CS 152
Computer Architecture and Engineering

Lecture 10 – Midterm I Review Session

2006-9-28

John Lazzaro
(www.cs.berkeley.edu/~lazzaro)

TAs: Udam Saini and Jue Sun

www-inst.eecs.berkeley.edu/~cs152/

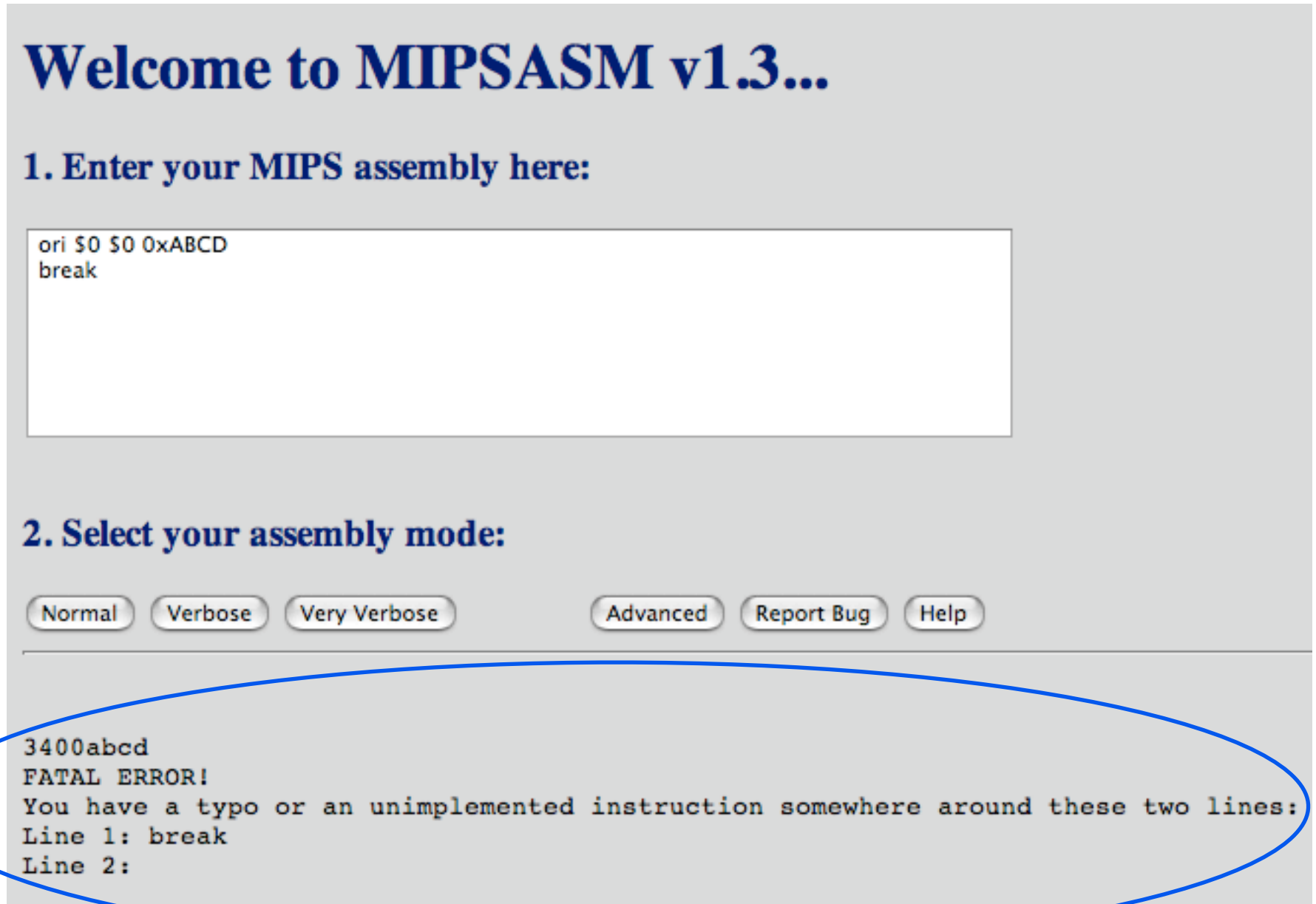
The Break Instruction

Lab 3: ISA Specifications

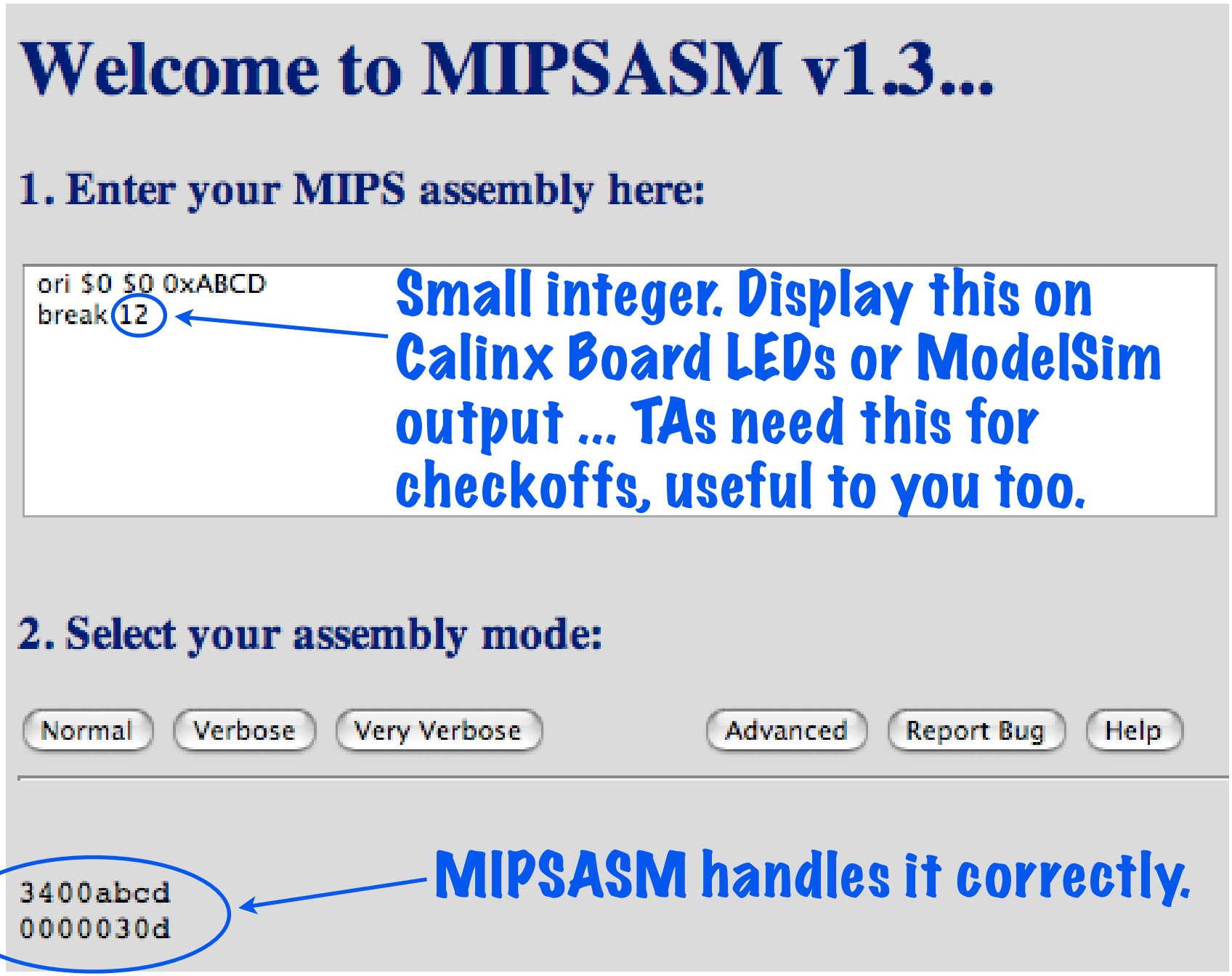
Type	Instructions
arithmetic	addu, subu, addiu
logical	and, andi, or, ori, xor, xori, lui
shift	sll, sra, srl
compare	slt, slti, sltu, sltui
control	beq, bne, bgez, bltz, j, jr, jal
data transfer	lw, sw
Other:	break

The **break** instruction is special. See COD for its bitfield. Although this is normally an exception-causing instruction, you should treat it more like a halt instruction. After being *decoded*, the break instruction should freeze the pipeline from advancing further. This means that the PC will not advance further, the break instruction will stay in the decode stage, and later instructions will drain from the pipeline as they complete. The proper terminology for this is that the **break** instruction will "stall" in the decode stage. Assume that there will be a single input signal called "release" that comes from outside. When it is high, you should release a blocked **break** instruction exactly once (you need to

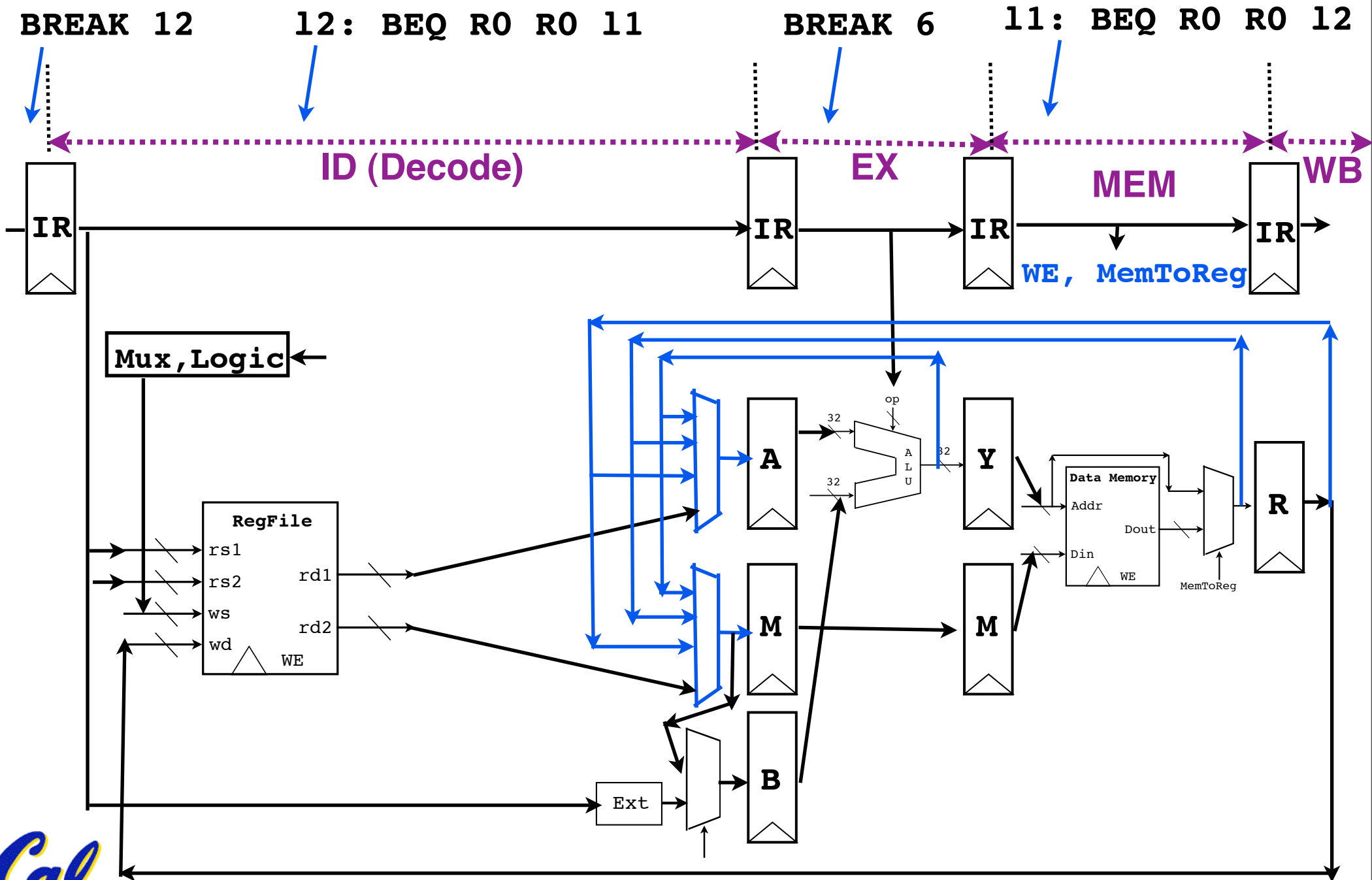
MIPSASM doesn't like it ???



A feature, not a bug ...



Recall ...



Today - Midterm I Review Session

- HW 1, problem by problem ...
  - Recall: HW 1 was Fall 05 Mid-term I.
- Study tips, test ground rules
- All questions answered (almost ...)

CS152 Midterm I
October 4th 2005

Name: _____

SSID: _____

"All the work is my own. I have no prior knowledge of the exam contents, aside from guidance from class staff. I will not share the contents with others in CS152 who have not taken it yet."

Signature: _____

Please write clearly, and put your name on each page. Please abide by word limits. Good luck!

David Marquardt
Udam Saini
John Lazzaro

#	Points	
1	10	
2	15	
3	10	
4	10	
5	15	
6	15	
7	10	
8	15	
Tot	100	

CS152 Homework I, Fall 2006

Name: Solution Sheet

SSID: _____

#	Points	
1	10	10
2	15	15
3	10	10
4	10	10
5	15	15
6	15	15
7	10	10
8	15	15
Tot	100	100

Homework I is due in class on Thursday September 28 at 11:10 AM. This class is the Mid-term I review session.

Late homeworks are NOT accepted. Thus, if you will not be attending the review session, you MUST make arrangements to hand off the homework to the instructor before classtime.

Homework will be graded on effort (did you make an honest attempt to solve each problem?), not correctness. We will distribute the correct answers for the homework in the review session, but we will probably not return the homework you hand in until after the exam. So, you may wish to make a copy for reference before you hand it in.

This homework will count for approximately 1.5% of your final grade. The homework is based on the Mid-term I exam from Fall 05. You may discuss the homework problems with fellow students and the TAs, but what you write down must be your own work (no copying the answers from someone else's homework). Good luck! John Lazzaro

Part I - Single Cycle

Mid-term, Part I: ISA & Single Cycle

Typical Topics

Modify a single-cycle CPU to match an unusual ISA change.

Design a CPU component (ex: register file) for a novel ISA.

Analyze if a CPU design matches a proposed ISA.

Write machine language for an unusual ISA.

