# Midterm 2 Review

**EECS 370 – Introduction to Computer Organization - Winter 2016**

**Profs. Valeria Bertacco & Reetu Das**

**EECS Department**
**University of Michigan in Ann Arbor, USA**

**© Bertacco-Das, 2016**
The material in this presentation cannot be
copied in any form without our written permission

# Announcements

- No class on TUESDAY 3/15
- Project 3 due 3/13
- 2z competition winners to be announced on 3/22

# Exam: Time

- **Tuesday March 15, 2016 – 7-8:30pm**

- No Michigan time, exam starts at 7pm sharp
  - Show a few minutes early to your assigned room

- Closed book/notes
  - No phones, computers, calculators, smart watches, nothing electronic
  - You are only allowed
    a) #2 pencils,
    b) eraser and
    c) one letter-size single-sided sheet of paper with any notes you wish to have

- **Bring your Mcard to the exam**

# Exam: Location

- Check your email for the location
  You should have received an email from either myself or Prof. Winsor with your exam location

- Go to your assigned room!
  Exams taken in the wrong room show up as missing from the correct room's roster.

# Preparing for the exam

- Review sessions
  - Today's lecture - Profs
  - Friiday 3/11 6:30 – 8:30pm – DOW 1013 – GSIs/IAs
- Office Hours
  - Thursday 3/10 – 10.30-noon – Prof. Austin
  - Thursday 3/10 – 12:30pm-1pm – Prof. Das
  - Thursday 3/10 – 3-6pm Gabe & 6:30-9:30 Jasmine
  - Friday 3/11 – Anoushe 2-4pm
  - Monday 3/14 – Nathan, Anoushe, Allan, Harry covering all day (check calendar)
  - Tuesday 3/15 – 10.30-noon – Prof. Austin
  - Tuesday 3/15 – 12:30pm-1pm – Prof. Das

# Midterm Material

Covers all material from course start with emphasis on post midterm-1 material

- Emphasis on post floating-point material
- Lectures 8-14
- Related to Project 2, 3
- Covered with Homeworks 3, 4 and 5

# Preparation strategies

- Studying tips
  - Go through the lecture notes, slide by slide
    - Do the class problems without looking at the answers
  - Skim through the book reading material
  - Re-solve homework problems

- Practice with old exams
  - Exams from the past 2 semesters with answers are posted
  - Great practice, get used to “exam-type” questions
  - Note: topics covered may be a bit different
    - e.g. cache questions are not covered in this midterm
  - Create realistic exam setup: time yourself, don’t look at the solution until time is up, use your cheatsheet
    - **Do the old tests without looking at the solutions!**

# Midterm Strategy

- Expect 7-10 problems covering all the material
  - Easier problems first, harder ones towards end
- Pace yourself
- Do not spend all your time on a single problem. If you don't understand a problem, move ahead and come back to it later
- For verbose questions you may want to read on a per-need basis. But do not assume to know what the text says!
- Give yourself a few minutes at the end to check your work.
- Most importantly, don't panic

# Important Topics

- Digital logic design
- How sequential logic timing works
- Design of FSMs
- Singlecycle and Multicycle datapath
- Pipelining
- Data Hazards
- Branch prediction

# Important Topics

- Our timeless favorites (but, we may come up with new ones)
  - Logic design
  - new FSMs
  - Modify datapath to support a new instruction
  - Pipeline simulation
  - Figure out hazards
  - Performance of everything
  - Branch prediction accuracy

- LC2K/ARM code: understand and use
  - Always good to recap

# Select Practice Questions

Just because it is not covered in this review it does not mean that it is not important!

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

1. How many cycles does the code take assuming detect and stall for control hazards?

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

1. How many cycles does the code take assuming detect and stall for control hazards?

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

# Instructions = 100*(0.5*5 + 0.5*4) + 1 = 451

Time = 451 + load stalls + branch stalls + pipe fillup

Time = 451 + 100*0.5*1 +

Time = 451 + 100*0.5*1 + (100*3 + 100*3) + 4

Time = 1105

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

2. How many cycles does the code take assuming speculate and squash where all branches are predicted not taken?

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

2. How many cycles does the code take assuming speculate and squash where all branches are predicted not taken?

```
add   1 2 3
beq   1 5 1
lw    6 4 1
add   3 4 5
beq   5 7 -5
halt
```

# Instructions = 100*(0.5*5 + 0.5*4) + 1 = 451

Time = 451 + load stalls + branch stalls + pipe drain

Time = 451 + 100*0.5*1 + (100*0.5*3 + 99*3) + 4

Time = 952

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

3. How many cycles does the code take assuming speculate and squash where backward branches are predicted taken and forward branches not taken?

