CS 152
Computer Architecture and Engineering
Lecture 16 -- Midterm I Review Session

2014-3-13

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

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

Play:

Today - Midterm I Review Session

- Study tips, test ground rules
- All questions answered (almost ...)
- Short break
- HW 1, problem by problem ...

Recall: HW 1 was Fall 05 Mid-term I.

On Tuesday

Mid-term I ...

T 3/18	Midterm I

Ground rules ...

When is it? Where is it? Ground rules.

- 9:30 AM sharp, Tuesday March 18th, 306 Soda.
- Every-other-seat seating, except for the front row, where every-seat is permitted.
- No blue-books needed. We will be handing out a paper test. Pencil is preferred.
- Pencils down @ 10:55 AM, so we can collect papers before next class comes in.

When is it? Where is it? Ground rules.

- No use of calculators, smartphones, laptops, etc ... during the exam.
- Closed-book, closed-notes. Just pencils, erasers. No consulting with students.
- Restroom breaks are OK, but you'll still need to hand in your exam @ 10:55.
- Questions are reserved for serious concerns about a bug in the question.

What does it cover? First 8 lectures.

Not to be taken legalistically. For example, WAW hazards were covered in the pipelining lecture, and so it's fair for me to show a pipeline and say "does this have a WAW hazard", even if that example was also seen in a later lecture.

T 1/21	Single-Cycle Design
Th 1/23	Single-Cycle Wrap-up + VLIW
T 1/28	Metrics
Th 1/30	Pipelining
T 2/4	Instruction Set Design + Microcode + Cost
Th 2/6	Super-Pipelining + Branch Prediction
T 2/11	Power and Energy
Th 2/13	CPU Verification

Mid-term: How to do well ...

- 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.
- There will not be "you can only get it if do the reading" problems ... but the reading helps you understand how to think through the problem.

Mid-term: There may be math ...

- **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: simple algebra and calculus, add a few numbers by hand, etc.

Cannot use electronic devices ... more administrative info after we do some content.

Example starting slides ...

These are meant to be examples, not a complete list!

A problem may start with a slide that is not in this part of the presentation!

T 1/21	Single-Cycle Design
Th 1/23	Single-Cycle Wrap-up + VLIW
T 1/28	Metrics
Th 1/30	Pipelining
T 2/4	Instruction Set Design + Microcode + Cost
Th 2/6	Super-Pipelining + Branch Prediction
T 2/11	Power and Energy
Th 2/13	CPU Verification

CS 152
Computer Architecture and Engineering
Lecture 2 – Single Cycle Wrap-up

2014-1-23

John Lazzaro

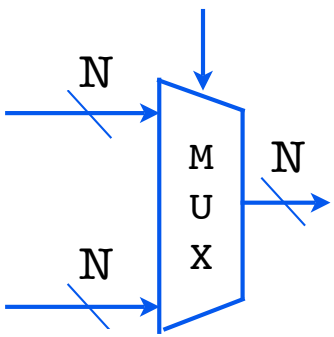
(not a prof - "John" is always OK)

TA: Eric Love

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

Merging data paths ...

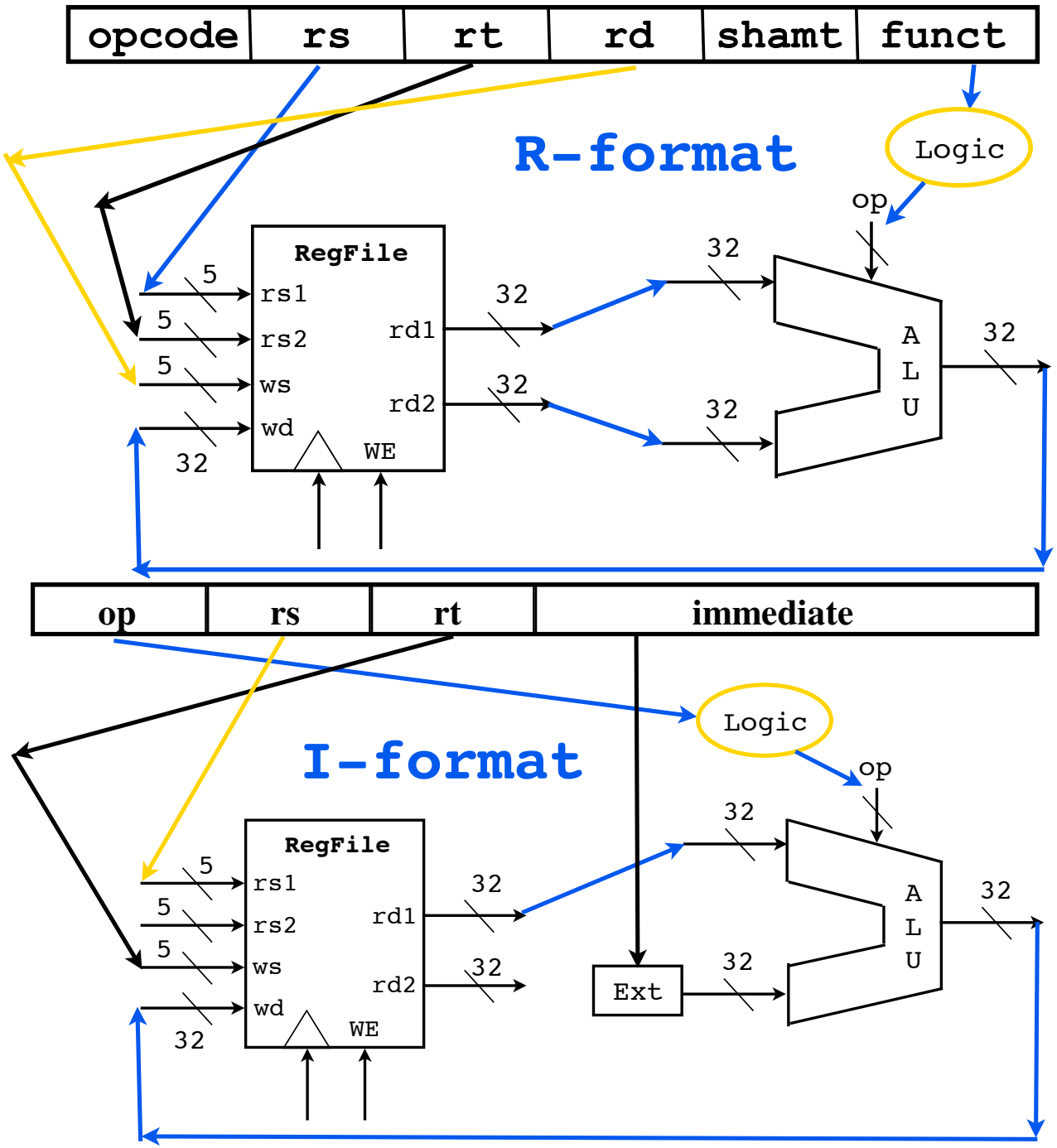
Add muxes



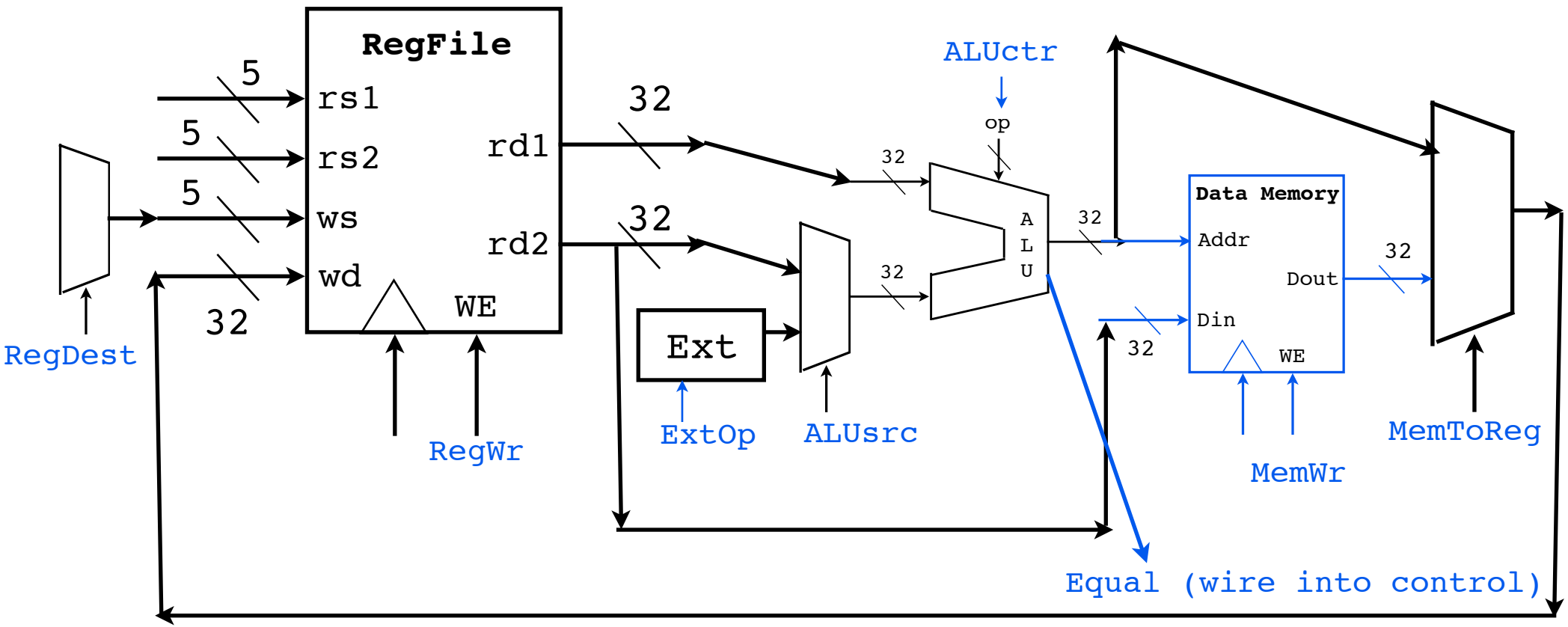
How many ? 2

(ignore ALU control)

Where ?



Adding branch testing to the data path



op	rs	rt	immediate

Syntax: BEQ $1, $2, 12

Action: If ($1 != $2), PC = PC + 4

Action: If ($1 == $2), PC = PC + 4 + 48

Josh Fisher: idea grew out of his Ph.D (1979) in compilers

[4] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. 10th Symp. Comput. Architecture*, IEEE, June 1983, pp. 140–150.

VLIW

Very
Long
Instruction
Words

Led to a startup (MultiFlow) whose computers worked, but which went out of business ... the ideas remain influential.