T 8/29	The MIPS ISA
W 8/30	
Th 8/31	Single-Cycle Datapaths
F 9/1	
Sa 9/2	
Su 9/3	
M 9/4	(Labor Day Holiday)
T 9/5	Single-Cycle Wrap-Up + VLIW

#	Points	
1	10	
2	15	
3	10	
4	10	
5	15	
6	15	
7	10	
8	15	
Tot	100	

Mid-term, Part I: How to do well …

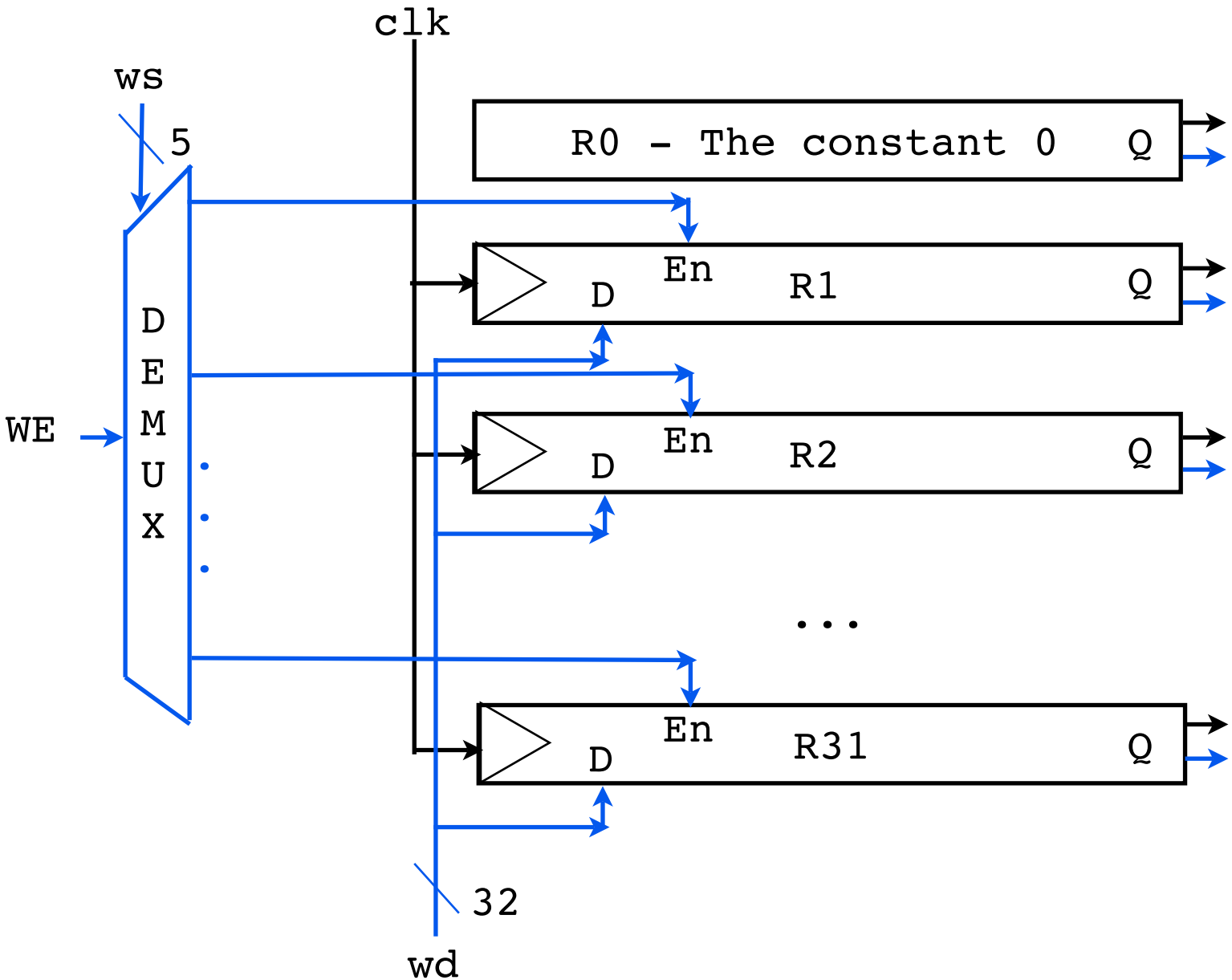
T 8/29	The MIPS ISA
W 8/30	
Th 8/31	Single-Cycle Datapaths
F 9/1	
Sa 9/2	
Su 9/3	
M 9/4	(Labor Day Holiday)
T 9/5	Single-Cycle Wrap-Up + VLIW

- Problem intro often features a lecture slide. If you have to teach yourself that slide during the test, you're starting out behind.
- Getting the problem correct requires thinking on your feet to do a new design or analyze one given to you.

Just like an on-site job interview …

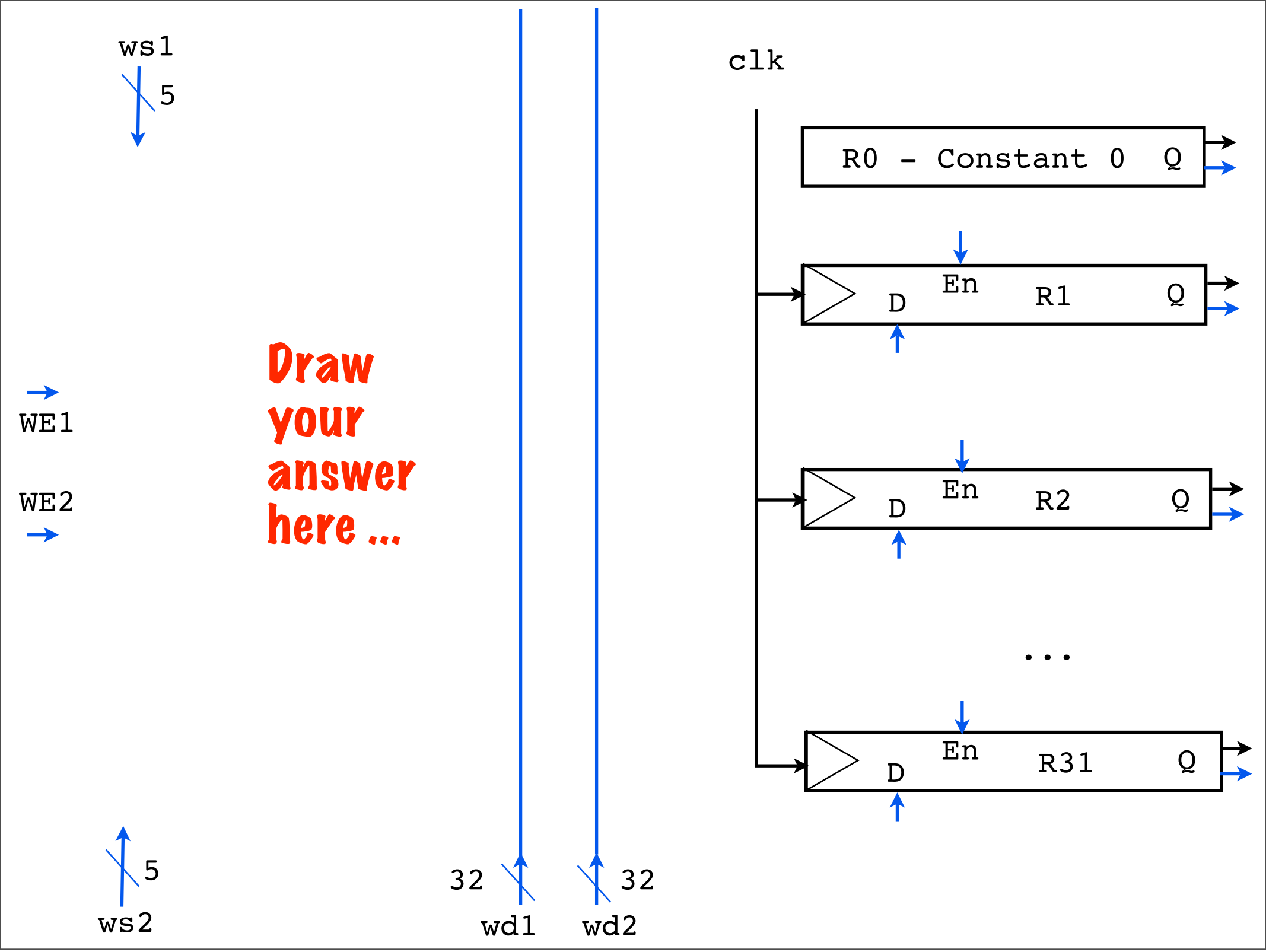
Q1. Register File Design (10 points)

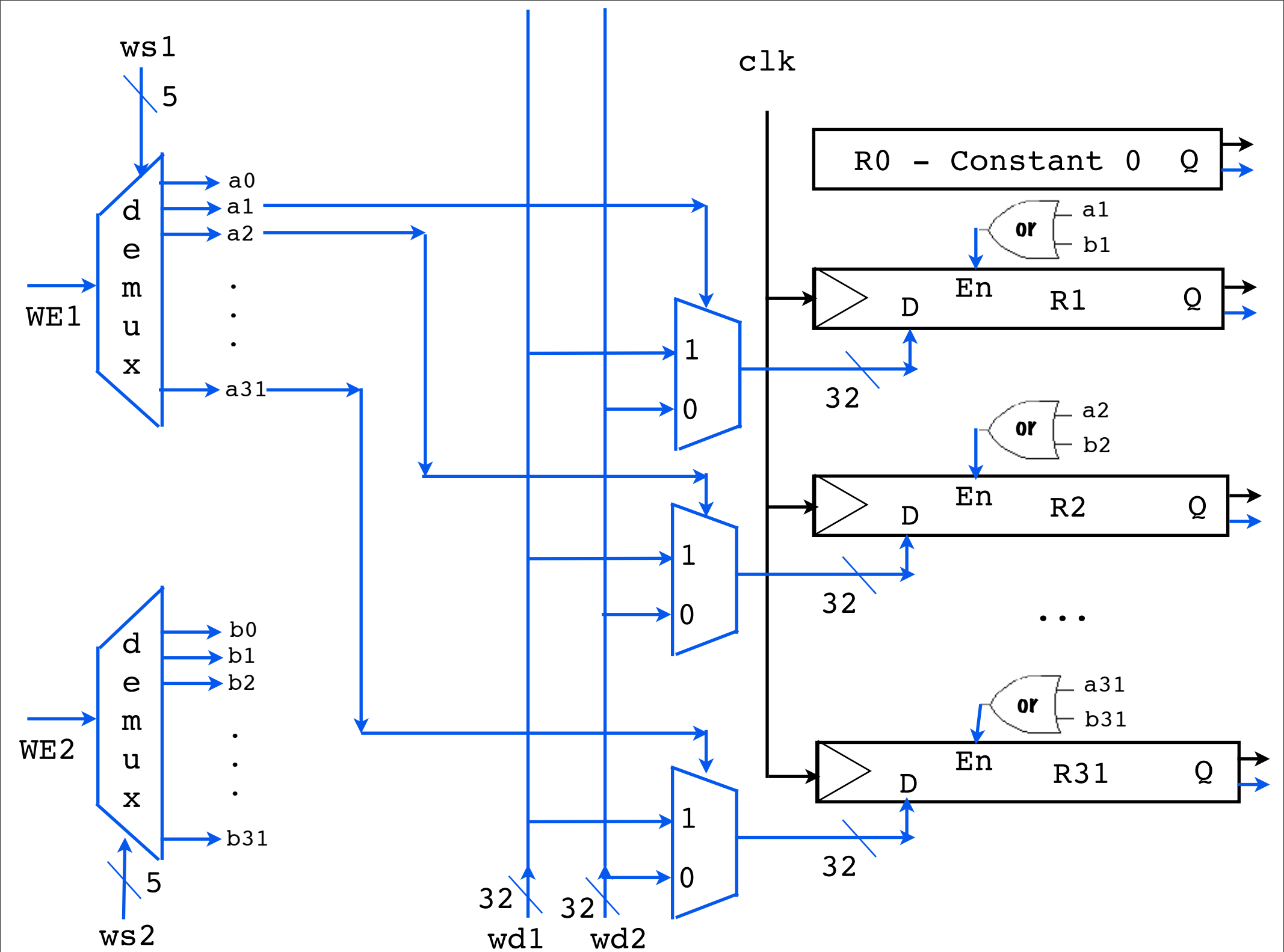
On the top slide on the next page, we show the write logic for the register file design we showed in Lecture 1-2.



Q1: The actual question ...

Redesign the write logic for the register file, so that two registers may be written on the same positive clock edge. The 5-bit values ws1 and ws2 specify the registers to write, the 1-bit values WE1 and WE2 enable writing for each port (1 = enabled, 0 = disabled), and the 32-bit values wd1 and wd2 are the data to be written. If both write ports are enabled, and ws1 and ws2 specify the same register, this register MUST be written with the value wd1.


ws1
5
clk
R0 - Constant 0 Q
En
D
R1
Q
WE1
WE2
Draw your answer here ...
En
D
R2
Q
...
En
D
R31
Q
5
ws2
32
wd1
32
wd2


ws1
5
clk
R0 - Constant 0 Q
a0
a1
a2
a31
d
e
m
u
x
WE1
or
a1
b1
En
D
R1
Q
1
0
32
or
a2
b2
En
D
R2
Q
1
0
32
...
b0
b1
b2
b31
WE2
or
a31
b31
En
D
R31
Q
1
0
32
5
ws2
32
wd1
32
wd2

Q2: Single Cycle Design (part A)

Below, we show a slightly-modified version of the single-cycle datapath that we derived in the first weeks of class. On the following pages, we ask questions about this design.

Syntax:

```
LWA $rt imm($rs)
```

LWA: Load Word and Auto-update Index

Actions:

```
$rt = M[$rs + sign_extended(imm)]
$rs = $rs + sign_extended(imm)
```

Is the datapath, as shown, able to execute LWA?