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

# Calculating performance with control hazards

Assume the first branch is taken 50% of the time and the loop iterates 100 times, and forwarding for all data hazards.

3. How many cycles does the code take assuming speculate and squash where backward branches are predicted taken and forward branches not taken, andBTB has all entrys to start?

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

# Instructions = 100*(0.5*5 + 0.5*4) + 1 = 451

Time = 451 + load stalls + branch stalls + empty pipe

Time = 451 + 100*0.5*1 + (100*0.5*3 + 1*3) + 4

Time = 658

## Calculating performance with control hazards

Assume the first branch has the pattern **TTTN** that repeats, and the loop is iterated 100 times

4. How many cycles does the code take if a 2-bit counter BTB is used to predict each branch, how many cycles does the code take? Assume initial state of branch predictor counter is "10" (WT)

```
add    1  2  3
beq    1  5  1
lw     6  4  1
add    3  4  5
beq    5  7  -5
halt
```

## Calculating performance with control hazards

Assume the first branch has the pattern **TTTN** that repeats, and the loop is iterated 100 times

4. How many cycles does the code take if a 2-bit counter BTB is used to predict each branch, how many cycles does the code take? Assume initial state of branch predictor counter is "10" (WT)

```
add   1 2 3
beq   1 5 1
lw    6 4 1
add   3 4 5
beq   5 7 -5
halt
```

√ √ √ X √ √ √ X √ √ √ X

**beq 1** T T T N T T T N T T T N ...

**beq 2** is correct 99 times, then incorrect last iter

# Instructions = 100*(0.25*5 + 0.75*4) + 1 = 426

Time = 426 + load stalls + branch stalls + empty pipe

Time = 426 + 100*0.25*1 + 100*0.25*3 + 1*3 + 4

Time = 533

# Classic performance problem

- Program with following instruction breakdown:

  | | |
  |---|---|
  | lw | 10% |
  | sw | 15% |
  | beq | 25% |
  | R-type | 50% |

- Speculate "always not-taken" and squash. 80% of branches not-taken
- Full forwarding to execute stage. 20% of loads stall for 1 cycle
- What is the CPI of the program?
- What is the total execution time if cycle time is 100MHz?

# Classic performance problem

- Program with following instruction breakdown:

  | | |
  |---|---|
  | lw | 10% |
  | sw | 15% |
  | beq | 25% |
  | R-type | 50% |

- Speculate "always not-taken" and squash. 80% of branches not-taken
- Full forwarding to execute stage. 20% of loads stall for 1 cycle
- What is the CPI of the program?
- What is the total execution time if cycle time is 100MHz?

**CPI = 1 + 0.10 (loads) * 0.20 (load use stall)*1**
**+ 0.25 (branch) * 0.20 (miss rate)*3**
**CPI = 1 + 0.02 + 0.15 = 1.17**
**Time = 1.17 * 10ns = 11.7ns**

# Classic performance problem (cont.)

- Assume branches are resolved at Execute?
  - What is the CPI?
  - What happens to cycle time?
  - What is the total execution time?

- What if branches are resolved at Decode?

# Classic performance problem (cont.)

- Assume branches are resolved at Execute?
  - What is the CPI?
  - What happens to cycle time?
  - What is the total execution time?

**CPI = 1 + 0.10 (loads) *0.20 (load use stall)*1**
**+ 0.25 (branch) * 0.20 (miss rate)*2**
**CPI = 1 + 0.02 + 0.1 = 1.12**

- What if branches are resolved at Decode?

**CPI = 1 + 0.10 (loads) *0.20 (load use stall)*1**
**+ 0.25 (branch) * 0.20 (miss rate)*1**
**CPI = 1 + 0.02 + 0.05 = 1.07**

## Performance with deeper pipelines

- Assume the setup of the previous problem.
- What if we have a 10 stage pipeline?
  - Instructions are fetched at stage 1.
  - Register file is read at stage 3.
  - Execution begins at stage 5.
  - Branches are resolved at stage 7.
  - Memory access is complete in stage 9.
- What's the CPI of the program?
- If the clock rate was doubled by doubling the pipeline depth, is performance also doubled?