32-bit & 64-bit semantics different? Yes!

```
Assume: $7 = 7, $8 = 8, $9 = 9, $10 = 10 (decimal)
```

32-bit MIPS:

```
ADD $8 $9 $10;     Result: $8 = 19
ADD $7 $8 $9;      Result: $7 = 28
```

VLIW:

```
Instr:    ADD $8 $9 $10   ; result $8 = 19
          ADD $7 $8 $9    ; result $7 = 17 (not 28)
```

Branch policy: All instr operators execute

BNE $8 $9 Label ADD $7 $8 $9

opcode	rs	rt	rd	shamt	funct
opcode	rs	rt	rd	shamt	funct

ADD executes if branch is taken or not taken.

Problem: Large N machines find it hard to fill all operators with useful work.

Solution: New "predication" operator.

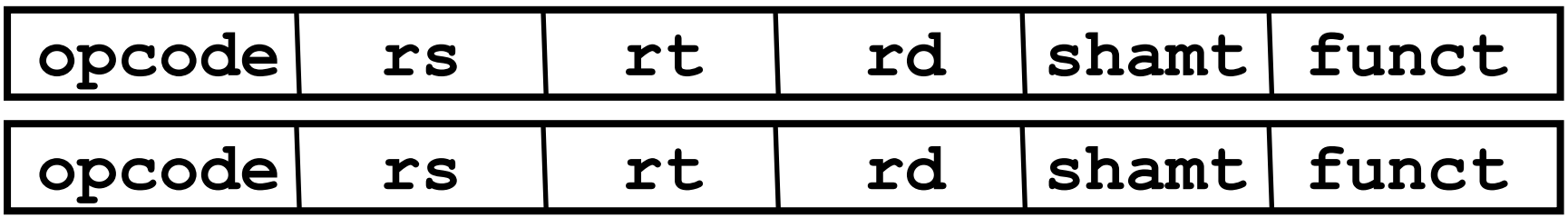
Syntax: SELECT $7 $8 $9 $10

Semantics: If $8 == 0, $7 = $10, else $7 = $9

Permits simple branches to be converted to inline code.

Branch nesting in a single instruction ...

BEQ $8 $9 LabelOne



BEQ $11 $12 LabelTwo

Conundrum: **How to define the semantics of multiple branches in one instruction?**

Solution: **Nested branch semantics**

```
If $8 == $9, branch to LabelOne
  Else $11 == $12, branch to LabelTwo
```

CS 152
Computer Architecture and Engineering
Lecture 3 – Metrics

2014-1-28

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

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

CPU time: Proportional to Instruction Count

Q. Once ISA is set, who can influence instruction count?

A. Compiler writer, application developer.

Q. Static count? (lines of program printout) Or dynamic count? (trace of execution)

A. Dynamic.

$$\frac{\text{CPU time}}{\text{Program}} \propto \frac{\text{Machine Instructions}}{\text{Program}}$$

Rationale: Every additional instruction you execute takes time.

Q. How does a architect influence the number of machine instructions needed to run an algorithm?

A. Create new instructions: instruction set architect.

Recall Lecture 2: Multi-flow VLIW CPU

Q. Which right-hand-side term decreases with "N" ?

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \quad \frac{\text{Cycles}}{\text{Instruction}} \quad \frac{\text{Seconds}}{\text{Cycle}}$$

A. This one gets smaller.

A. We hope this one doesn't grow.

```
Syntax:  ADD $8 $9 $10      Semantics: $8 = $9 + $10
```

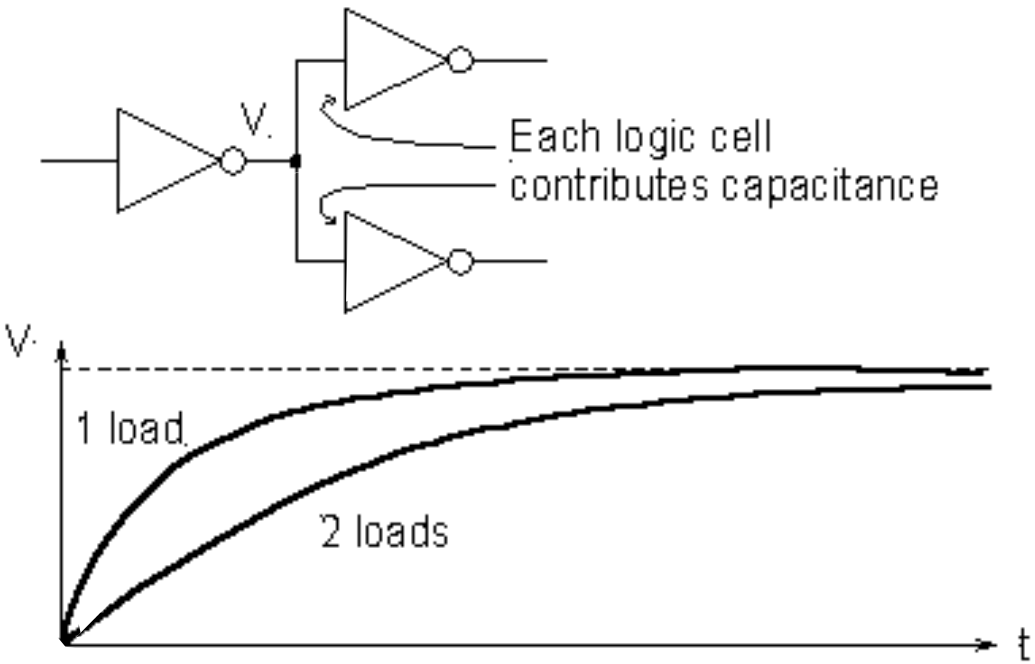
opcode	rs	rt	rd	shamt	funct
opcode	rs	rt	rd	shamt	funct

```
Syntax:  ADD $7 $8 $9       Semantics: $7 = $8 + $9
```

N x 32-bit VLIW yields factor of N speedup!

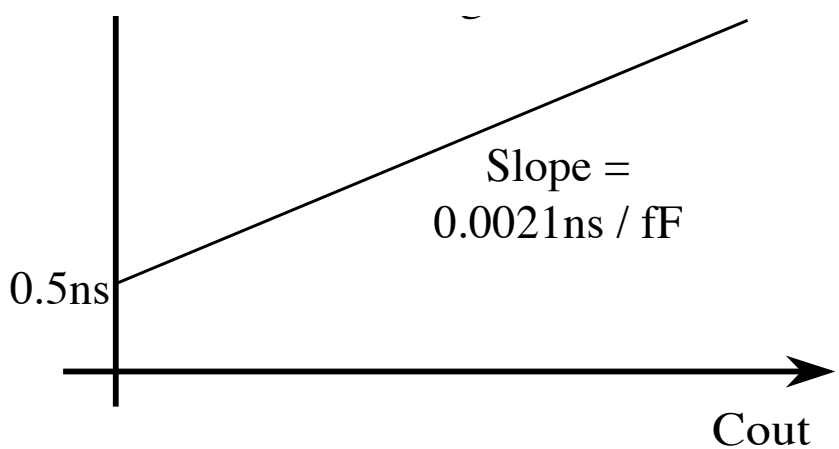
Multiflow: N = 7, 14, or 28 (3 CPUs in product family)

A closer look at fan-out ...

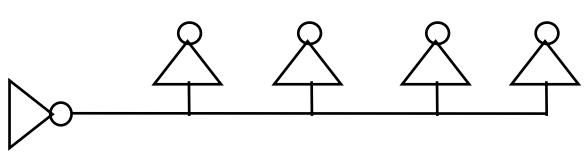


Driving more gates adds delay.

Linear model works for reasonable fan-out

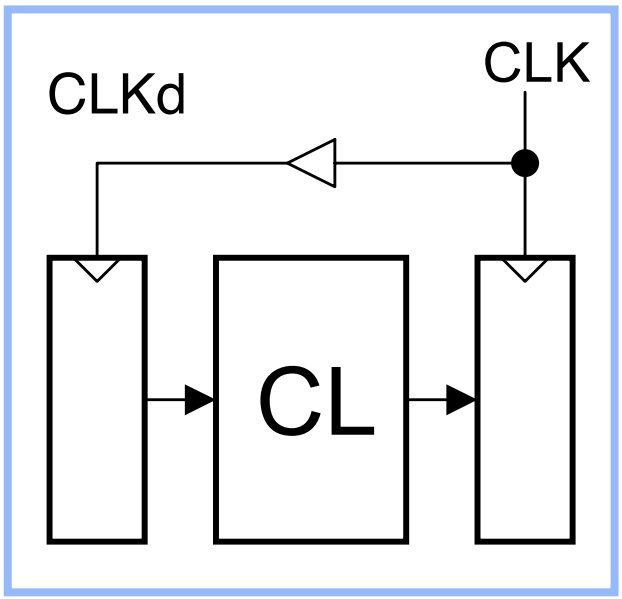


FO4: Fanout of four delay.

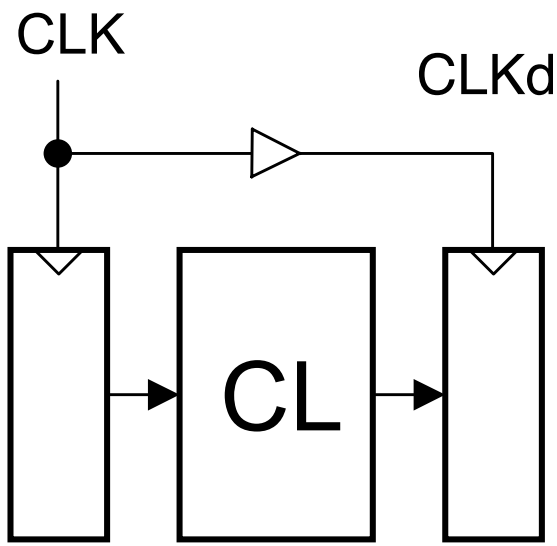


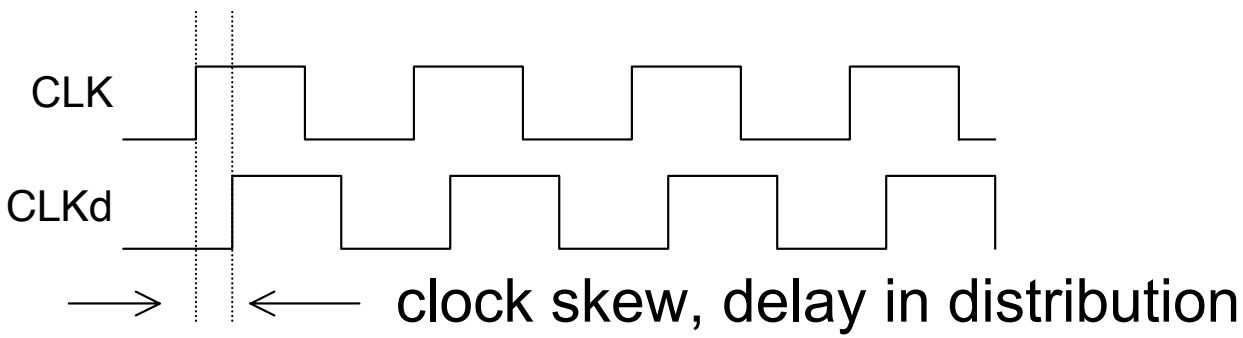
Delay time of an inverter driving 4 inverters.