Circle YES or NO below (X points):

YES NO

LWA $rt imm($rs)

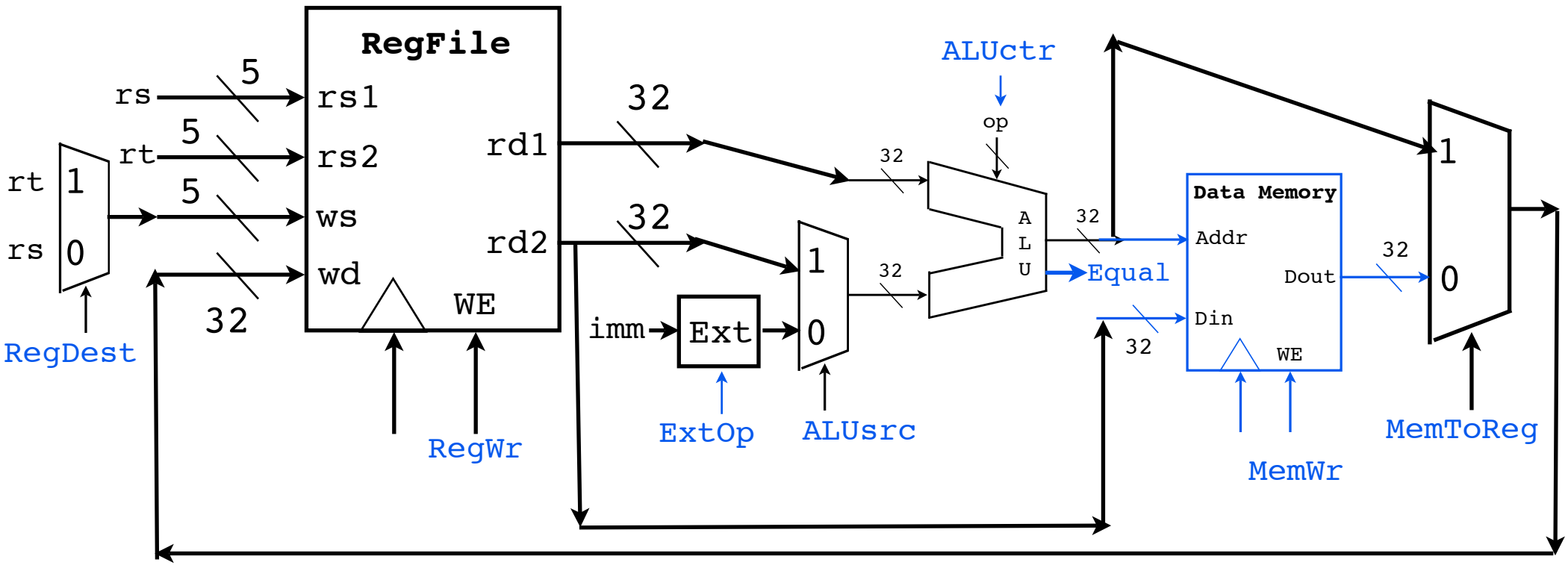
$rt = M[$rs + sign_extended(imm)]
$rs = $rs + sign_extended(imm)

R-format

op	rs	rt	rd	shamt	funct

I-format

op	rs	rt	imm



Mux control: 0 is lower mux input, 1 upper mux input
RegWr, MemWr: 1 = write, 0 = no write.
ExtOp: 1 =sign-extend, 0 = zero-extend.

Q2: Single Cycle Design (part A)

```
LWA $rt imm($rs)        $rt = M[$rs + sign_extended(imm)]
                        $rs = $rs + sign_extended(imm)
```

Is the datapath, as shown, able to execute LWA?

Circle YES or NO below (X points):

```
YES          -------
            |  NO   |
             -------
```

If the answer is no, precisely state how to modify the datapath to support the instruction, in 30 words or less. Write this answer below:

Answer: Replace the register file with a register file with 2 write ports (for example, the design in Problem 1), and wire up the ws1, ws2, wd1, and wd2 inputs so that you can write the registers coded by $rt and $rs with the values specified by the LWA instruction.

Q2: Single Cycle Design (part B)

Below, we show a slightly-modified version of the single-cycle datapath that we derived in the first weeks of class. On the following pages, we ask questions about this design.

Syntax:

```
SWA $rt imm($rs)
```

SWA: Store Word and Auto-update Index

Actions:

```
M[$rs + sign_extended(imm)] = $rt
$rs = $rs + sign_extended(imm)
```

Is the datapath, as shown, able to execute SWA?

Circle YES or NO below (X points):

YES NO

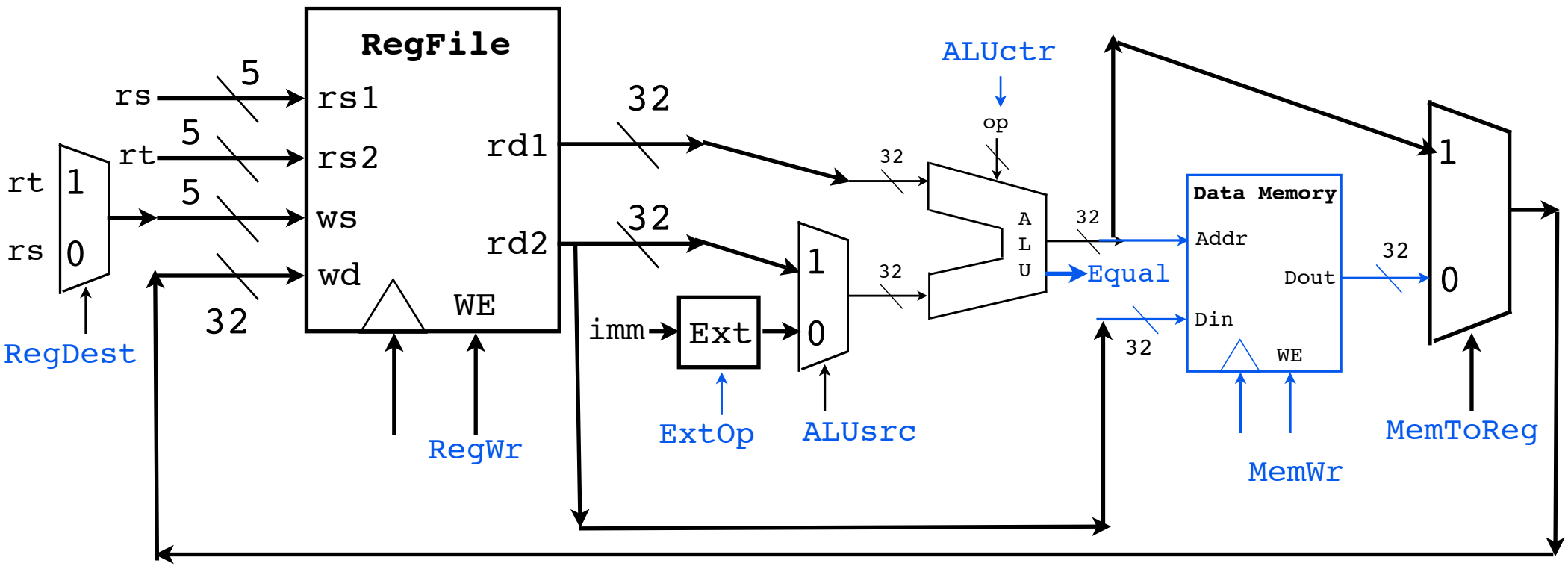
SWA $rt imm($rs)	M[$rs + sign_extended(imm)] = $rt $rs = $rs + sign_extended(imm)

R-format

op	rs	rt	rd	shamt	funct

I-format

op	rs	rt	imm



Mux control: 0 is lower mux input, 1 upper mux input
RegWr, MemWr: 1 = write, 0 = no write.
ExtOp: 1 =sign-extend, 0 = zero-extend.

Q2: Single Cycle Design (part B)

`SWA $rt imm($rs)`	`M[$rs + sign_extended(imm)] = $rt` `$rs = $rs + sign_extended(imm)`

Is the datapath, as shown, able to execute SWA?

Circle YES or NO below (X points):

```
--------
|  YES |                    NO
--------
```

BRSrc	PCSrc	RegDest	RegWr	ExtOp	ALUSrc	MemWr	MemToReg
X	1	0	1	1	0	1	1

Express ALU function below as a function of "A" (upper input) and "B" (lower input). Specify if equation is boolean or numeric, specify if constants are in decimal or hex.

Function for ALU: Simple addition (a + b)

Q2: Single Cycle Design (part C)

Below, we show a slightly-modified version of the single-cycle datapath that we derived in the first weeks of class. On the following pages, we ask questions about this design.

Syntax:

```
BEQR $rs imm $rt
```

BEQR: Branch if EQual to address in Register

Actions:

```
if ($rs == sign_extended(imm))
   PC = PC + 4 + $rt
else
   PC = PC + 4
```

Is the datapath, as shown, able to execute BEQR?

Circle YES or NO below (X points):

YES NO

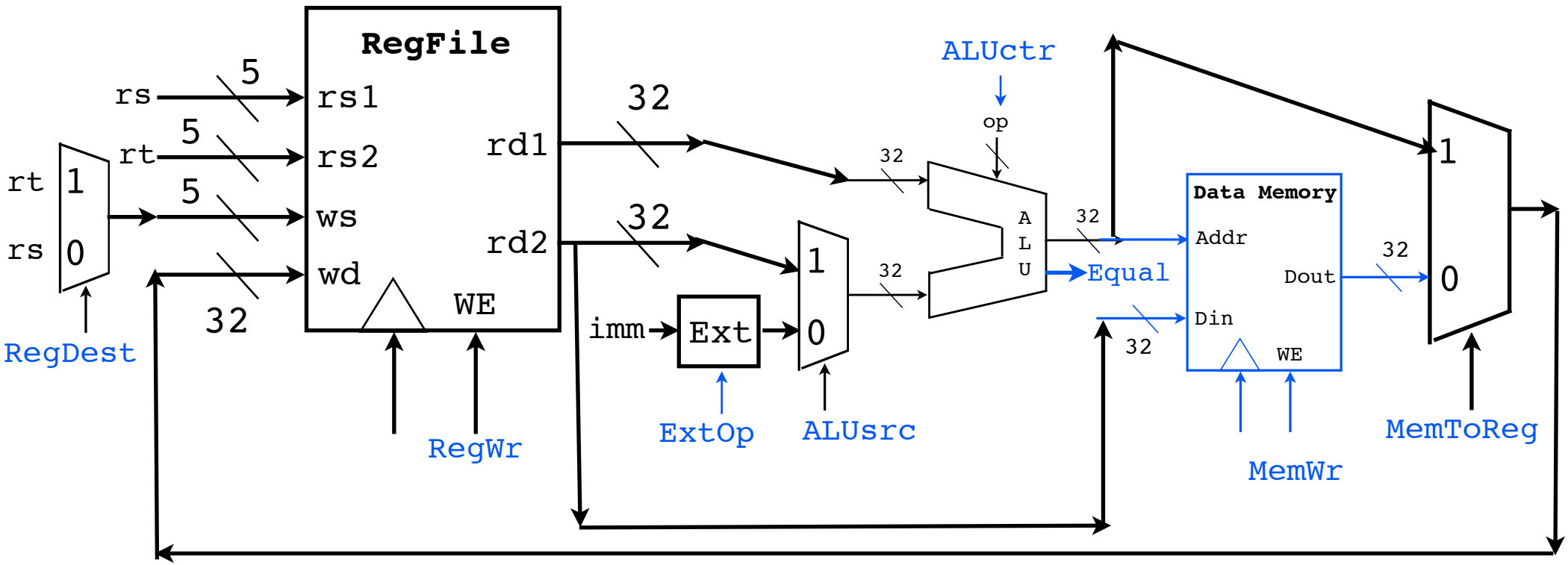
```
BEQR $rs imm $rt     if ($rs == sign_extended(imm))
                        PC = PC + 4 + $rt
                     else
                        PC = PC + 4
```

R-format

op	rs	rt	rd	shamt	funct

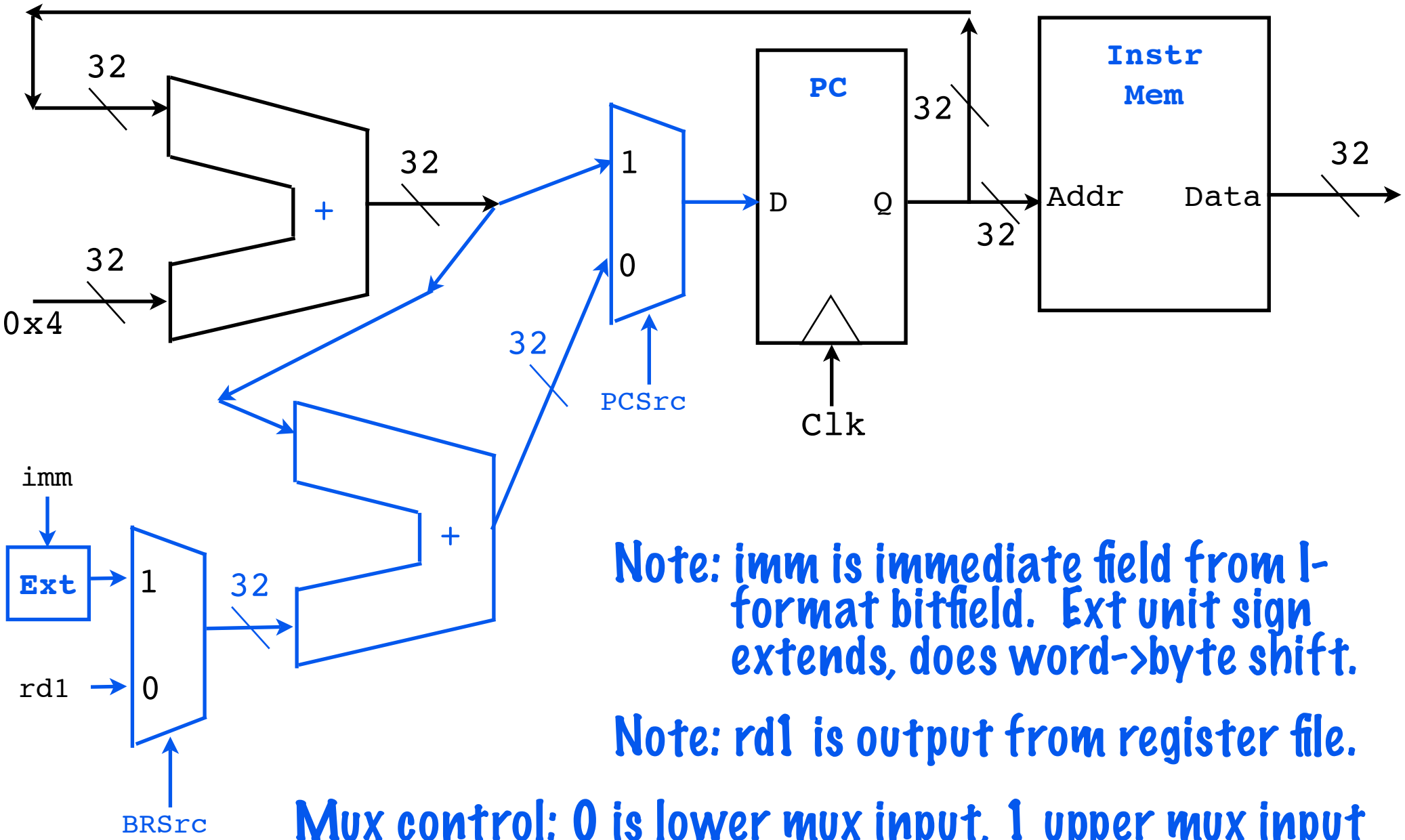
I-format

op	rs	rt	imm



Mux control: 0 is lower mux input, 1 upper mux input
RegWr, MemWr: 1 = write, 0 = no write.
ExtOp: 1 =sign-extend, 0 = zero-extend.

```
BEQR $rs imm $rt     if ($rs == sign_extended(imm))
                        PC = PC + 4 + $rt
                     else
                        PC = PC + 4
```



Note: imm is immediate field from I-format bitfield. Ext unit sign extends, does word->byte shift.

