all connections needed to support data forwarding. Hint: you should not add any additional register files or memories. Below the diagram, list the changes that you have made.



1. Add "0" value to bottom mux fo ALU in EX stage
2. Add regB to MEM/NEW pipeline register
3. Add ALU to stage NEW that takes loaded value from the MEM stage and regB
4. Add ALUResult1 (result from the default ALU), ALUResult2 (result from the newly added ALU), and loaded value to NEW/WB pipeline register
5. Widen mux at stage WB to 3-to-1 to mux ALUResult1, ALUResult2, and loaded value.

Part C. Multi-Cycle Datapath [20 points]



Consider the above LC-2K5 multi-cycle datapath covered in lecture given above. We want to provide hardware support for a new instruction:

$$\text{regA} \leftarrow \text{regB}++ + \text{regC}$$

This instruction adds the contents of registers regB and regC and places the sum in register regA. It then increments (i.e., adds 1 to) the contents of regB.

1. [5 points] List any new hardware components that need to be introduced to the LC-2K5 datapath to directly support the new instruction.

Just one wire connecting the regB output of the register file to the upper mux in front of the ALU.

2. [5 points] List any new control signals that need to be introduced to the LC-2K5 control to provide support for the new instruction.

Second control bit for mux.

3. [10 points] Give a cycle by cycle description of the LC-2K5 operation when executing the new instruction. For each cycle, give the following information:

- a. Single-sentence description of what the cycle is about.
- b. Register updates.

Use as few cycles as possible. To get you going, we provide the first 2 cycles.

Cycle 1	Fetch instruction. Instruction Register \leftarrow new instruction ALU Result \leftarrow PC + 1
Cycle 2	Decode instruction and read registers PC \leftarrow PC + 1
Cycle 3	Add regA and regB ALUResult \leftarrow regA + regB
Cycle 4	Load regA + regB onto destReg RegisterFile[destReg] \leftarrow ALUResult Add regB and 1 ALUResult \leftarrow regB + 1
Cycle 5	Load regB + 1 onto regB RegisterFile[regB] \leftarrow ALUResult

Note, you may NOT need to use all the boxes.