Clock skew also eats into "time budget"



As T →0, which circuit fails first?





$$T \geq T_{CL} + T_{setup} + T_{clk \rightarrow Q} + \text{worst case skew.}$$

CS 152
Computer Architecture and Engineering
Lecture 4 – Pipelining

2014-1-30

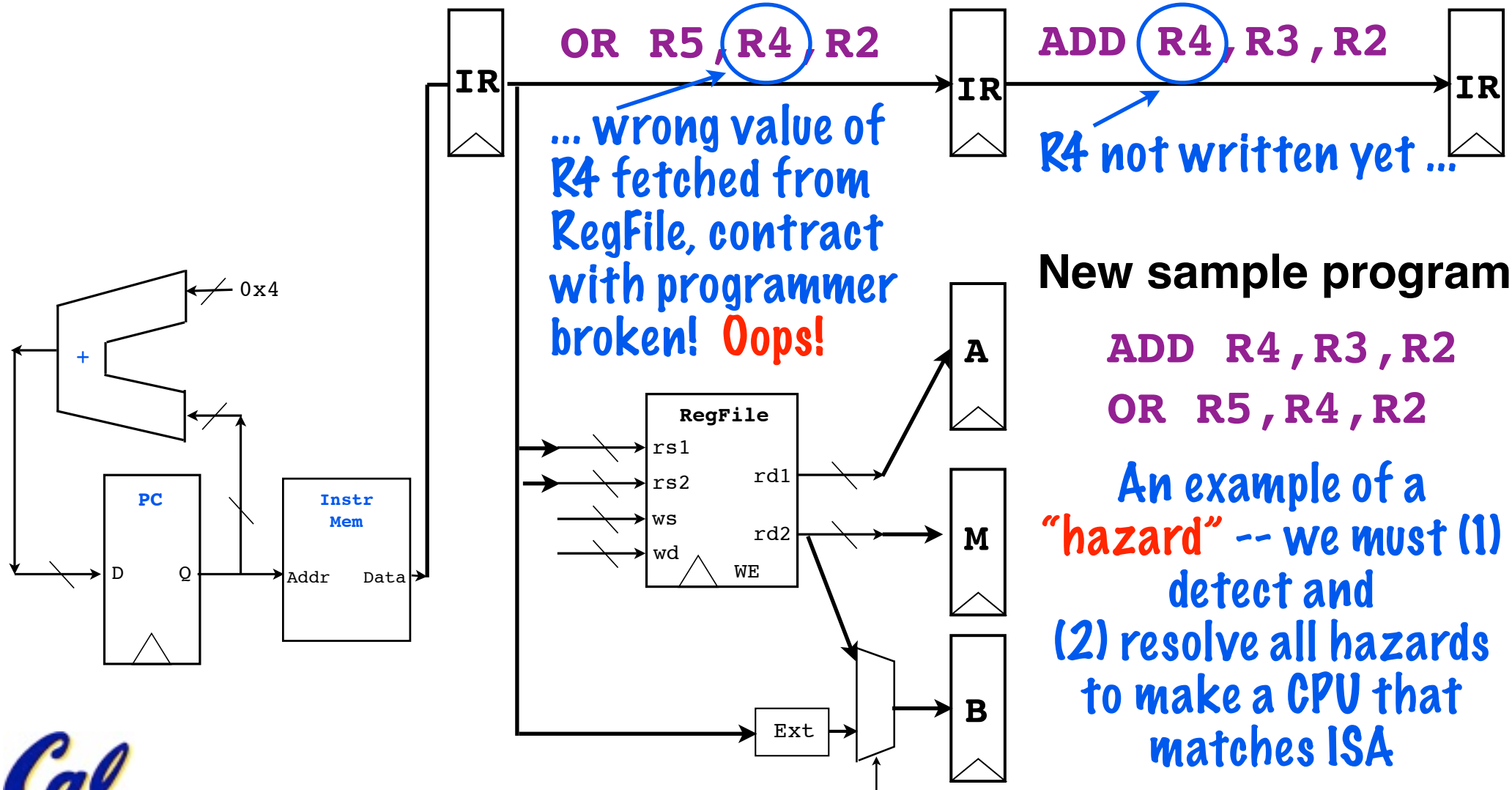
John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

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

Hazards: An instruction is not a car ...



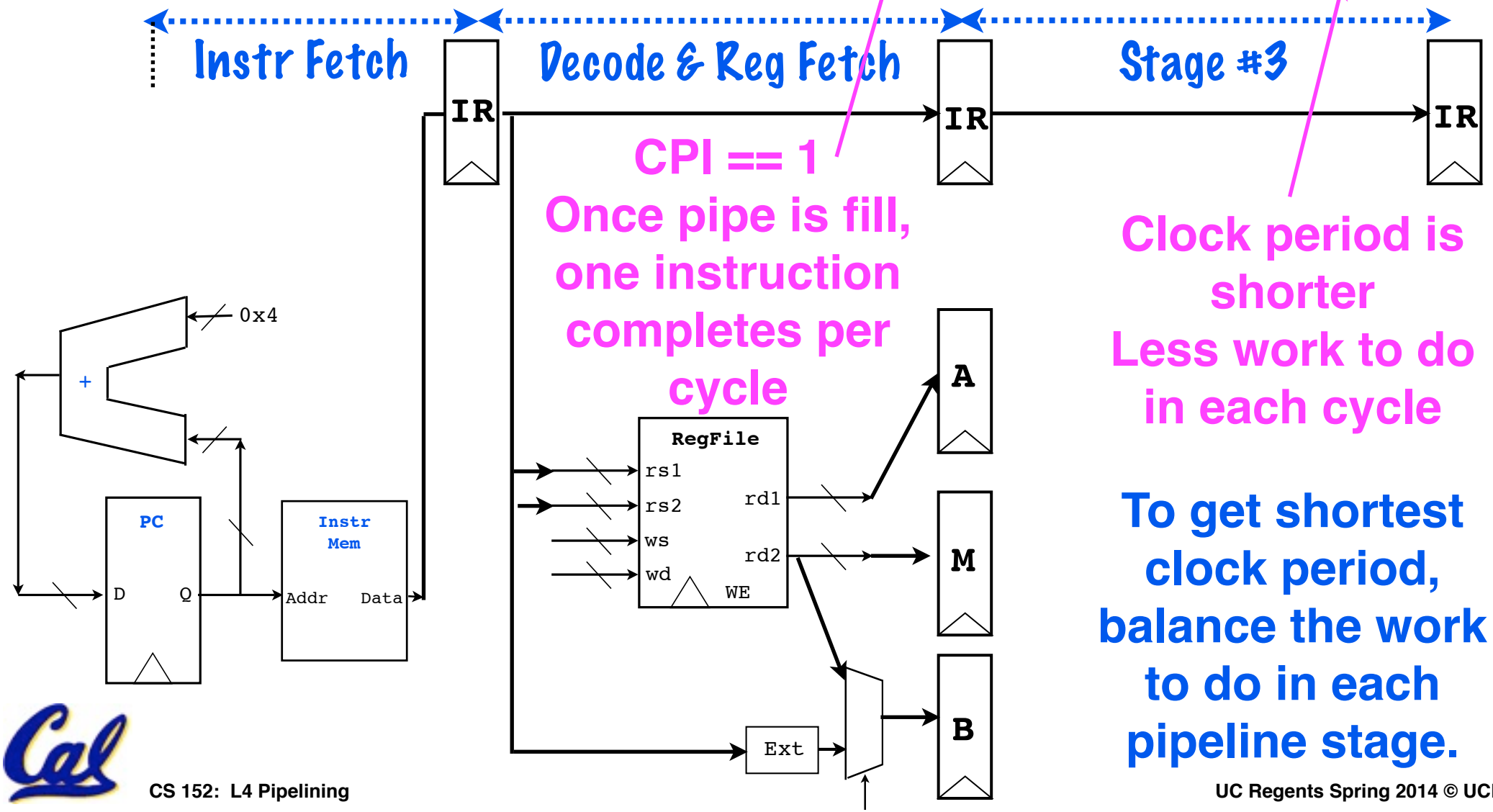
New sample program

```
ADD R4,R3,R2
OR R5,R4,R2
```

An example of a "hazard" -- we must (1) detect and (2) resolve all hazards to make a CPU that matches ISA

Performance Equation and Pipelining

$$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$



To get shortest clock period, balance the work to do in each pipeline stage.

Data Hazards: 3 Types (RAW, WAR, WAW)

Write After Read (WAR) hazards. Instruction I2 expects to write over a data value after an earlier instruction I1 reads it. But instead, I2 writes too early, and I1 sees the new value.