Note: rd1 is output from register file.

Mux control: 0 is lower mux input, 1 upper mux input

Q2: Single Cycle Design (part C)

```
BEQR $rs imm $rt        if ($rs == sign_extended(imm))        else
                           PC = PC + 4 + $rt                     PC = PC + 4
```

Is the datapath, as shown, able to execute BEQR?

Circle YES or NO below (X points):

```
                                  -------
YES                              |  NO   |
                                  -------
```

If the answer is no, precisely state how to modify the datapath to support the instruction, in 30 words or less.

Answer: We need to take output rd2 from the register file and add it as a new input to the BRSrc mux in the instruction fetch slide.

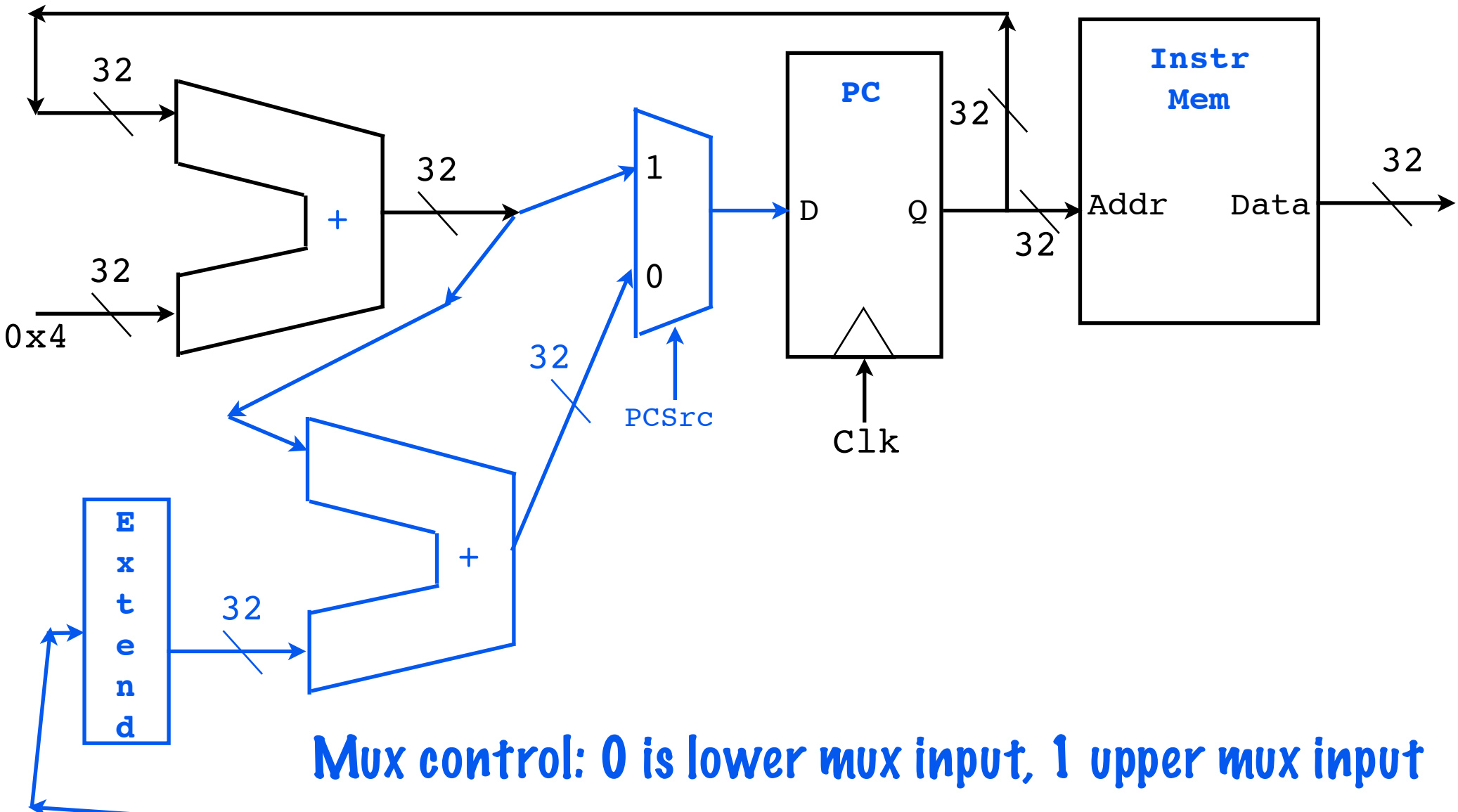
Q3: Single-Cycle Branch Delay Slot

3 Branch Delay Slot (10 points)

On the top slide on the next page, we show the branch logic for the single-cycle processor showed in Lecture 1-2. Recall that this processor does not use the branch delay slot.

Redesign this datapath so that branches have a branch delay slot. You may not change the main controller to add new signals; instead, you must generate your own control signals from local logic and posedge flip-flops. The math for computing the branch target address (PC + 4 + ext(imm), where PC is the branch instruction address) is unchanged from the no-delay-slot case.

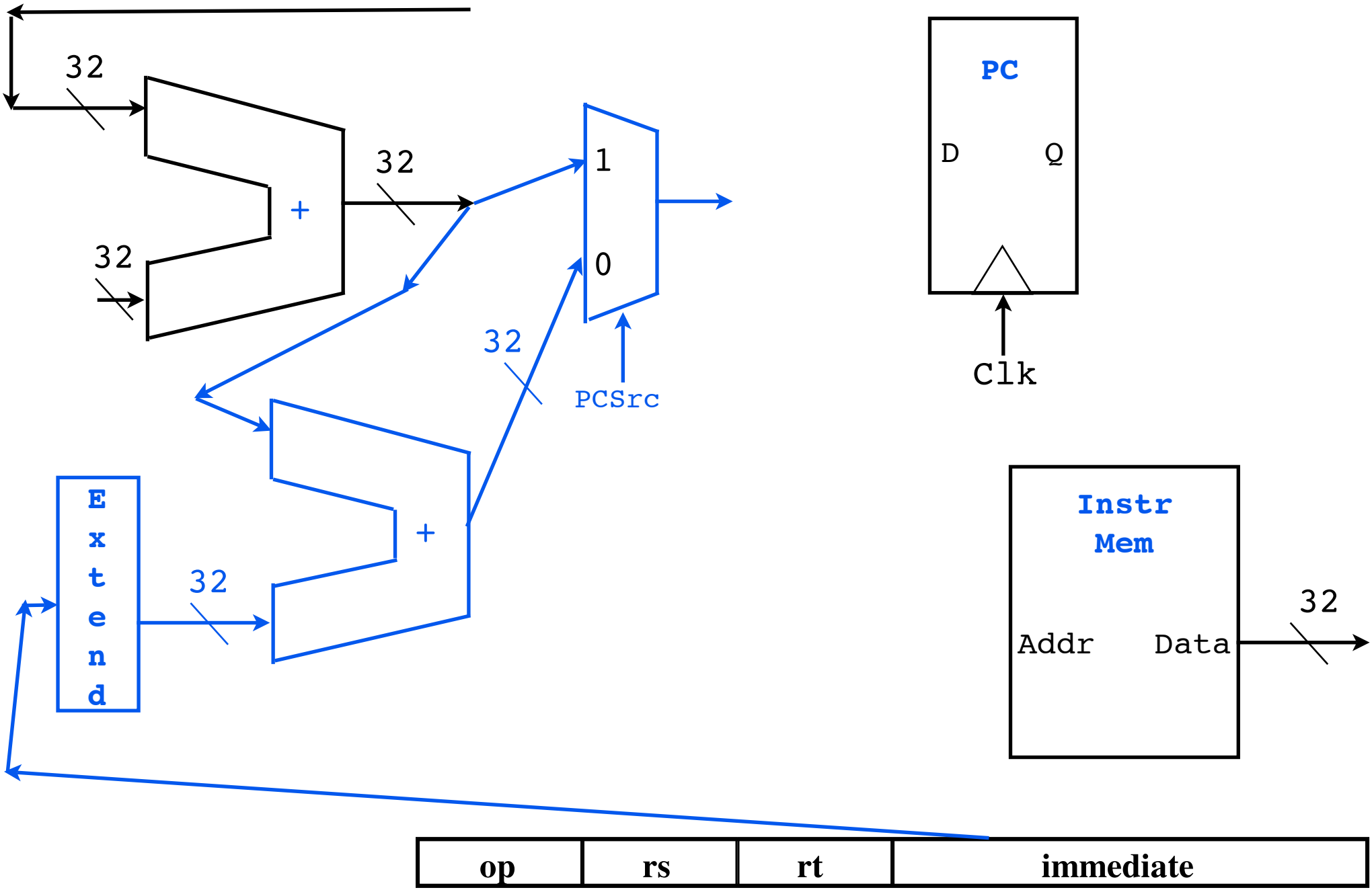
Single-Cycle I-Fetch (no delay slot)



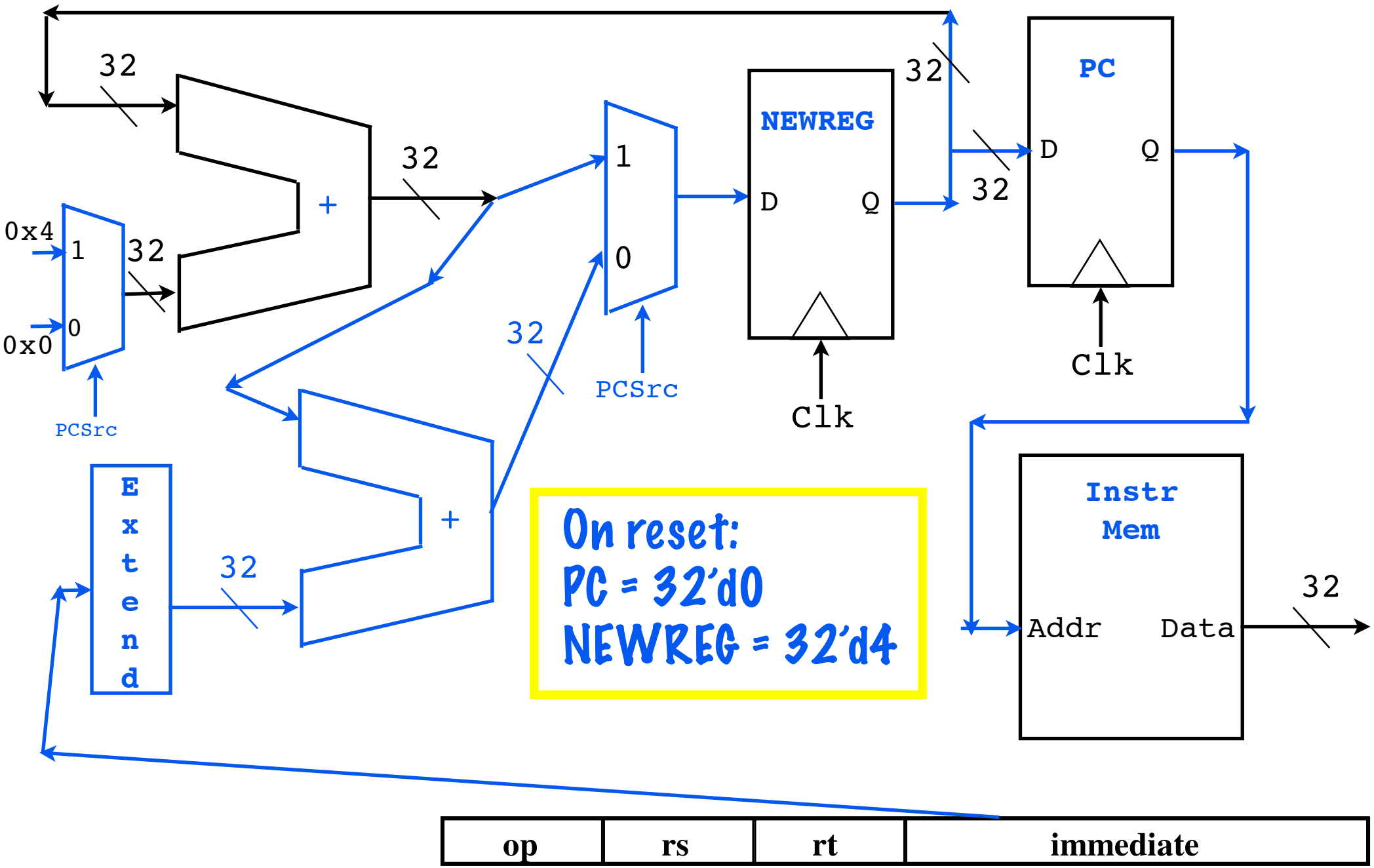
Mux control: 0 is lower mux input, 1 upper mux input

op	rs	rt	immediate

Design Instruction Fetch WITH delay slot



Design Instruction Fetch WITH delay slot



Part II - Design Toolkit

Mid-term, Part II: Design Toolkit

Typical Topics

Might be several "design toolkit" problems this year ... or not ...

Compare energy use of several CPU system designs.

Specify an FPGA routing net that minimizes a critical path.

Use Amdahl's Law to evaluate a parallel program

Given a state machine, specify a test method.

Th 9/7	Testing and Teamwork
F 9/8	
Sa 9/9	
Su 9/10	
M 9/11	
T 9/12	Timing
W 9/13	
Th 9/14	Performance and Energy

#	Points	
1	10	
2	15	
3	10	
4	10	
5	15	
6	15	
7	10	
8	15	
Tot	100	

Mid-term, Part II: How to do well …

Th 9/7	Testing and Teamwork
F 9/8	
Sa 9/9	
Su 9/10	
M 9/11	
T 9/12	Timing
W 9/13	
Th 9/14	Performance and Energy

- **No memorization:** if we ask about Amdahl's Law, we will show its definition lecture slide.
- **Understanding is needed:** A problem may require you to apply equation to a design, etc.
- You may need to do: algebra, simple calculus (ex: maximize f(x)), add a few numbers by hand, etc.

Note: no calculators permitted.

Q4: Interpreting Schmoo Plots ...

4 Energy (10 points)

Below, we show the Schmoo plot for a processor, and remind you of the definitions of power and energy.

In this problem, the processor characterized by the Schmoo plot is used in a system along with support components that use K Watts of power. For example, in a laptop, the support chips might consume 2 Watts. For this system, $K = 2$.