## Performance with deeper pipelines

- Assume the setup of the previous problem.
- What if we have a 10 stage pipeline?
  - Instructions are fetched at stage 1.
  - Register file is read at stage 3.
  - Execution begins at stage 5.
  - Branches are resolved at stage 7.
  - Memory access is complete in stage 9.
- What's the CPI of the program?
- If the clock rate was doubled by doubling the pipeline depth, is performance also doubled?

**CPI = 1 + 0.10 (loads) *0.20 (load use stall)*4 + 0.25 (branch) * 0.20 (N stalls)*6**

**CPI = 1 + 0.08 + 0.30 = 1.38**

**Time = 1.38 * 5ns = 6.9 ns**

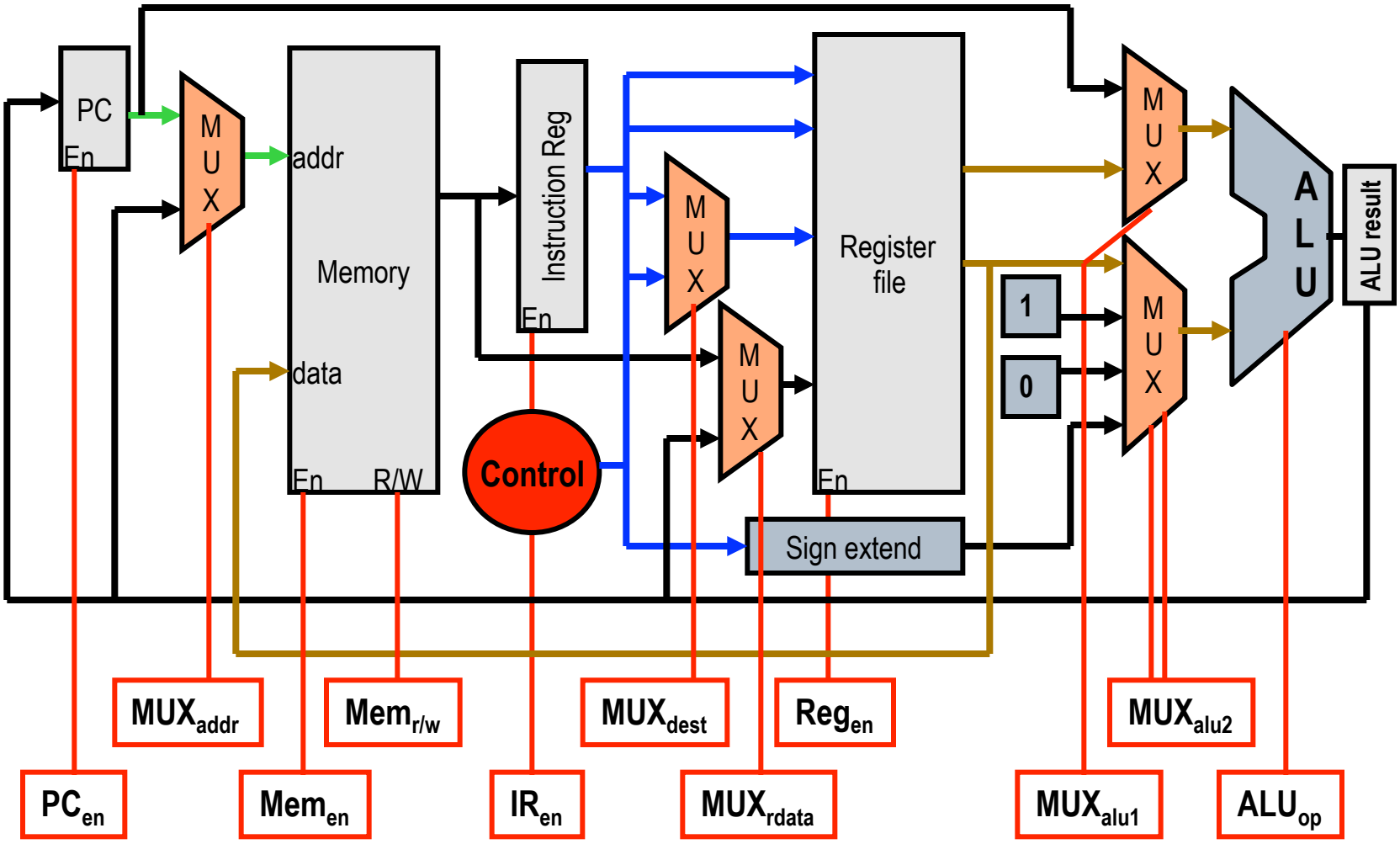
# Multicycle Datapath – F05E2 (Question II.C)