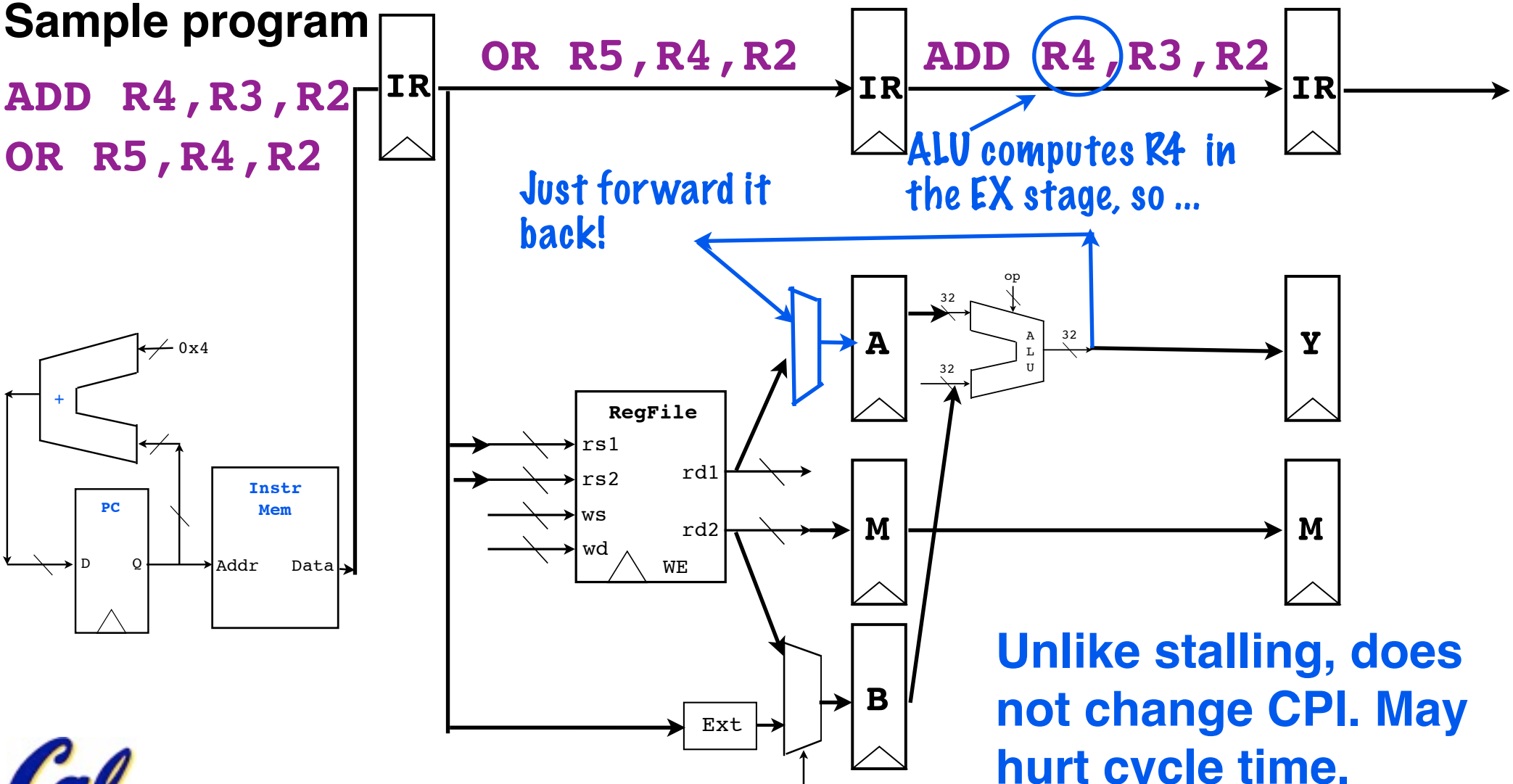
Write After Write (WAW) hazards. Instruction I2 writes over data an earlier instruction I1 also writes. But instead, I1 writes after I2, and the final data value is incorrect.

WAR and WAW not possible in our 5-stage pipeline.
But are possible in other pipeline designs.

Resolving a RAW hazard by forwarding

Sample program

```
ADD R4,R3,R2
OR R5,R4,R2
```



Unlike stalling, does not change CPI. May hurt cycle time.

CS 152
Computer Architecture and Engineering
Lecture 5 – ISA Design + Microcode + Cost

BORN ON THIS DAY IN 2004.

FACEBOOK

2014-2-4

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

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

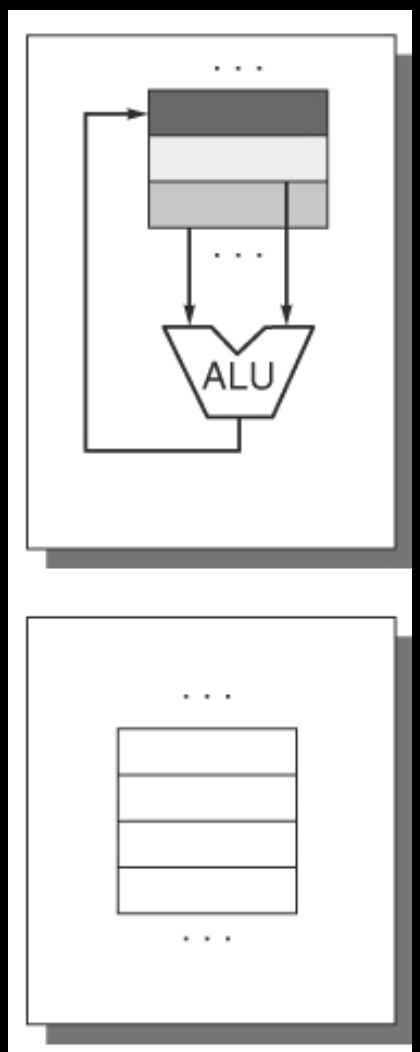
Register-register 1990s technology was ready for RISC

Type	Advantages	Disadvantages
Register-register (0, 3)	Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute (see Appendix C).	Higher instruction count than architectures with memory references in instructions. More instructions and lower instruction density lead to larger programs.

Machine code for
c = a + b;

Register (load-store)
Load R1,A
Load R2,B
Add R3,R1,R2
Store R3,C

Not really quantitative ...



Transistors were available for on-chip instruction cache.

So, larger code size would not monopolize bandwidth to off-chip DRAM memory.

Fixed-length instructions made fast pipelining practical.

For the right target ISA, compiled code quality could match hand-coded assembly.

Instruction modes: "C is a high-level assembler" origin

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer p, then mode yields $*p$.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + d	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, d.
Autodecrement	Add R1, -(R2)	Regs[R2] ← Regs[R2] − d Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * d]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

w += i

w += 3

w += a[100 + i]

w += a[i]

w += a[i + j]

w += a[1001]

w += a[*p]

a[i++]

a[i--]

w += a[100 + i + d*j]

How compiler technology can inform an ISA decision

Example:

f = b + c + d - 1

becomes:

```
a := c + d
e := a + b
f := e - 1
```

Temporaries: a, e

During code generation, a compiler allocates registers to temps when available, because registers are faster than memory.

```
a := c + d
e := a + b
f := e - 1
```

```
r1 := r2 + r3
r1 := r1 + r4
r1 := r1 - 1
```

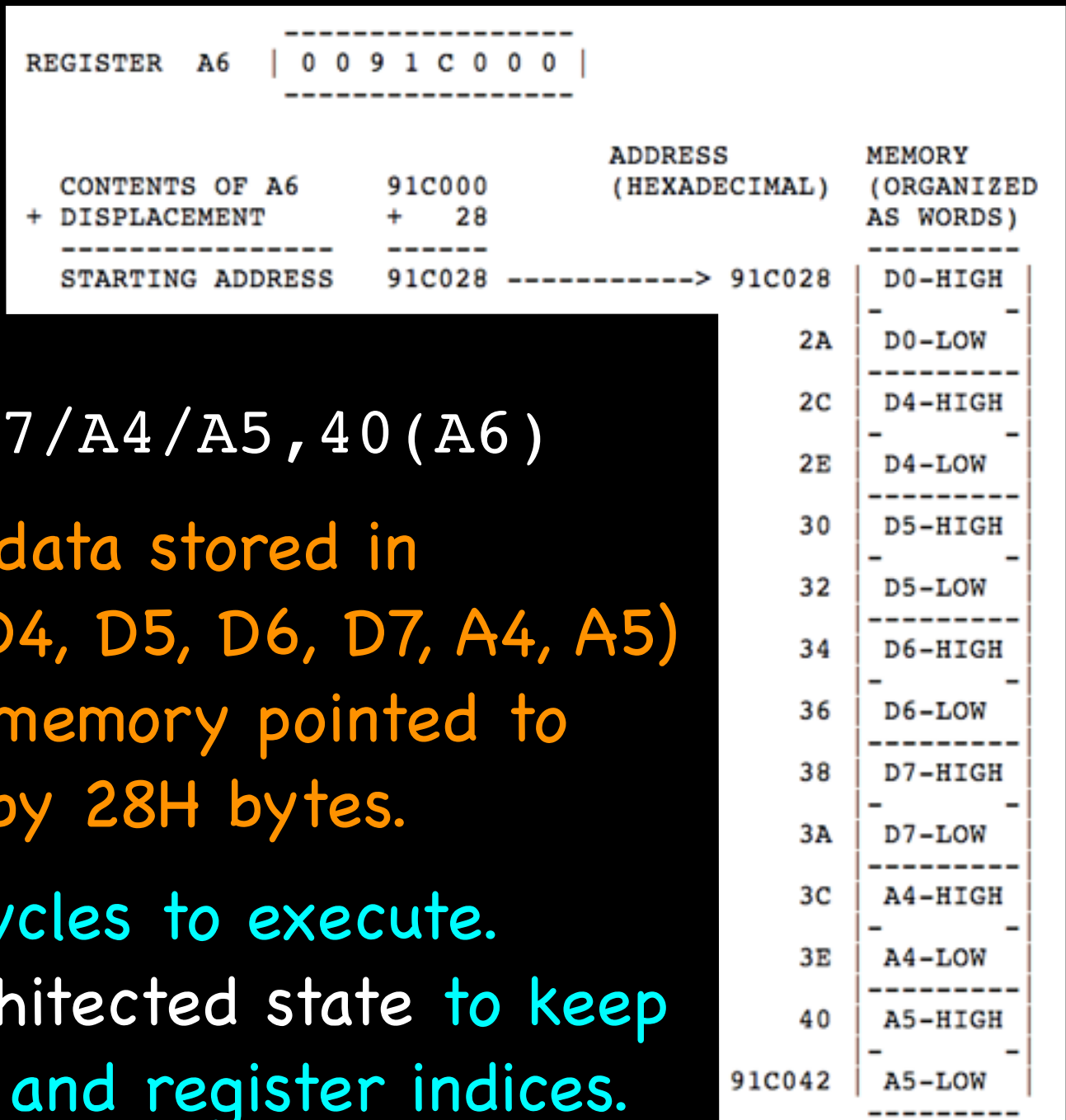
In the general case, register allocation task is NP-complete ...

There are good heuristic solutions, but they require 16 free registers (preferably more) to work well.

This line of reasoning quantifies one advantage of 32 general purpose registers

An example of a complex instruction

8 byte instruction fetch amortized by 28 byte data move




```
MOVEM.L D0/D4-D7/A4/A5,40(A6)
```

Move the 32-bit data stored in 7 registers (D0, D4, D5, D6, D7, A4, A5) to the region of memory pointed to by A^, displaced by 28H bytes.

Takes 58 clock cycles to execute.

Requires non-architected state to keep track of memory and register indices.

But ... what exactly is microcode?