When the processor is running, the support components must stay on. A CPU instruction may be used to turn the processor and the support components off. When off, the processor and the support components both use no power at all.

A "Schmoo" plot for a Cell SPU ...

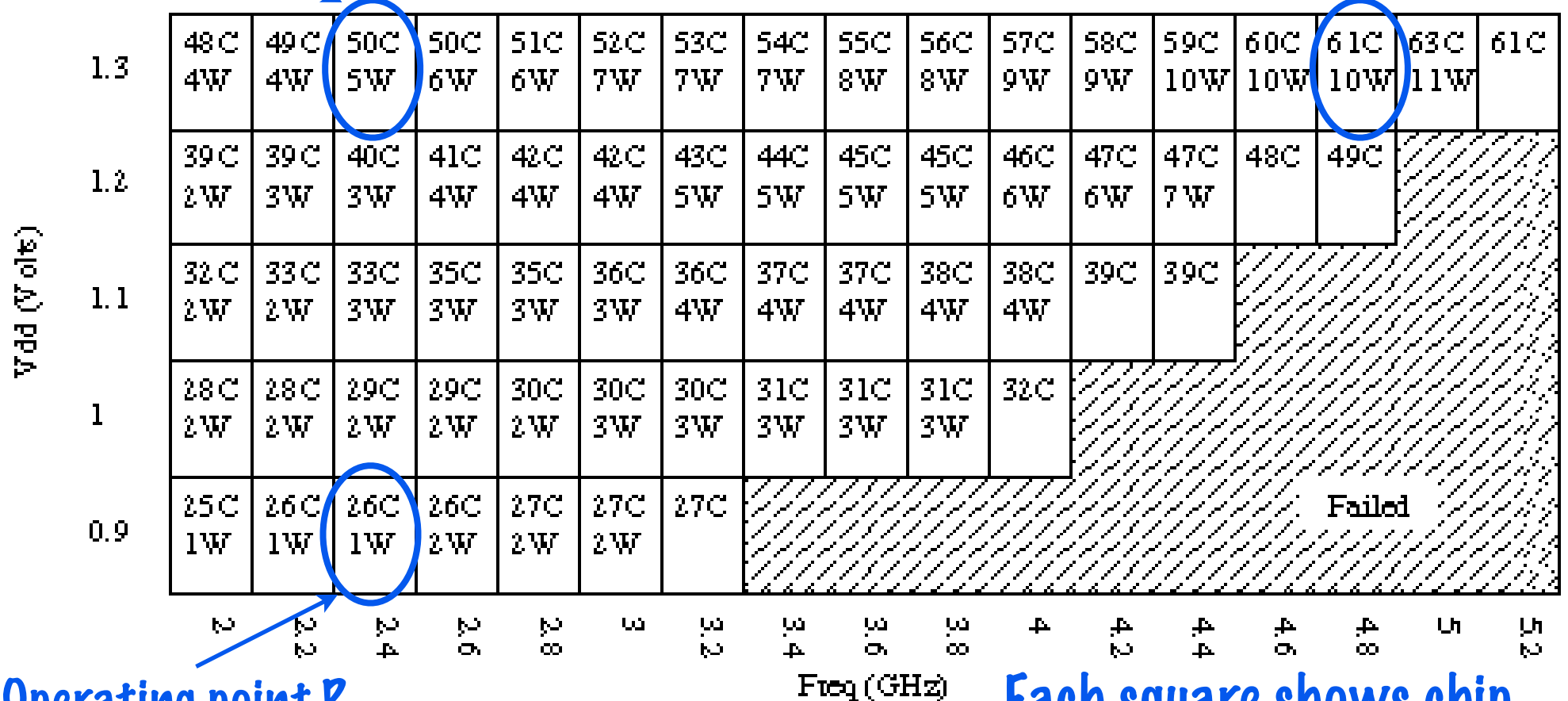
The energy equations: $E_{0->1} = \frac{1}{2} C V_{dd}^2$ $E_{1->0} = \frac{1}{2} C V_{dd}^2$

1 Joule of energy is dissipated by a 1 Amp current flowing through a 1 Ohm resistor for 1 second.

Also, 1 Joule of energy is 1 Watt (1 amp into 1 ohm) dissipating for 1 second.

Operating point Q

Operating point P



Operating point R

Each square shows chip temperature (C) and power (W)

Q4: Part A ...

Question 4a (4 points). Different systems may have different values of K. For example, the support chips for a laptop design may consume 2 Watts (K = 2), while the support chips for a desktop design may consume 7 Watts (K = 7).

A program runs twice as fast at Operating Point P than at Operating Point Q. The last instruction of the program turns off power to the processor and its support chips. For what range of value for K does (1) operating point P use the lowest amount of energy to run the program (2) operating point Q use the lowest amount of energy to run the program (3) the two operating points use the same amount of energy?

Operating point P: 1.3 V, 4.8 GHz, 10 W.

Operating point Q: 1.3 V, 2.4 GHz, 5 W.

Q4: Part A answer

Answer: We assume operating point Q takes t seconds to run, and operating point P takes $t/2$ seconds to run. Given this definition, and the numbers in the Schmoo chart, we deduce:

$$\text{Total P energy} = (t/2)(K + 10)$$

$$\text{Total Q energy} = (t)(K + 5)$$

For part (1) of this question, we solve the inequality:

$$\text{Total P energy} < \text{Total Q energy}$$

$$(t/2)(K + 10) < (t)(K + 5)$$

$$(K + 10) < (2K + 10)$$

$$K < 2K$$

Since for all positive K this inequality holds, the answer to (1) is "all K greater than 0". Using the same technique, we discover the answer to (2) is "never" (as K would need to be negative, which is impossible, unless we have an energy source), and the answer to (3) is "if K is equal to 0".

Q4: Part B ...

Question 4b (6 points). A program runs twice as fast at Operating Point P than at Operating Point R. The last instruction of the program turns off power to the processor and its support chips. For what values of K does (1) operating point P use the lowest amount of energy to run the program (2) operating point R use the lowest amount of energy to run the program (3) the two operating points use the same amount of energy? Draw a box around your final answer, which should be in three parts (an answer for (1), an answer to (2), an answer for (3)).

Operating point P: 1.3 V, 4.8 GHz, 10 W.

Operating point Q: 1.3 V, 2.4 GHz, 5 W.

Operating point R: 0.9 V, 2.4 GHz, 1 W.

Q4: Part B answer

Answer: We assume operating point R takes t seconds to run, and operating point P takes $t/2$ seconds to run. Given this definition, and the numbers in the Schmoo chart, we deduce:

$$\text{Total P energy} = (t/2)(K + 10)$$

$$\text{Total R energy} = (t)(K + 1)$$

For part (1) of this question, we solve the inequality:

$$\text{Total P energy} < \text{Total R energy}$$

$$(t/2)(K + 10) < (t)(K + 1)$$

$$(K + 10) < (2K + 2)$$

$$K > 8$$

Thus, the answer to (1) is "all K greater than 8". Using the same technique, we discover the answer to (2) is "all K less than 8", and the answer to (3) "if K is equal to 8".

Part III - Pipelining

Mid-term, Part III: Pipelining

Typical Topics

Visualize the necessary stalls in a pipelined CPU with no interlocks.

Analyze the impact of a structural hazard on a pipeline.

Visualize the behavior of a novel forwarding network.

Show control signals for a forwarding network.

T 9/19	Pipelining I
W 9/20	
Th 9/21	Pipelining II
F 9/22	
Sa 9/23	
Su 9/24	
M 9/25	
T 9/26	Pipelining III

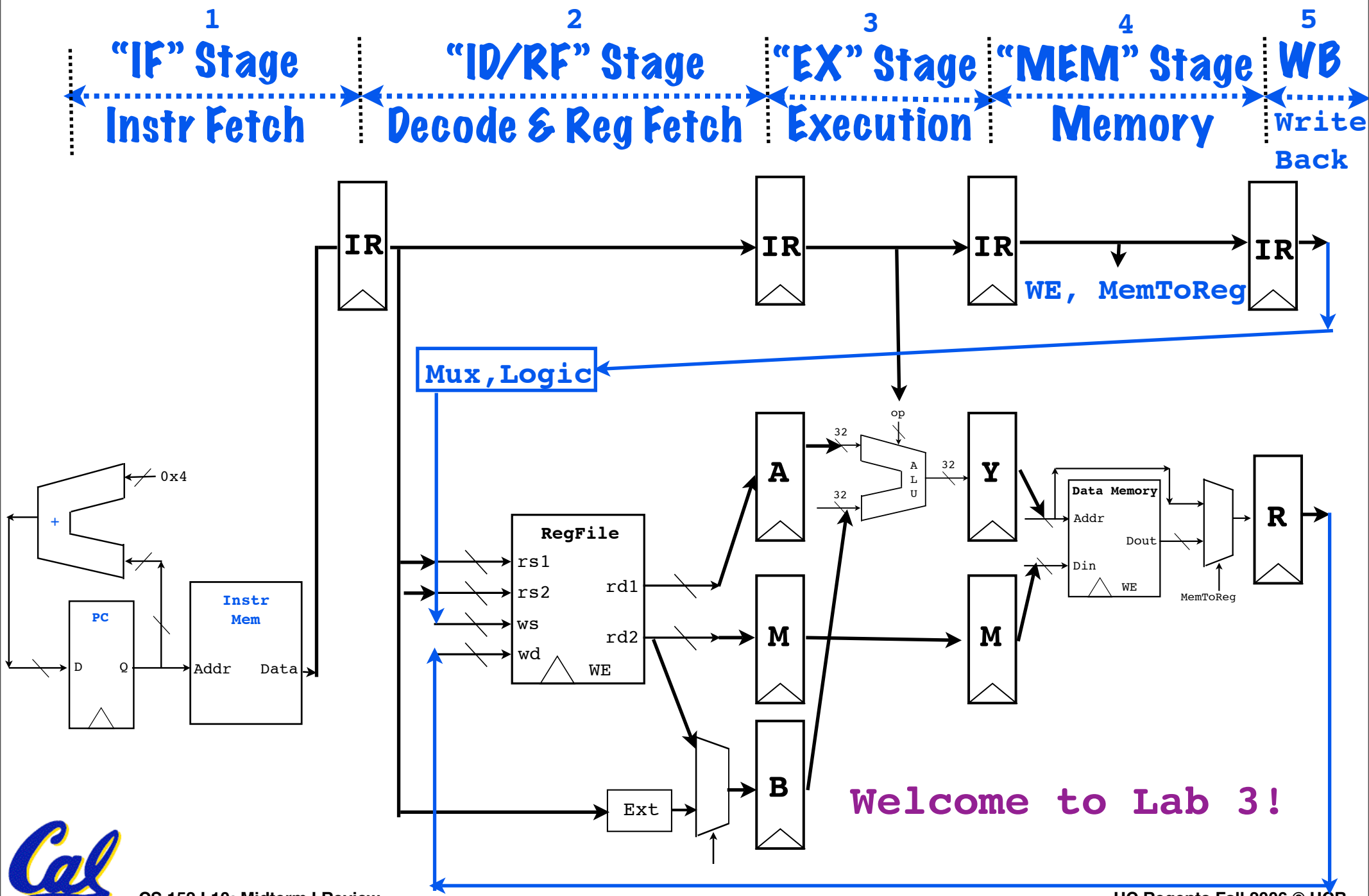
#	Points	
1	10	
2	15	
3	10	
4	10	
5	15	
6	15	
7	10	
8	15	
Tot	100	

Mid-term, Part III: How to do well …

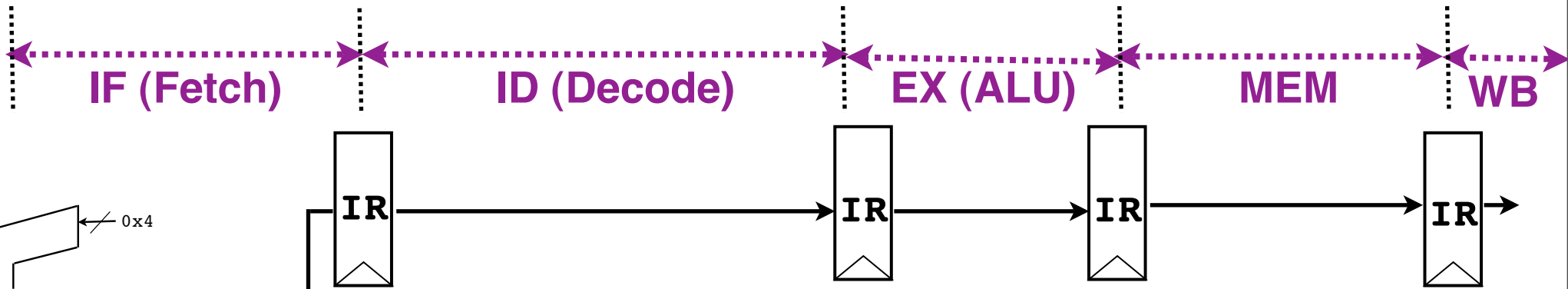
T 9/19	Pipelining I
W 9/20	
Th 9/21	Pipelining II
F 9/22	
Sa 9/23	
Su 9/24	
M 9/25	
T 9/26	Pipelining III

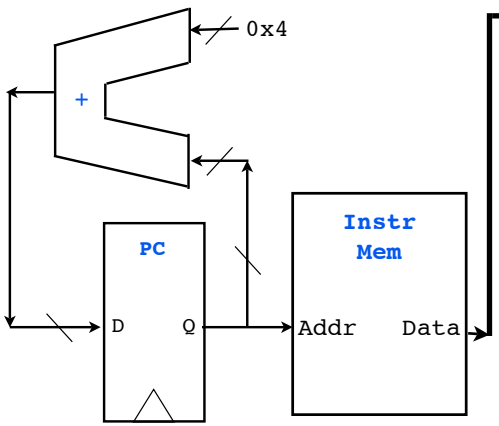
- Understand all parts of **all three lectures.**
- Understand your **Lab 3** preliminary **design doc.**
- Understand the **HW 1 pipeline** solutions.
- There will **not** be "you can only get it if you read the book" problems … but the reading helps you with the 3 "understands" above.

A (simplified) 5-stage pipelined CPU



Pipeline Representation #1: Timeline





Good for visualizing pipeline fills.