Consider the above LC-2K multicycle datapath covered in lecture. We want to provide hardware support for a new instruction:

destReg = regA++ + regB

1. List any new hardware components that need to be introduced to the LC-2K datapath to directly support the new instruction.

2. List any new control signals that need to be introduced to the LC-2K control to provide support for the new instruction.


PC
En
M
U
X
addr
Memory
data
En
R/W
Instruction Reg
En
M
U
X
M
U
X
Register
file
En
Control
Sign extend
1
0
M
U
X
M
U
X
A
L
U
ALU result
MUX_addr
Mem_r/w
MUX_dest
Reg_en
MUX_alu2
PC_en
Mem_en
IR_en
MUX_rdata
MUX_alu1
ALU_op

# Multicycle Datapath – F05E2 (Q. II.C cont.)

Give a cycle by cycle description of the LC-2K operation when executing the new instruction. For each cycle, give the following information: Single-sentence description of what the cycle is about, and Register updates. Use as few cycles as possible.

| | |
|---|---|
| Cycle 1 | Fetch instruction<br>Instruction Register = new instruction<br>ALU Result = PC + 1 |
| Cycle 2 | Decode instruction and read registers<br>PC = PC + 1 |
| Finish the rest | |

# Multicycle Datapath – F05E2 (Q. II.C cont.)

| | |
|---|---|
| Cycle 1 | Fetch instruction<br>Instruction Register = new instruction<br>ALU Result = PC + 1 |
| Cycle 2 | Decode instruction and read registers<br>PC = PC + 1 |

# Question II.C 3 - answer

| | |
|---|---|
| Cycle 1 | // Fetch instruction<br>Instruction Register = new instruction<br>ALU Result = PC + 1 |
| Cycle 2 | // Decode instruction and read registers<br>PC = PC + 1 |
| Cycle 3 | // Add regA and regB<br>ALUResult = regA + regB |
| Cycle 4 | // Store ALUResult to destR<br>RegisterFile[destR] = ALUResult<br>// Add regA and 1<br>ALUResult = regA + 1 |
| Cycle 5 | // Store ALUResult to regA<br>RegisterFile[regA] = ALUResult |

# Question II.C 1 and 2 - answers

1. Add a entry to the MUX connecting to the register write address port to the register file, making it 3-1 MUX. Connect the instruction_reg[21:19] (the regA field) to the new MUX input.

2. A second control bit is required for the new MUX.

# Pipelining – F05E2 (Question I.5)

```
        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
```

Consider running the following assembly program on the 5-stage, LC-2K 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?