Each clock cycle, we can think of this data path as executing an 11-bit microcode instruction word:

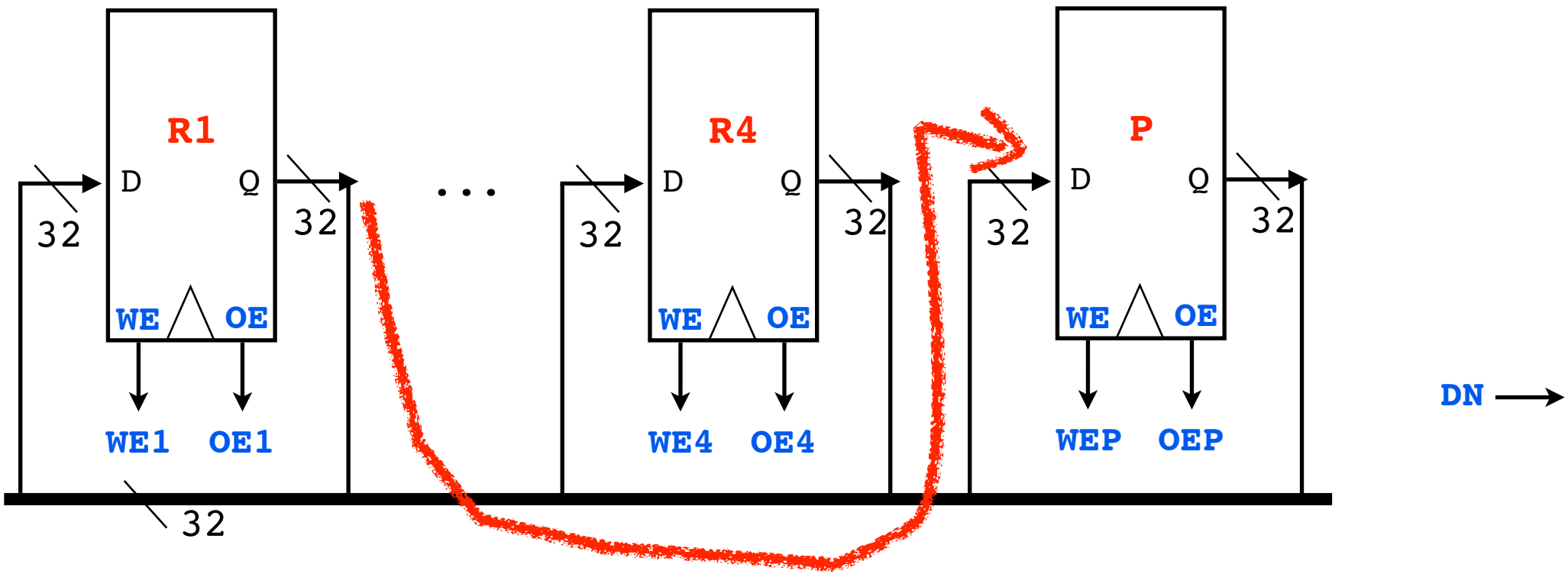
One microcode instruction - binary format

DN	WE1	WE2	WE3	WE4	WEP	OE1	OE2	OE3	OE4	OEP
0	0	0	0	0	1	1	0	0	0	0

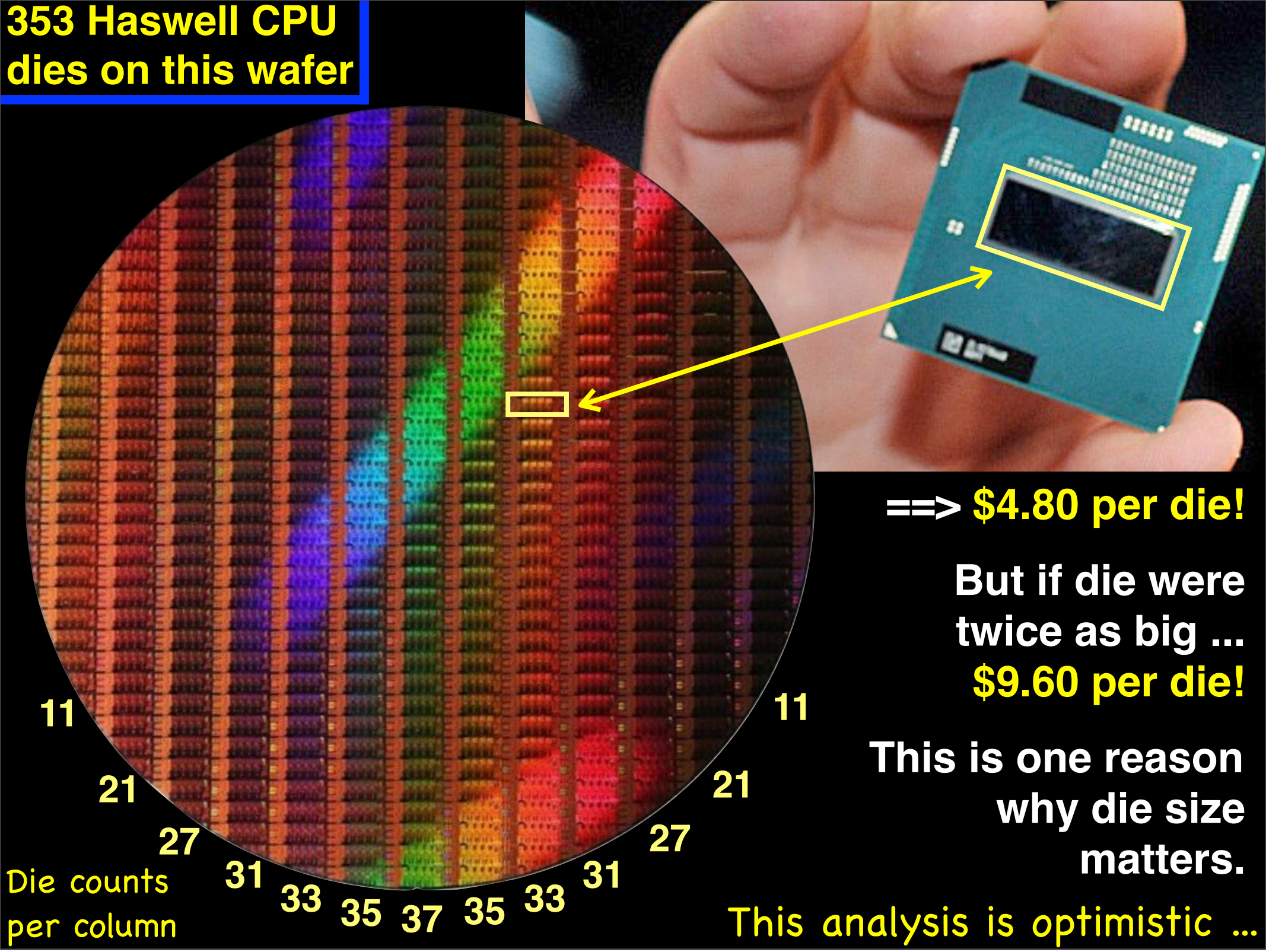
Assembler format

```
OE1, WEP;
```

a list of "1" columns.



353 Haswell CPU dies on this wafer



==> $4.80 per die!

But if die were twice as big ... $9.60 per die!

This is one reason why die size matters.

This analysis is optimistic ...

CS 152
Computer Architecture and Engineering
Lecture 6 – Superpipelining + Branch Prediction

2014-2-6

John Lazzaro

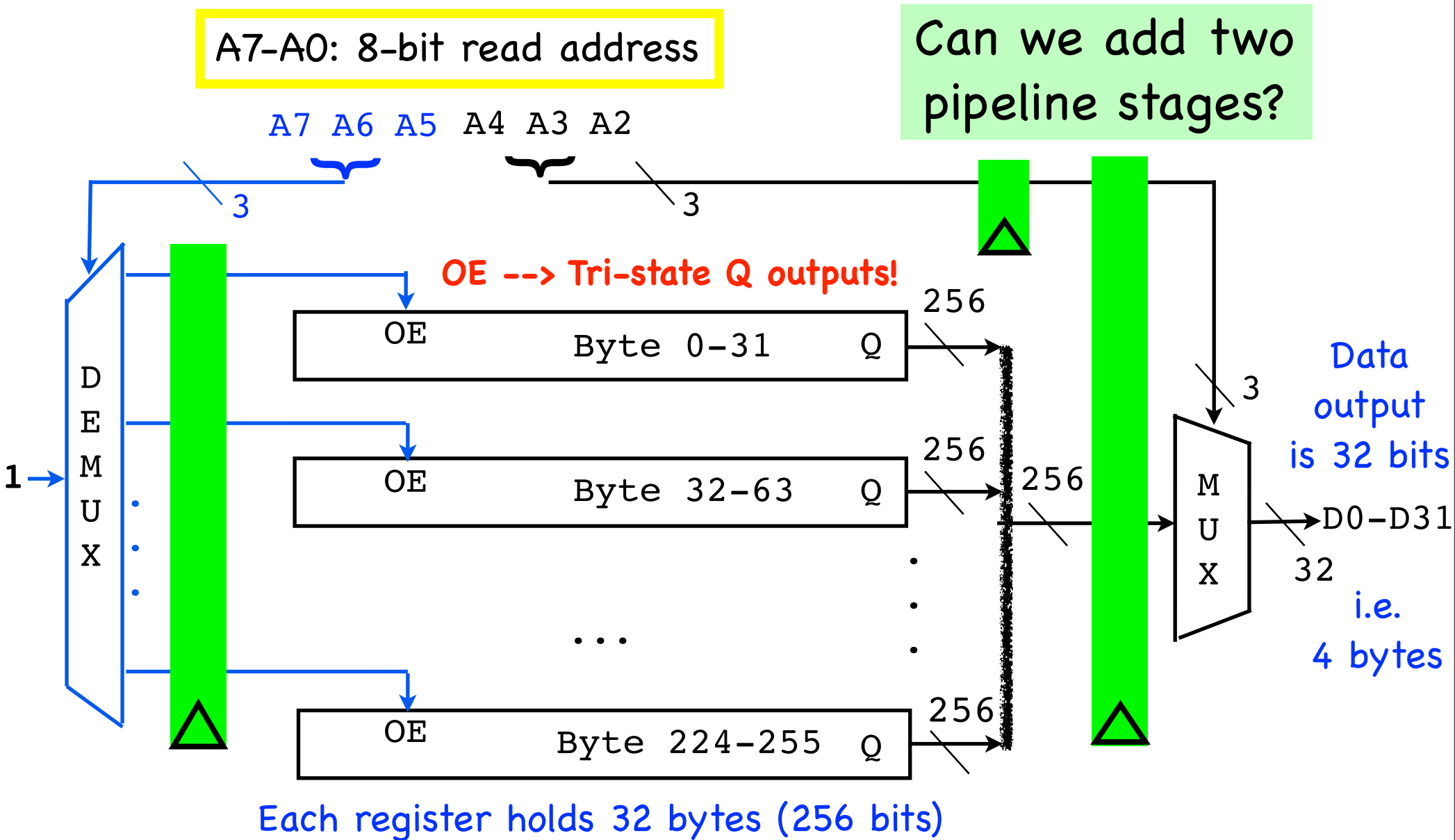
(not a prof - "John" is always OK)

TA: Eric Love

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

Pipelining a 256 byte instruction memory.

Fully combinational (and slow). Only read behavior shown.



Spatial Predictors

C code snippet:

```
b1 if (aa==2)
       aa=0;
b2 if (bb==2)
       bb=0;
b3 if (aa!=bb) {
```

Idea: Devote hardware to four 2-bit predictors for BEQZ branch.

- P1: Use if b1 and b2 not taken.
- P2: Use if b1 taken, b2 not taken.
- P3: Use if b1 not taken, b2 taken.
- P4: Use if b1 and b2 taken.