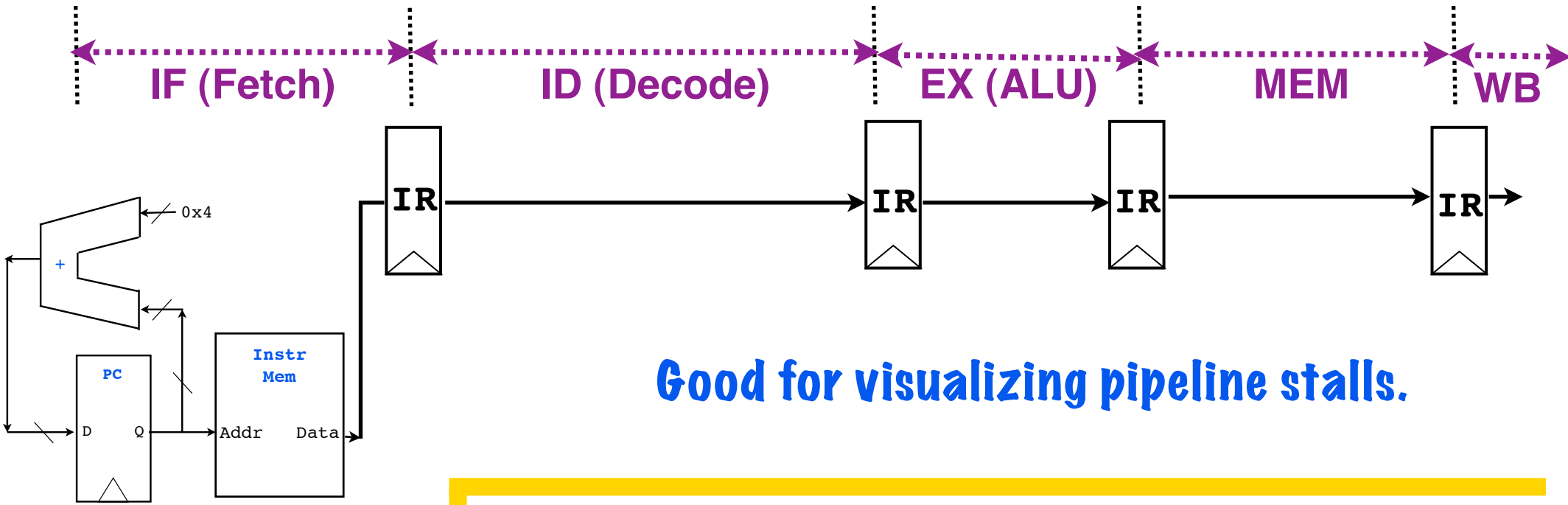
Sample Program

```
I1: ADD R4,R3,R2
I2: AND R6,R5,R4
I3: SUB R1,R9,R8
I4: XOR R3,R2,R1
I5: OR  R7,R6,R5
```

Time:	t1	t2	t3	t4	t5	t6	t7	t8
Inst								
I1:	IF	ID	EX	MEM	WB			
I2:		IF	ID	EX	MEM	WB		
I3:			IF	ID	EX	MEM	WB	
I4:				IF	ID	EX	MEM	WB
I5:					IF	ID	EX	MEM
I6:						IF	ID	EX

Pipeline is "full"

Representation #2: Resource Usage



Good for visualizing pipeline stalls.

Sample Program

```
I1: ADD R4,R3,R2
I2: AND R6,R5,R4
I3: SUB R1,R9,R8
I4: XOR R3,R2,R1
I5: OR  R7,R6,R5
```

Time:	t1	t2	t3	t4	t5	t6	t7	t8
Stage								
IF:	I1	I2	I3	I4	I5	I6	I7	I8
ID:		I1	I2	I3	I4	I5	I6	I7
EX:			I1	I2	I3	I4	I5	I6
MEM:				I1	I2	I3	I4	I5
WB:					I1	I2	I3	I4

Pipeline is "full"

Q5: Visualizing Stalls and Kills

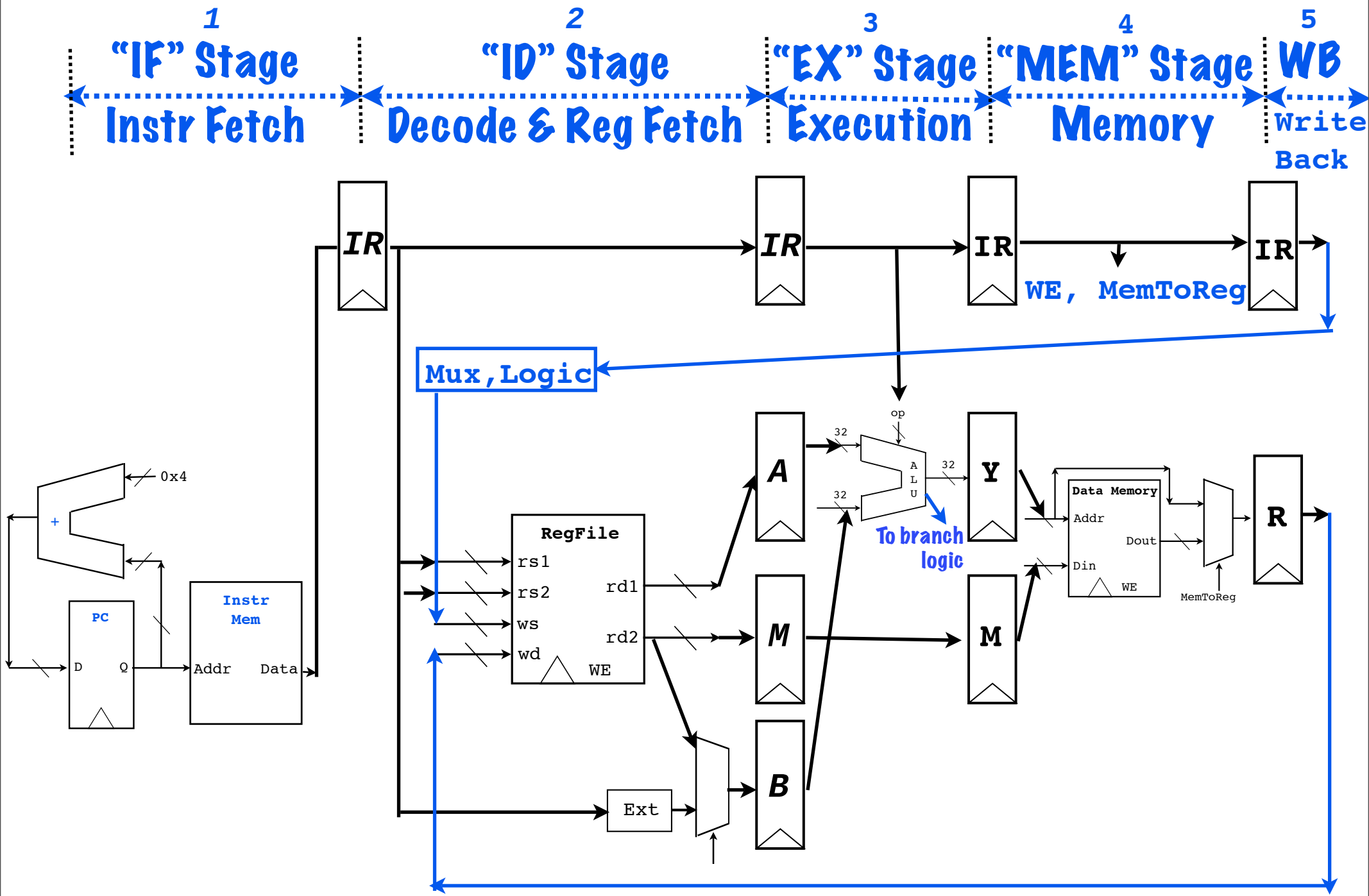
On the next page, we show the simple 5-stage MIPS pipelined processor from Lecture 4-1. This processor does not have forwarding paths and muxes, and does not have a comparator ALU for branches. The processor does have the ability to stall and to kill instructions at each stage of the pipeline. For clarity,

The programmers contract for this processor indicates that all branch and jump instructions have a single delay slot, but load instructions do NOT have load delay slot (thus, if a load instruction writes a register R6, the CPU must

On the next page, we also show a short machine language program that we wish to run on the processor. The controller for the CPU meets the programmers contract while minimally impacting performance (for example, stalls do not

In the visualization diagram below the program, draw in the instruction (I1, I2, etc) that occurs in each pipeline stage for each time tick, assuming that I1 appears in IF at clock tick t1. Use the symbol N for a stage that holds a NOP.

Note: no forwarding muxes, no "==" ID ALU



Notes:

In BEQ, the I7 denotes the branch target instruction (if the branch is taken). Look at the code to figure out if branch is taken or not.

Use N to denote a stage with a muxed-in NOP instruction.

Fill out the table until all slots of t1 3 are filled in. Do not add and fill in t1 4, t1 5, etc. We filled in I1 to get you started.

Program

```
I1:  OR R5,R1,R2
I2:  OR R6,R1,R2
I3:  BEQ R6,R5,I7
I4:  LW  R3 0(R5)
I5:  OR R7,R5,R6
I6:  OR R0,R3,R7
I7:  OR R6,R0,R3
I8:  OR R5,R0,R1
I9:  OR R11,R9,R9
I10: OR R12,R9,R9
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1												
ID:		I1											
EX:			I1										
MEM:				I1									
WB:					I1								

Notes:

In BEQ, the I7 denotes the branch target instruction (if the branch is taken). Look at the code to figure out if branch is taken or not.

Use N to denote a stage with a muxed-in NOP instruction.

Fill out the table until all slots of t13 are filled in. Do not add and fill in t14, t15, etc. We filled in I1 to get you started.

Program

```
I1:  OR R5,R1,R2
I2:  OR R6,R1,R2
I3:  BEQ R6,R5,I7
I4:  LW  R3 0(R5)
I5:  OR R7,R5,R6
I6:  OR R0,R3,R7
I7:  OR R6,R0,R3
I8:  OR R5,R0,R1
I9:  OR R11,R9,R9
I10: OR R12,R9,R9
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1	I2	I3	I4	I4	I4	I4	I5	I7	I8	I8	I8	I9
ID:		I1	I2	I3	I3	I3	I3	I4	N	I7	I7	I7	I8
EX:			I1	I2	N	N	N	I3	I4	N	N	N	I7
MEM:				I1	I2	N	N	N	I3	I4	N	N	N
WB:					I1	I2	N	N	N	I3	I4	N	N

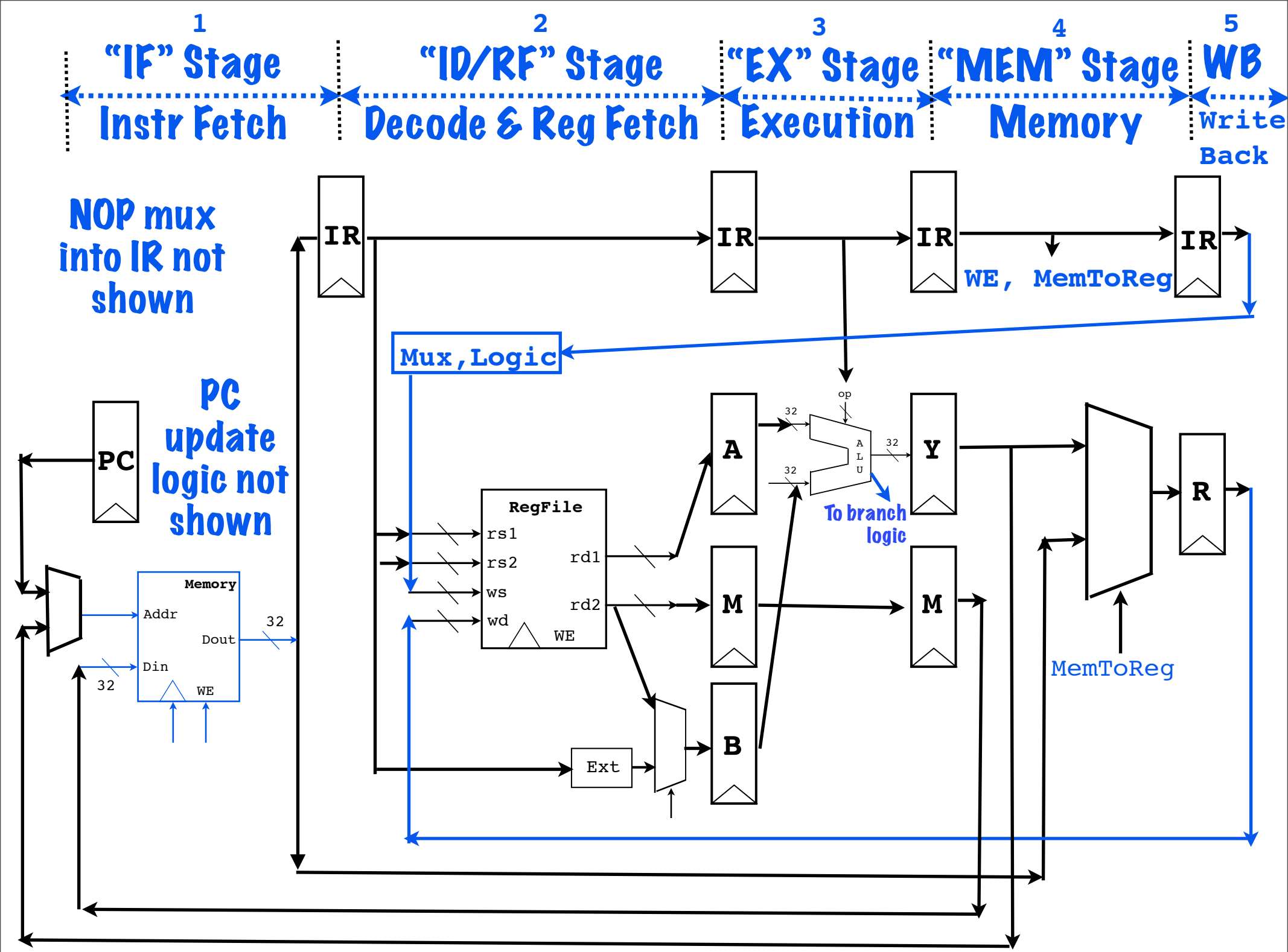
Q6: Unified Memory and Pipelines

On the next page, we show a simple 5-stage MIPS pipelined processor. However, a single memory (in stage 1) is used for both instruction and data memory. As

The programmers contract for this processor indicates that all branch and jump instructions have a single delay slot, and load instructions have a load delay slot (for this problem, defined as "the processor is under no obligation to

This design creates a structural hazard. The controller solves this hazard by always giving a load or store instruction in the MEM stage access to the unified memory, forcing instruction fetches to wait for the next free cycle. During this cycle, the IF stage itself holds a NOP.

Whenever permitted by the programmers contract, the controller lets instructions flow through the pipeline without stalls. Only when necessary, the controller stalls the pipeline, by muxing a NOP into the stage following the stall. The controller is also able to kill instructions, including the NOP instruction that appears in the IF stage during a memory hazard. The controller chooses to stall or kill in each stage in order to optimize the average number of cycles per instruction while fulfilling the programmer's contract.