# Question 1.5 answer

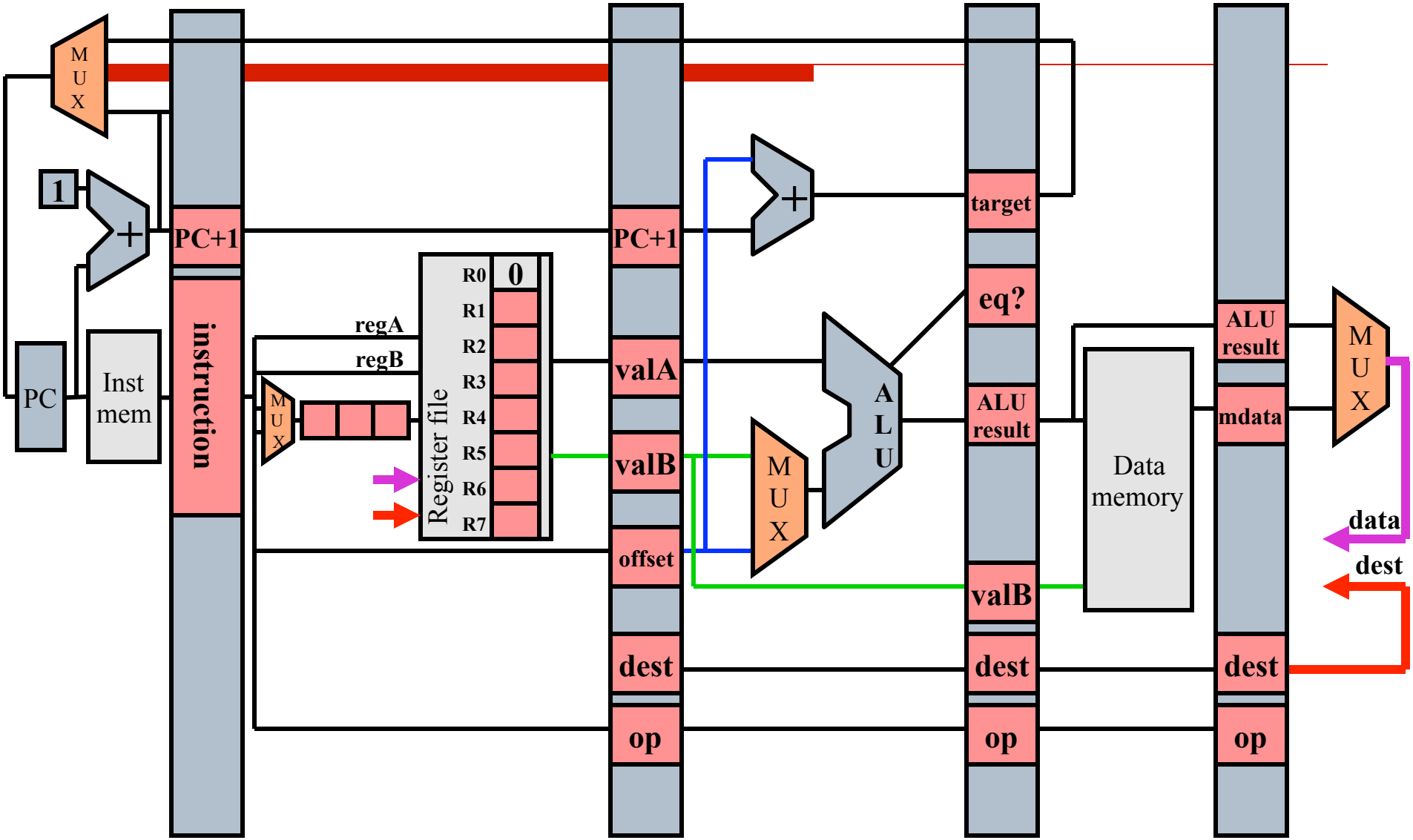
```
            lw      0       1       neg1
            2 nops due to use of reg1 by next instruction
            lw      1       3       neg1
            2 nops due to use of reg3 by next instruction
begin       beq     0       3       exit
            3 nops due to control hazard caused by previous beq
            add     1       3       3
            lw      2       4       val
            2 nops due to use of reg4 by next instruction
            (note 1 of nops caused by use of reg3 defined by the add)
            nand    3       4       4
            2 nops due to use of reg4 by next instruction
            sw      2       4       loc
            beq     0       0       begin
             3 nops due to control hazard caused by previous beq
exit        halt
val         .fill 10
neg1        .fill -1
loc         .fill 0
```

14 total nops inserted,
8 for data hazards,
6 for control hazards

# Pipelining – F05E2 (Question II.B)

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: Mem[destReg] = regA + regB.

1. Extend the LC-2k 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.


M
U
X
1
+
PC+1
PC
Inst
mem
instruction
M
U
X
regA
regB
Register file
R0
R1
R2
R3
R4
R5
R6
R7
0
PC+1
valA
valB
offset
dest
op
+
M
U
X
A
L
U
target
eq?
ALU
result
valB
dest
op
Data
memory
ALU
result
mdata
dest
op
M
U
X
data
dest
IF/
ID
ID/
EX
EX/
Mem
Mem/
WB

# Question II.B 1 answer

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

# Pipelining

Using the LC2k pipeline with data forwarding support, for the following code sequence, which pipeline registers will contain the operands for the add?

```
lw 1 3 A
lw 3 4 B
add 3 4 3
sw 1 3 C
halt
```

Answer: reg 3 from ID/EX, reg4 from Mem/WB