Track the current taken/not-taken status of b1 and b2, and use it to choose from P1 ... P4 for BEQZ ... How?

After compilation:

We want to predict this branch.

```
      DADDIU   R3,R1,#-2
      BNEZ     R3,L1       ;branch b1   b1 (aa!=2)
      DADD     R1,R0,R0    ;aa=0
L1:   DADDIU   R3,R2,#-2
      BNEZ     R3,L2       ;branch b2   b2 (bb!=2)
      DADD     R2,R0,R0    ;bb=0
L2:   DSUBU    R3,R1,R2    ;R3=aa-bb
      BEQZ     R3,L3       ;branch b3   b3 (aa==bb)
```

Can b1 and b2 help us predict it?

CS 152
Computer Architecture and Engineering
Lecture 7 -- Power and Energy

2014-2-11

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

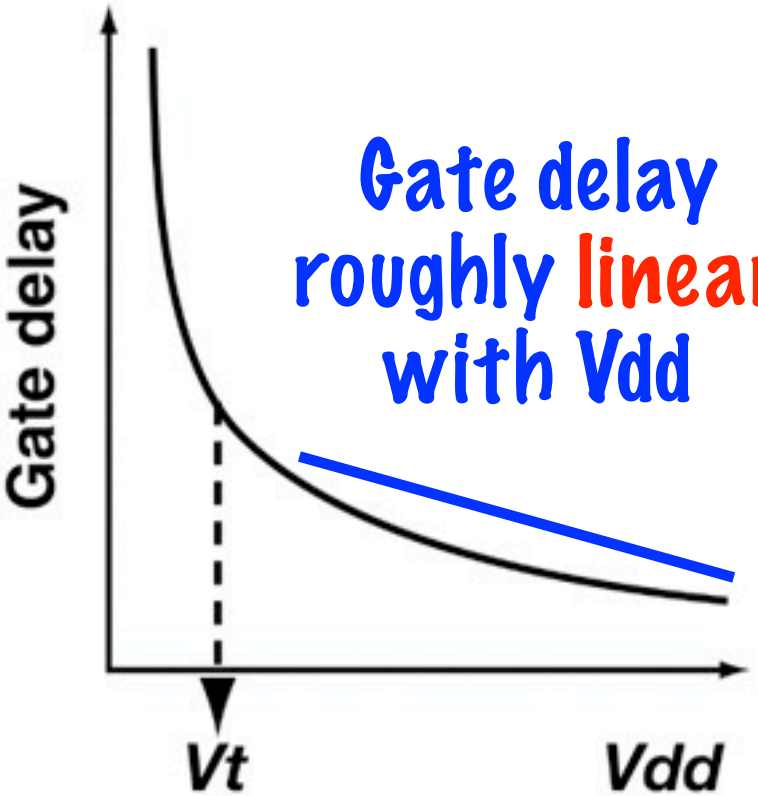
www-inst.eecs.berkeley.edu/~cs152/

The Watt: Unit of power. A rate of energy (J/s). A gas pump hose delivers 6 MW.

The Joule: Unit of energy. A 1 Gallon gas container holds 130 MJ of energy.

120 KW: The power delivered by a Tesla Supercharger. Tesla Model S has a 306 MJ battery (good for 265 miles).

1 J = 1 W s. 1 W = 1 J/s.



Gate delay roughly linear with Vdd

And so, we can transform this:

Vdd

Logic Block →

$P \sim F \times V_{dd}^2$

$P \sim 1 \times 1^2$

Freq = 1
Vdd = 1
Throughput = 1
Power = 1
Area = 1
Pwr Den = 1

Block processes stereo audio. 1/2 of clocks for "left", 1/2 for "right".

Into this:

Top block processes "left", bottom "right".

Vdd/2

Logic Block →

Logic Block →

Freq = 0.5
Vdd = 0.5
Throughput = 1
Power = 0.25
Area = 2
Pwr Den = 0.125

$P \sim \#blks \times F \times V_{dd}^2$

$P \sim 2 \times 1/2 \times 1/4 = 1/4$

CV^2 power only

THIS MAGIC TRICK BROUGHT TO YOU BY CORY HALL ...

CS 152
Computer Architecture and Engineering
Lecture 8 -- CPU Verification

2014-2-13

John Lazzaro

(not a prof - "John" is always OK)

TA: Eric Love

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

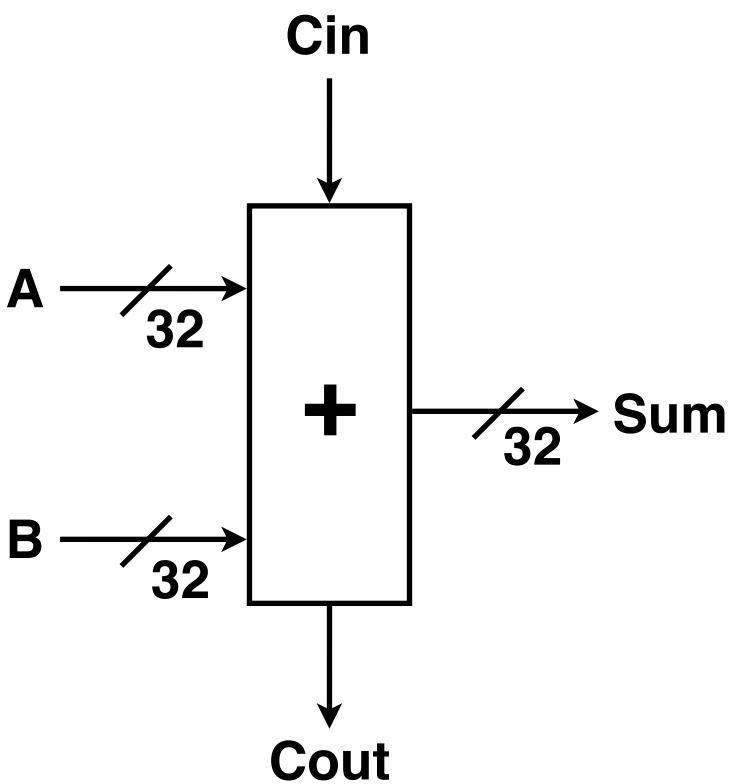
"CPU program" diagnosis is tricky ...

Observation: On a buggy CPU model, the correctness of every executed instruction is suspect.

Consequence: One needs to verify the correctness of instructions that surround the suspected buggy instruction.

Depends on: (1) number of "instructions in flight" in the machine, and (2) lifetime of non-architected state (may be "indefinite").

Combinational Unit Testing: 32-bit Adder



Number of input bits ? 65

Total number of possible input values?

$2^{65} = 3.689e+19$

Just test them all?

Exhaustive testing does not "scale".

"Combinatorial explosion!"

On Tuesday

Mid-term I ...

T 3/18	Midterm I

Ground rules ...

When is it? Where is it? Ground rules.

- 9:30 AM sharp, Tuesday March 18th, 306 Soda.
- Every-other-seat seating, except for the front row, where every-seat is permitted.
- No blue-books needed. We will be handing out a paper test. Pencil is preferred.
- Pencils down @ 10:55 AM, so we can collect papers before next class comes in.

When is it? Where is it? Ground rules.

- No use of calculators, smartphones, laptops, etc ... during the exam.
- Closed-book, closed-notes. Just pencils, erasers. No consulting with students.
- Restroom breaks are OK, but you'll still need to hand in your exam @ 10:55.
- Questions are reserved for serious concerns about a bug in the question.

Today - Midterm I Review Session

All questions answered (almost ...)

Break

Play:

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!

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

Now at Splunk (log files in the cloud)

Now at Redux (10-second online videos)

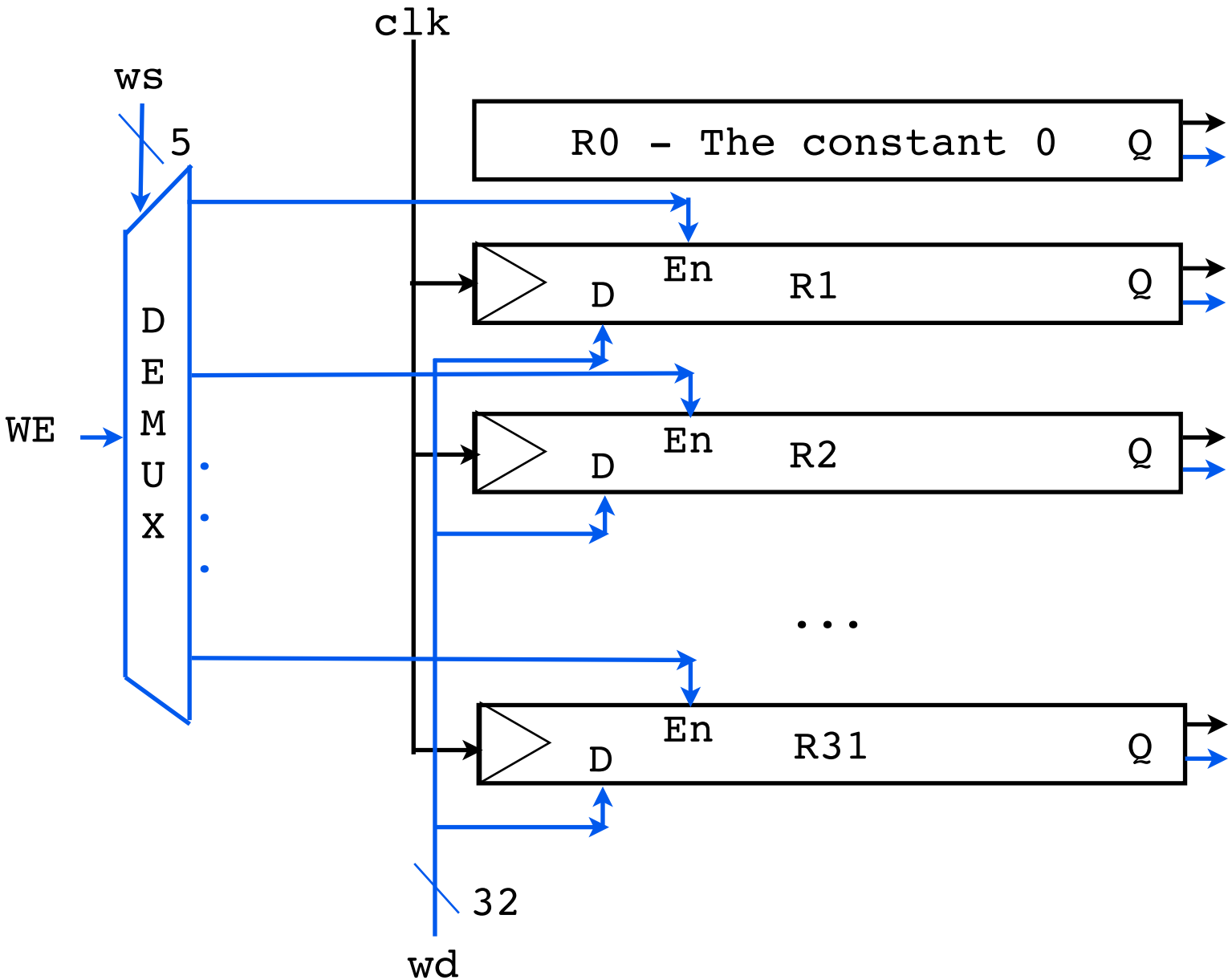
David Marquardt

Udam Saini

John Lazzaro (still @ berkeley)