1
"IF" Stage
Instr Fetch
2
"ID/RF" Stage
Decode & Reg Fetch
3
"EX" Stage
Execution
4
"MEM" Stage
Memory
5
WB
Write Back
NOP mux into IR not shown
PC update logic not shown
IR
IR
IR
IR
WE, MemToReg
Mux,Logic
PC
A
B
Y
M
M
R
op
32
32
32
ALU
To branch logic
RegFile
rs1
rs2
ws
wd
rd1
rd2
WE
Memory
Addr
Dout
Din
WE
32
32
Ext
MemToReg

Policy: Data reads and writes take precedence over instruction fetches.

Use N to denote a stage holding a NOP.

Fill out the table until all slots of t13 are filled in. Do not add and fill in t14, t15, etc. We filled in I1 to get you started.

Program

```
I1: LW R1, 0(R0)
I2: LW R2, 0(R1)
I3: LW R3, 0(R1)
I4: LW R4, 0(R3)
I5: LW R5, 0(R3)
I6: LW R6, 0(R4)
I7: OR R5,R6,R5
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1												
ID:		I1											
EX:			I1										
MEM:				I1									
WB:					I1								

Policy: Data reads and writes take precedence over instruction fetches.

Use N to denote a stage holding a NOP.

Fill out the table until all slots of t13 are filled in. Do not add and fill in t14, t15, etc. We filled in I1 to get you started.

Program

```
I1: LW R1, 0(R0)
I2: LW R2, 0(R1)
I3: LW R3, 0(R1)
I4: LW R4, 0(R3)
I5: LW R5, 0(R3)
I6: LW R6, 0(R4)
I7: OR R5,R6,R5
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1	I2	I3	N	N	I4	I5	N	N	I6	I7	N	N
ID:		I1	I2	I3	I3	I3	I4	I5	I5	I5	I6	I7	I7
EX:			I1	I2	N	N	I3	I4	N	N	I5	I6	N
MEM:				I1	I2	N	N	I3	I4	N	N	I5	I6
WB:					I1	I2	N	N	I3	I4	N	N	I5

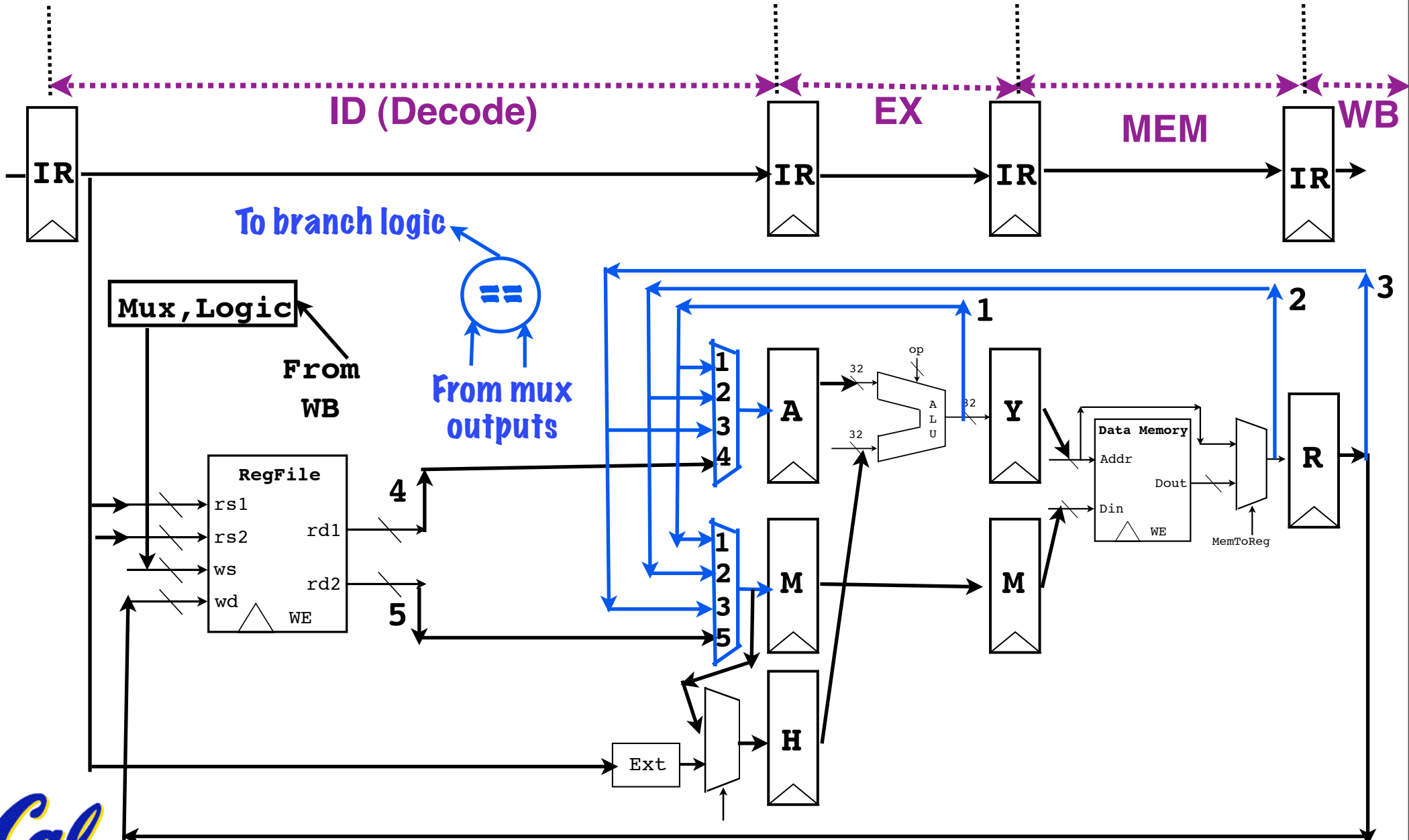
Q7: Forwarding Networks

On the next page, we show a 5-stage MIPS pipelined processor with a complete forwarding network, and an ID-stage comparator for branches. The programmer's contract for the machine specifies no load delay slot and one branch delay slot.

On the next page, we also show a machine language program, and a visualization diagram. Fill in the visualization diagram that correctly meets the programmer's contract, assuming that instruction I1 enters the IF stage at time t1. Add stalls ONLY if they are necessary, given the machine specification for

Note that each input to the two forwarding muxes is labelled with a number. The visualization diagram has two extra rows, A# and M#. For each time tick t_k, fill in these rows with the mux input that must be selected so that the clock edge at time t_{k+1} clocks in the right data value to meet the programmer's contract.

Forwarding muxes, with numbers inputs



Opcodes to datapath mapping:

```
OR ws,rs1,rs2          LW ws,imm(rs1)
BEQ rs1, rs2, branch target label
```

(1) Fill in IF/ID/EX/MEM/WB rows with instruction number (I1, I2, etc) or N for a stage that holds a NOP.

(2) Fill in A# with the selected input of the mux driving the A register needed to fulfill the programmers contract (1,2,3, 4, or X for don't care).

(3) Fill in M# with the selected input of the mux driving the M register needed to fulfill the programmers contract (1,2,3, 5, or X for don't care).

Program

```
I1:  OR R5,R1,R4
I2:  OR R4,R1,R2
I3:  OR R3,R5,R4
I4:  OR R3,R1,R2
I5:  BEQ R3,R4,I8
I6:  LW  R3 0(R3)
I7:  OR R6,R9,R3
I8:  OR R3,R3,R9
I9:  OR R9,R6,R3
I10: OR R3,R6,R6
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
IF:	I1									
ID:		I1								
EX:			I1							
MEM:				I1						
WB:					I1					
A#:	X									
M#:	X									

Opcodes to datapath mapping:

```
OR ws,rs1,rs2          LW ws,imm(rs1)
BEQ rs1, rs2, branch target label
```

(1) Fill in IF/ID/EX/MEM/WB rows with instruction number (I1, I2, etc) or N for a stage that holds a NOP.

(2) Fill in A# with the selected input of the mux driving the A register needed to fulfill the programmers contract (1,2,3, 4, or X for don't care).

(3) Fill in M# with the selected input of the mux driving the M register needed to fulfill the programmers contract (1,2,3, 5, or X for don't care).

Program

```
I1:  OR R5,R1,R4
I2:  OR R4,R1,R2
I3:  OR R3,R5,R4
I4:  OR R3,R1,R2
I5:  BEQ R3,R4,I8
I6:  LW  R3 0(R3)
I7:  OR R6,R9,R3
I8:  OR R3,R3,R9
I9:  OR R9,R6,R3
I10: OR R3,R6,R6
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
IF:	I1	I2	I3	I4	I5	I6	I8	I9	I9	I10
ID:		I1	I2	I3	I4	I5	I6	I8	I8	I9
EX:			I1	I2	I3	I4	I5	I6	N	I8
MEM:				I1	I2	I3	I4	I5	I6	N
WB:					I1	I2	I3	I4	I5	I6
A#:	X	4	4	2	4	1	2	X	2	4
M#:	X	5	5	1	5	3	X	X	5	1

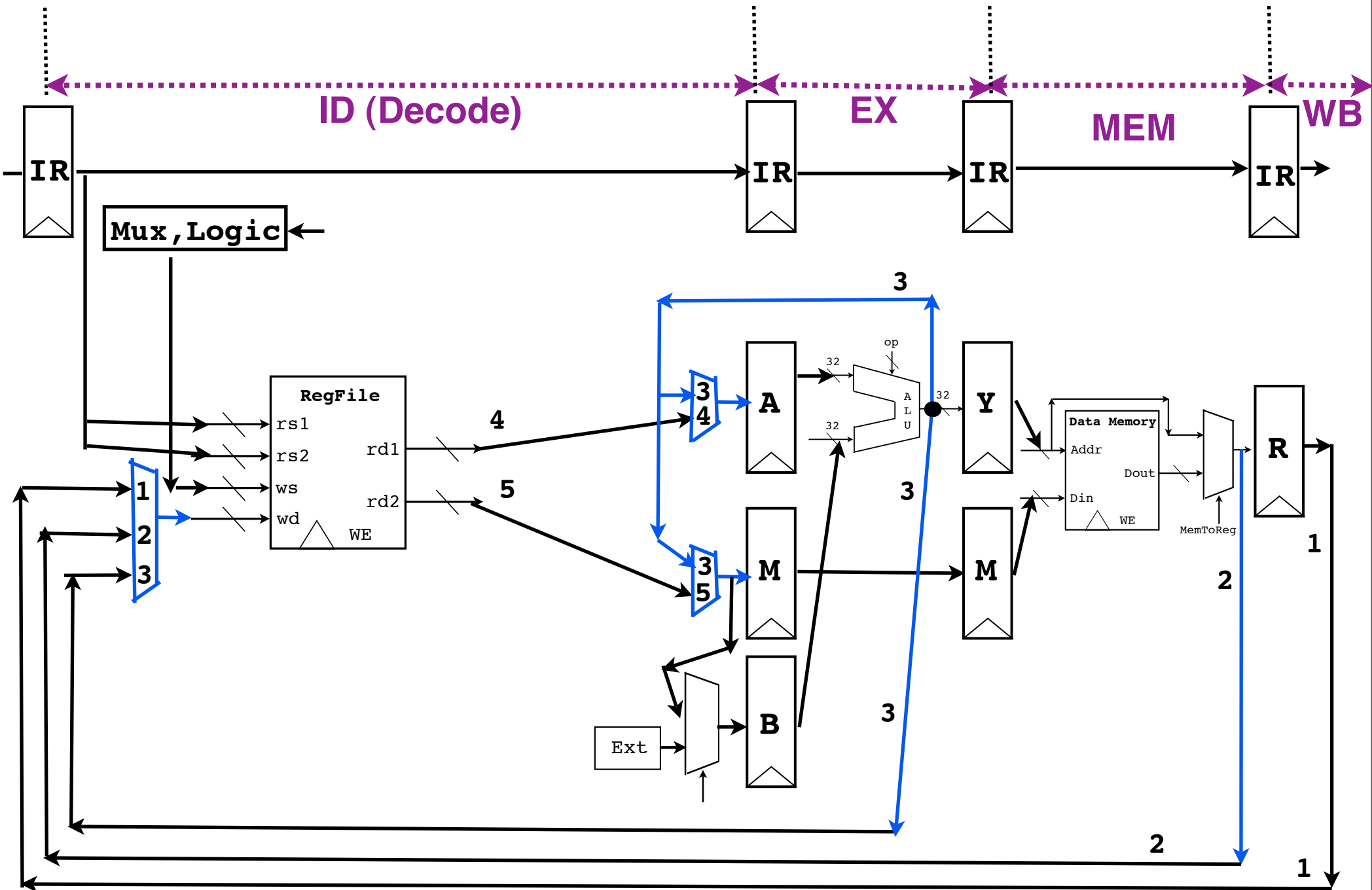
Q8: Forwarding Through Registers

On the next page, we show a 5-stage MIPS pipelined processor with a novel forwarding network. The processor has one branch delay slot and no load delay slot. As usual, the register file supports combinational reads and posedge clocked writes.

We also show a machine language program and a visualization diagram. If the forwarding network is used cleverly, it is possible for this CPU to execute this instruction stream stall-free while maintaining the programmers contract. Thus, we have filled out the top part of the visualization diagram with a stall-free pattern.

Note that each input to the three forwarding muxes is labelled with a number. The visualization diagram has three extra rows, one for each mux. For each time tick t_k, fill in these rows with the mux input that must be selected so that the clock edge at time t_{k+1} clocks in the right data value to meet the programmer's contract. See the comments on the visualization slide for details

A novel forward scheme (regfile mux)



Opcodes to datapath mapping: `OR ws,rs1,rs2`

Fill in A# with the selected input of the mux driving the A register needed to fufill the programmers contract (3, 4, or X for don't care).

Fill in M# with the selected input of the mux driving the M register needed to fufill the programmers contract (3, 5, or X for don't care).