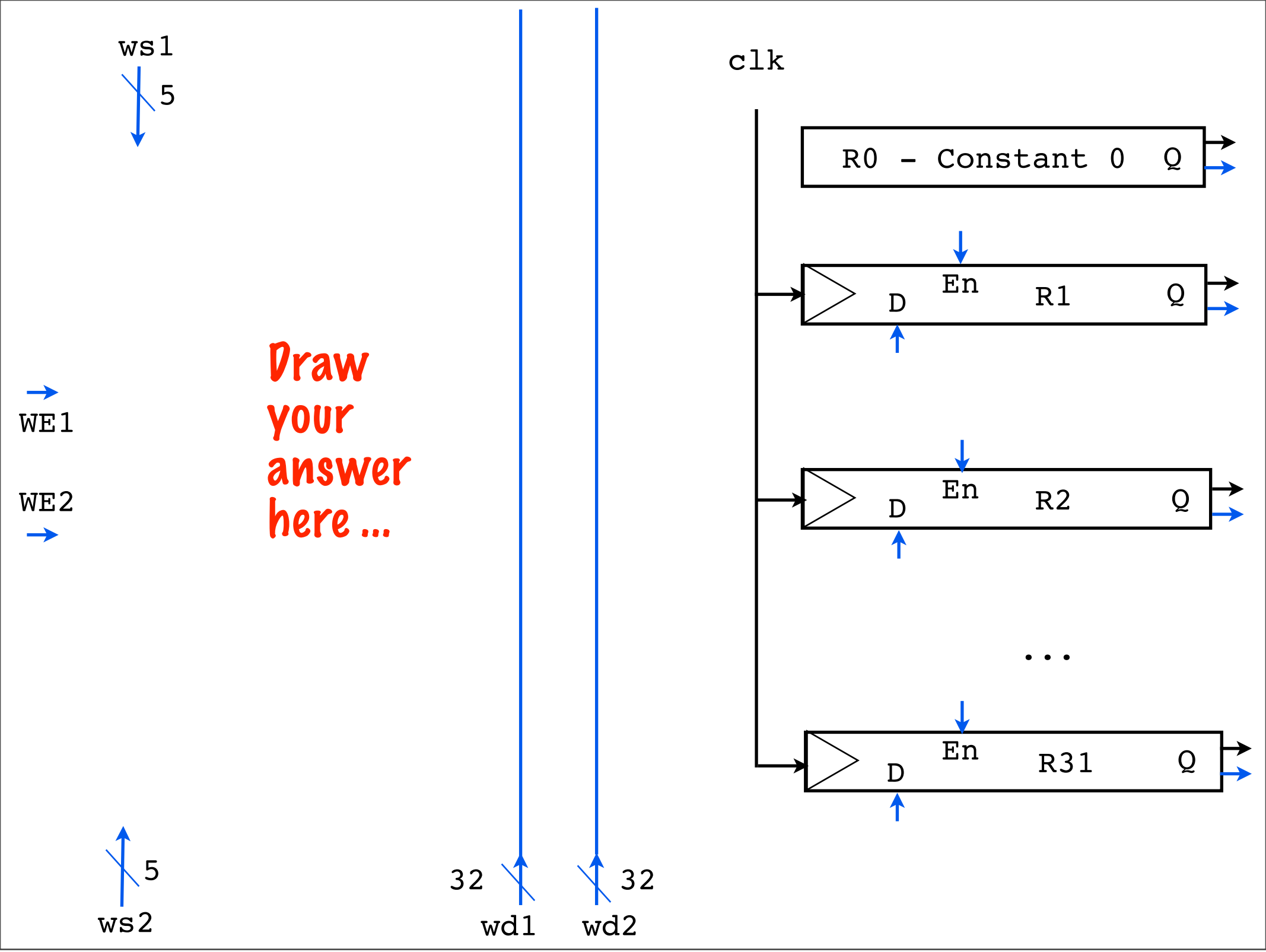
Q1. Register File Design (10 points)

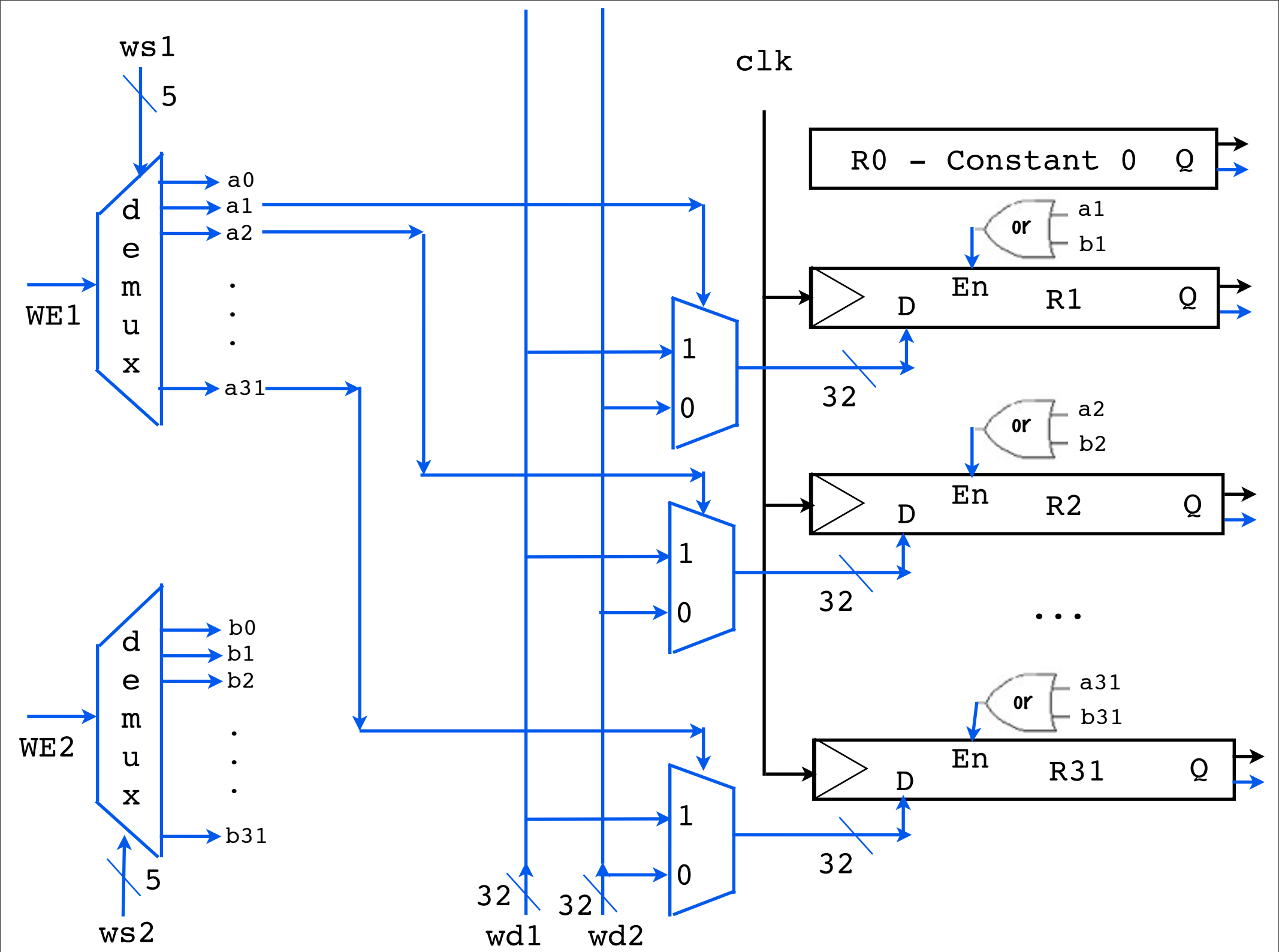
On the top slide on the next page, we show the write logic for the register file design we showed in Lecture 1-2.