Fill in wd with the selected input of the mux driving the wd register file input (1, 2, 3, or X for "don't care because there is no write this cycle")

Program

```
I1: OR R5,R1,R2
I2: OR R8,R3,R5
I3: OR R7,R8,R5
I4: LW R4 0(R8)
I5: OR R9,R8,R7
I6: OR R3,R9,R7
I7: OR R2,R4,R3
I8: OR R7,R3,R4
I9: OR R5,R7,R9
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1	I2	I3	I4	I5	I6	I7	I8	I9				
ID:		I1	I2	I3	I4	I5	I6	I7	I8	I9			
EX:			I1	I2	I3	I4	I5	I6	I7	I8	I9		
MEM:				I1	I2	I3	I4	I5	I6	I7	I8	I9	
WB:					I1	I2	I3	I4	I5	I6	I7	I8	I9
A#:	X												
M#:	X												
wd:	X												

Opcodes to datapath mapping:

`OR ws,rs1,rs2` `LW ws,imm(rs1)`

Fill in A# with the selected input of the mux driving the A register needed to fufill the programmers contract (3, 4, or X for don't care).

Fill in M# with the selected input of the mux driving the M register needed to fufill the programmers contract (3, 5, or X for don't care).

Fill in wd with the selected input of the mux driving the wd register file input (1, 2, 3, or X for "don't care because there is no write this cycle")

Program

```
I1: OR R5,R1,R2
I2: OR R8,R3,R5
I3: OR R7,R8,R5
I4: LW R4 0(R8)
I5: OR R9,R8,R7
I6: OR R3,R9,R7
I7: OR R2,R4,R3
I8: OR R7,R3,R4
I9: OR R5,R7,R9
```

	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13
IF:	I1	I2	I3	I4	I5	I6	I7	I8	I9				
ID:		I1	I2	I3	I4	I5	I6	I7	I8	I9			
EX:			I1	I2	I3	I4	I5	I6	I7	I8	I9		
MEM:				I1	I2	I3	I4	I5	I6	I7	I8	I9	
WB:					I1	I2	I3	I4	I5	I6	I7	I8	I9
A#:	X	4	4	3	4	4	3	4	4	3			
M#:	X	5	3	5	X	5	5	3	5	5			
wd:	X	X	3	3	3	X	2	3	1				

Q8: Forwarding Through Registers

For (at least) one of the "OR RX, RY, RZ" commands in the program, it is possible to change RY or RZ to a different register number, in such a way that it becomes impossible to use the forwarding network to execute the instruction stream in a stall-free way. What is the label of this instruction ($I1 \ldots I9$), and how should the instruction be changed to make stall-free execution impossible? Write your answer below.

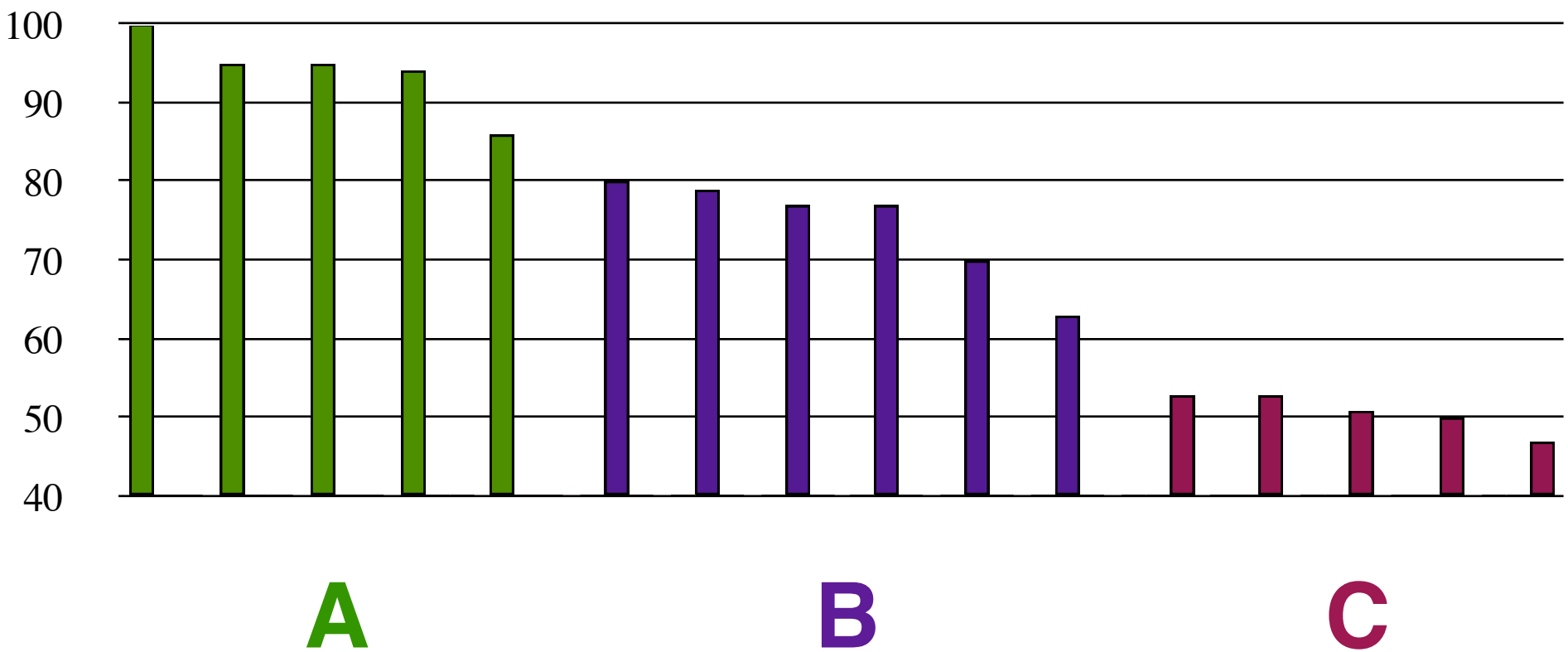
Answer: The label is I8. If this instruction is changed to be:

I8 : OR R7 R3 R9

a stall is impossible to avoid.

Fall 05 Mid-term I Grade Distribution ...

It's not about "absolute numbers" - the numerical cutoffs for Fall 06 will almost certainly be different. Example: "A" cutoff might be 90 instead of the 85 below.



If the "C" students didn't exist, most (but not all) "B" students would still have "B"s on this exam, and most (but not all) "A" students would still have "A"s.

Administrivia: Mid-term I ...

Starts at 6PM in 306 Soda. No class on Tuesday

T 10/3	Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)

Assigned seating: Look for your namecard on the desk.

Put all electronic devices (cell phones, caclulators, PDAs, computers) at the front of the room.

Put bookbags and other personal belongings at the front of the room.

Administrivia: Mid-term I ...

Starts at 6PM in 306 Soda. No class on Tuesday

T 10/3	Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)

Just writing implements at your desk (pencils, pens, erasers, etc).

Test problems include useful information for the problems ...

... so, no "one sheet of paper" allowed for this test. You won't be able to bring one to job interviews either ...

Administrivia: Mid-term I ...

Word answers may have "in 10 words or less", "in 20 words or less", etc -- these will be enforced. This prevents "write everything I can think of, and hope something matches" strategy.

I will be grading the test personally.

Admin: Team Evaluations due Thursday

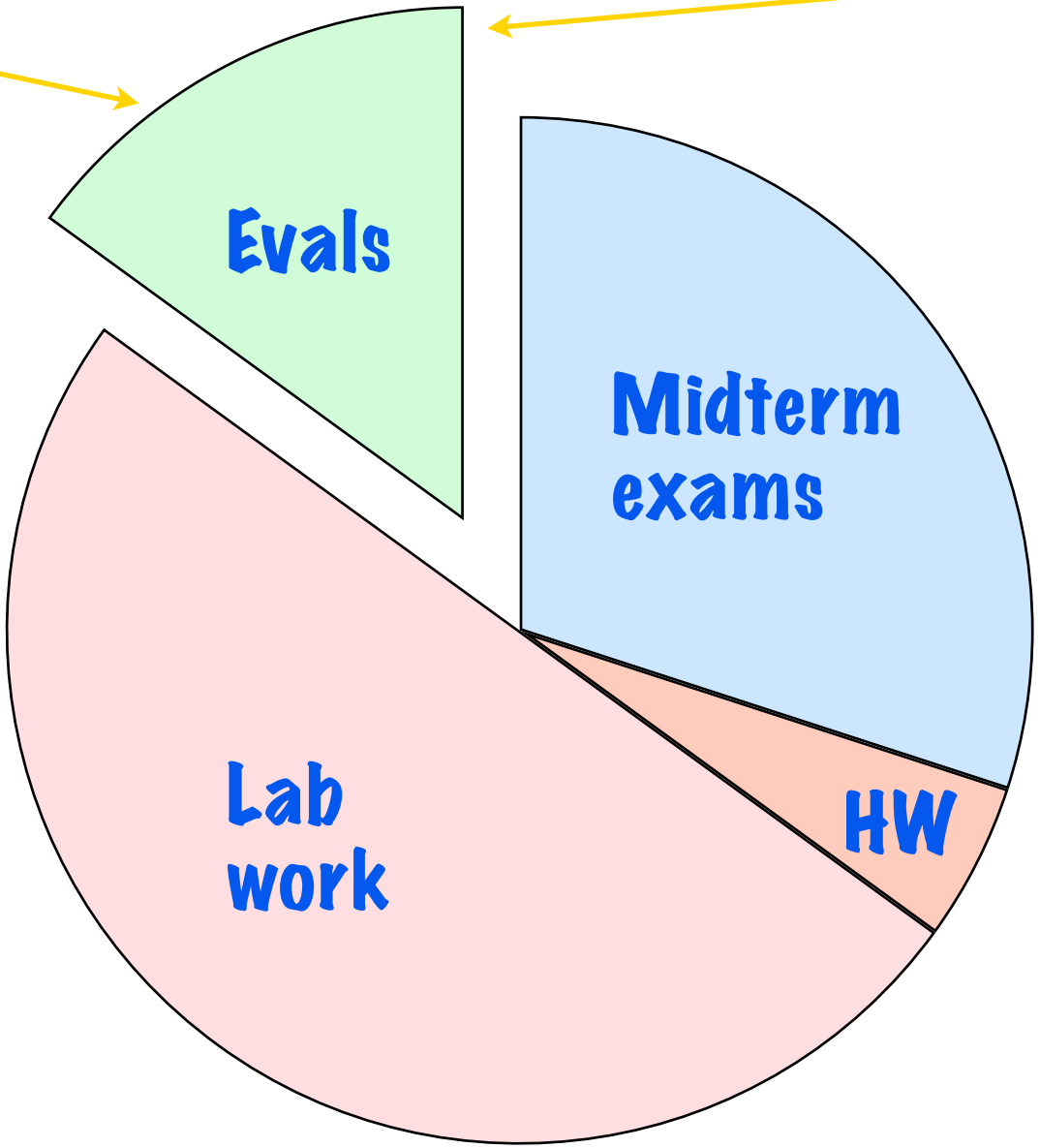
Th 9/28	Midterm Review Session in Class			HW 1 due in class Lab 3: ~~Preliminary Design Documer~~ and Team Evaluations due to TAs, 11:59PM

Name	Evaluation	Justification
Bill	30%	Bill **did** design the modules he was assigned to do, I'll give him that much. But he was missing in action during the 20 hours we spent in the lab doing integration and bug-fixes. This may be related to his behavior during our first meeting. Bill started yelling and screaming after we decided not to use the gated-clock reset scheme he proposed for the processor -- and then he stomped out of the room and slammed the door.
Sue	100%	5 hours into our final debugging section, Sue spotted the bug that kept our processor from working -- a bug that was my mistake. She saved our group. She also designed her own bug-free modules on-time, and set up our CVS system.
Joe	100%	A solid contribution from Joe -- he did everything I would expect from a good team member. He did many of the testing and integration tasks that Bill was assigned to do but never did.

Grading: Peer and staff evaluations

Peer evals:

Teammates grade each other after Labs 2, 3, 4. Rewards good "team players"



Staff evals:

TAs are your "managers"

They observe how well you work on the team.

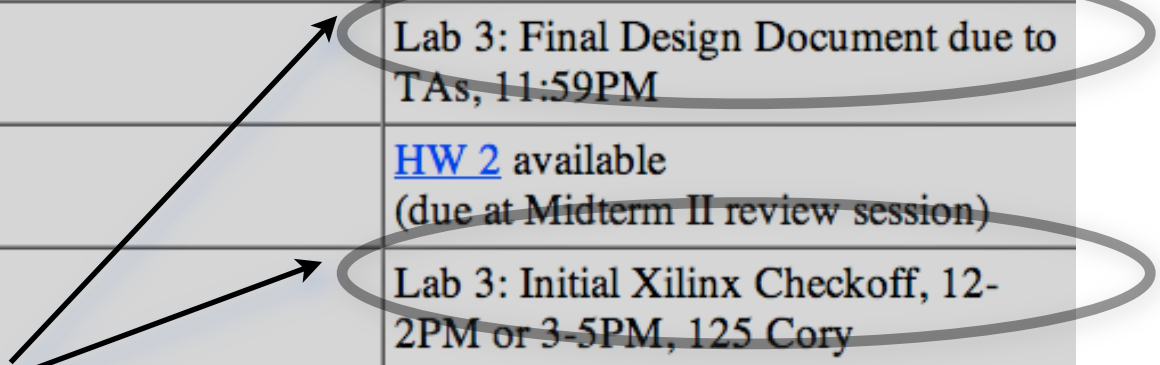
Admin: Design Document Deadlines

Th 9/28	Midterm Review Session in Class			HW 1 due in class Lab 3: Preliminary Design Documen and Team Evaluations due to TAs, 11:59PM
F 9/29				Lab 3: Preliminary Design Documen Review, 12-2PM or 3-5PM, 125 Co

1. The identity of the **spokesperson** for the lab, and a roster of group members. The responsibility of the spokesperson is to communicate questions to the TA and reply to questions from the TA. Choose a different spokesperson than the one you had for Lab 2.
2. A short description of the structure of the design. The description will be accompanied by preliminary high-level schematics of your datapath, and a preliminary discussion of the controller.
3. A description of the unit test benches and multi-unit test benches you intent to create for your processor, and a description of machine language programs you intent to write to use in complete processor testing. Also, a **test plan**, using the epoch charting method shown in the 9/7 lecture, that shows when you plan to run each type of test.
4. A tentative **division of labor**, showing the tasks each group member intends to do.
5. The "**paranoia**" section: discuss potential areas of difficulty in the lab. An early guess of critical timing paths for the design should be a part of this section.

After the mid-term: Labs go on ...

Th 9/28	Midterm Review Session in Class			HW 1 due in class Lab 3: Preliminary Design Document and Team Evaluations due to TAs, 11:59PM
F 9/29				Lab 3: Preliminary Design Document Review, 12-2PM or 3-5PM, 125 Cory
Sa 9/30				
Su 10/1				
M 10/2				
T 10/3	Midterm I: 6:00PM to 9:00PM, 306 Soda (HP Auditorium) (note: no class 11-12:30)			
W 10/4				Lab 3: Final Design Document due to TAs, 11:59PM
Th 10/5	VLSI			HW 2 available (due at Midterm II review session)
F 10/6				Lab 3: Initial Xilinx Checkoff, 12-2PM or 3-5PM, 125 Cory



Lab 3 design doc, checkoffs, later in week ...

Philosophy ...

- Be smart about how you study. This lecture helps you do that.
- Budget your time effectively between mid-term preparation and Lab 3. Don't stop work on Lab 3 until Weds - your group will never make the Friday checkoff!
- Good luck!