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
demux
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
demux
WE2
5
ws2
or
a31
b31
En
D
R31
Q
1
0
32
32
32
wd1
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)
```

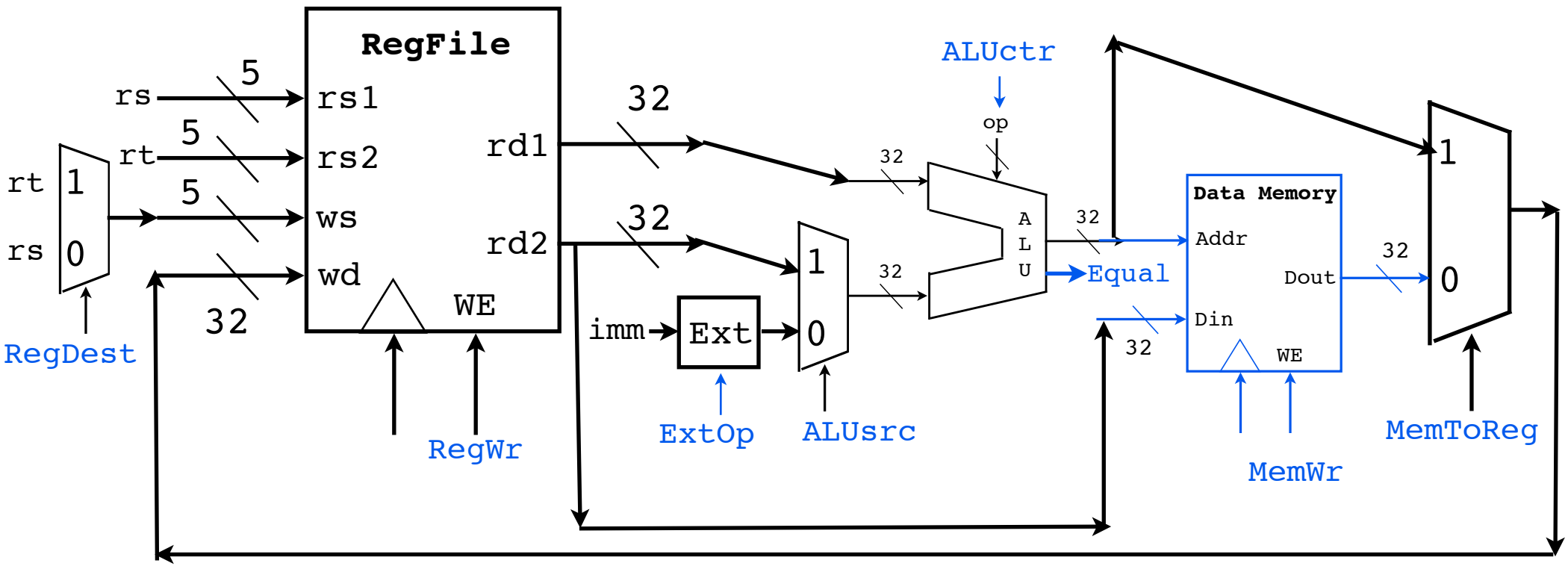
```
$rt = M[$rs + sign_extended(imm)]
$rs = $rs + sign_extended(imm)
```

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

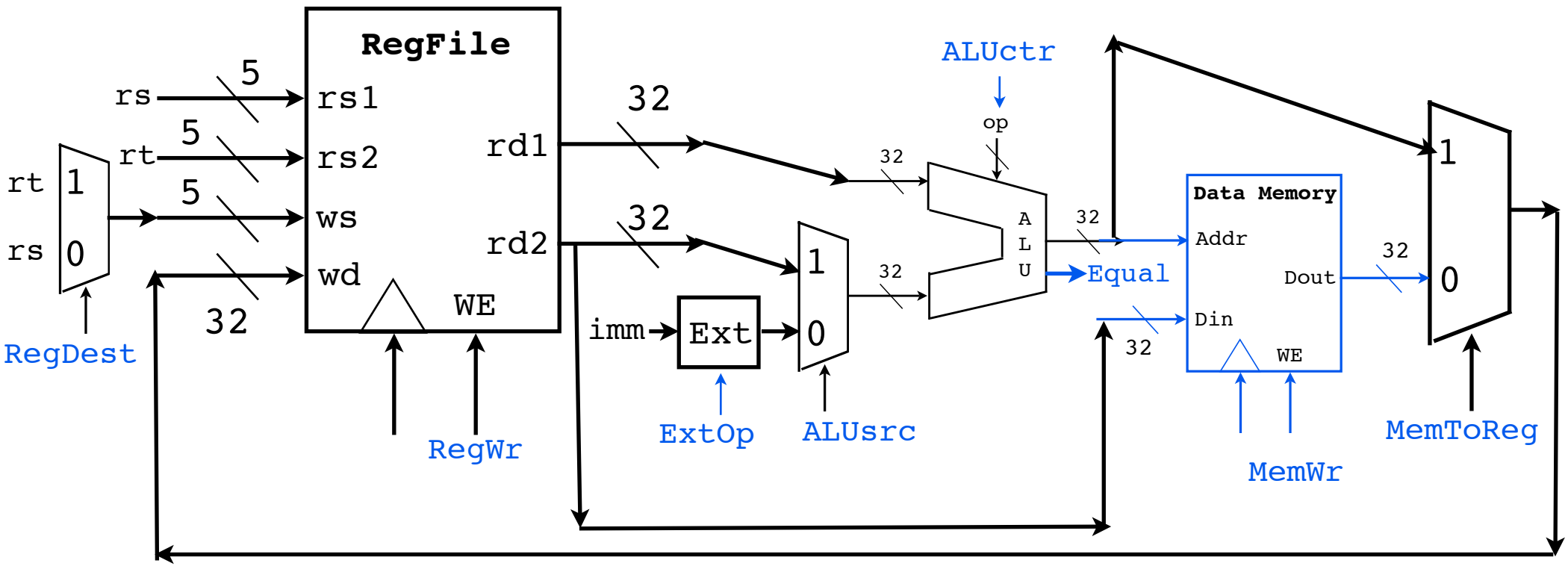
SWA $rt imm($rs)	M[$rs + sign_extended(imm)] = $rt $rs = $rs + sign_extended(imm)

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

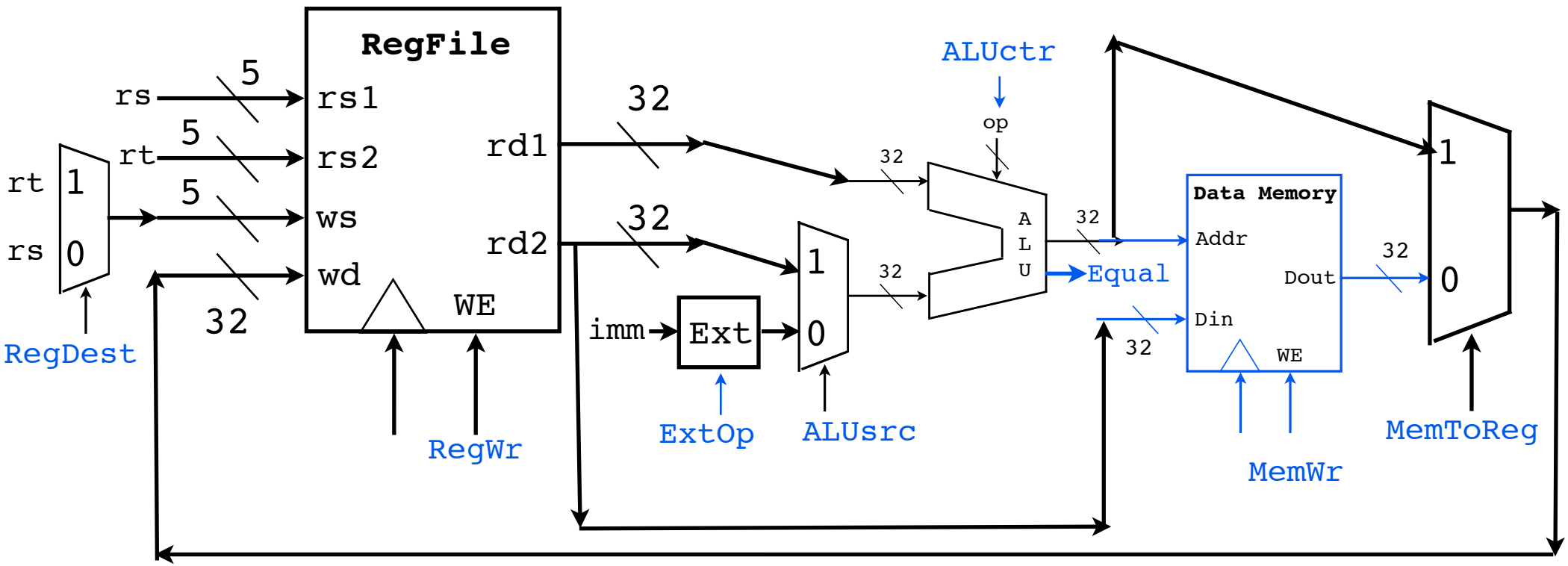
```
BEQR $rs imm $rt     if ($rs == sign_extended(imm))
                        PC = PC + 4 + $rt
                     else
                        PC = PC + 4
```

R-format

op	rs	rt	rd	shamt	funct

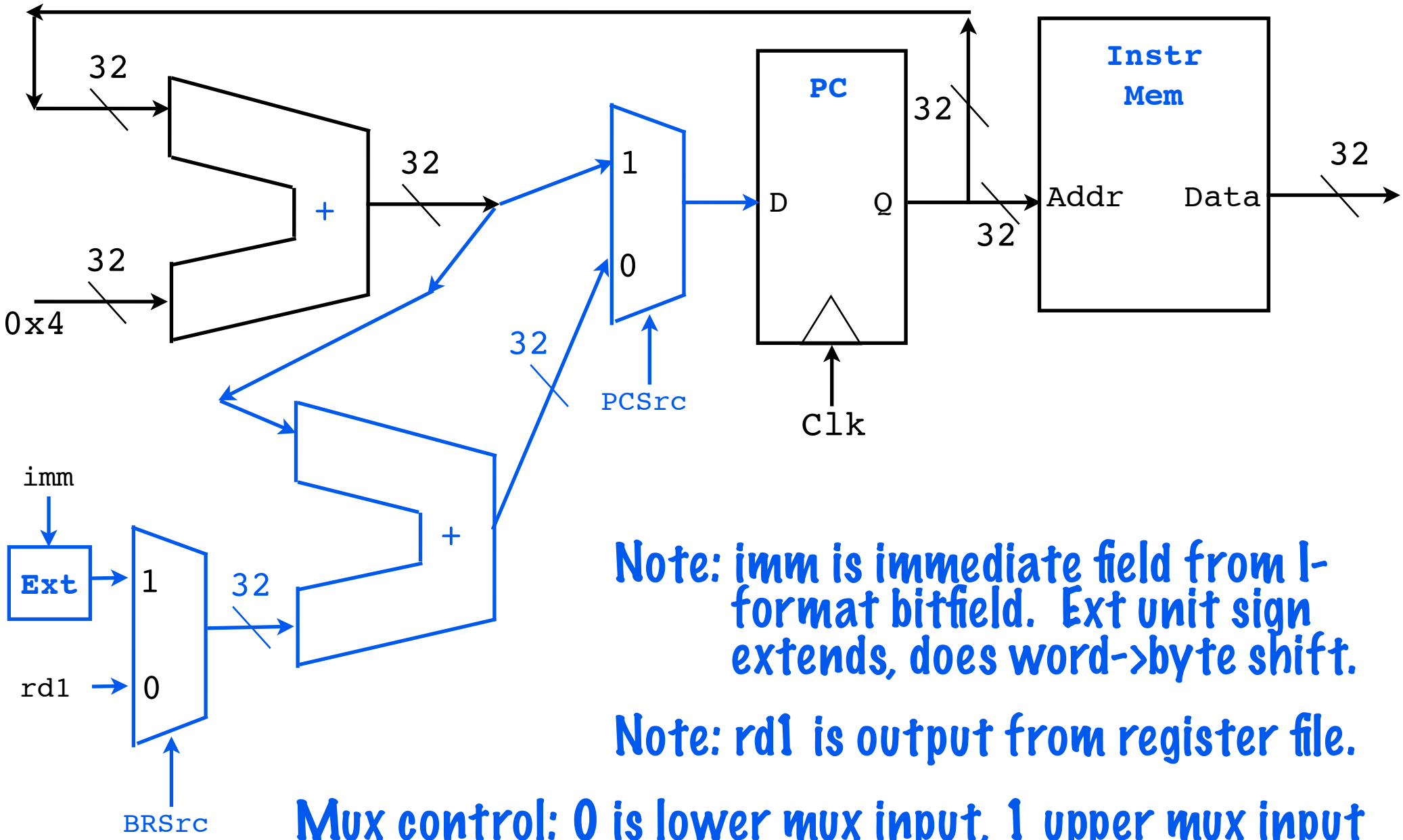
I-format

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.

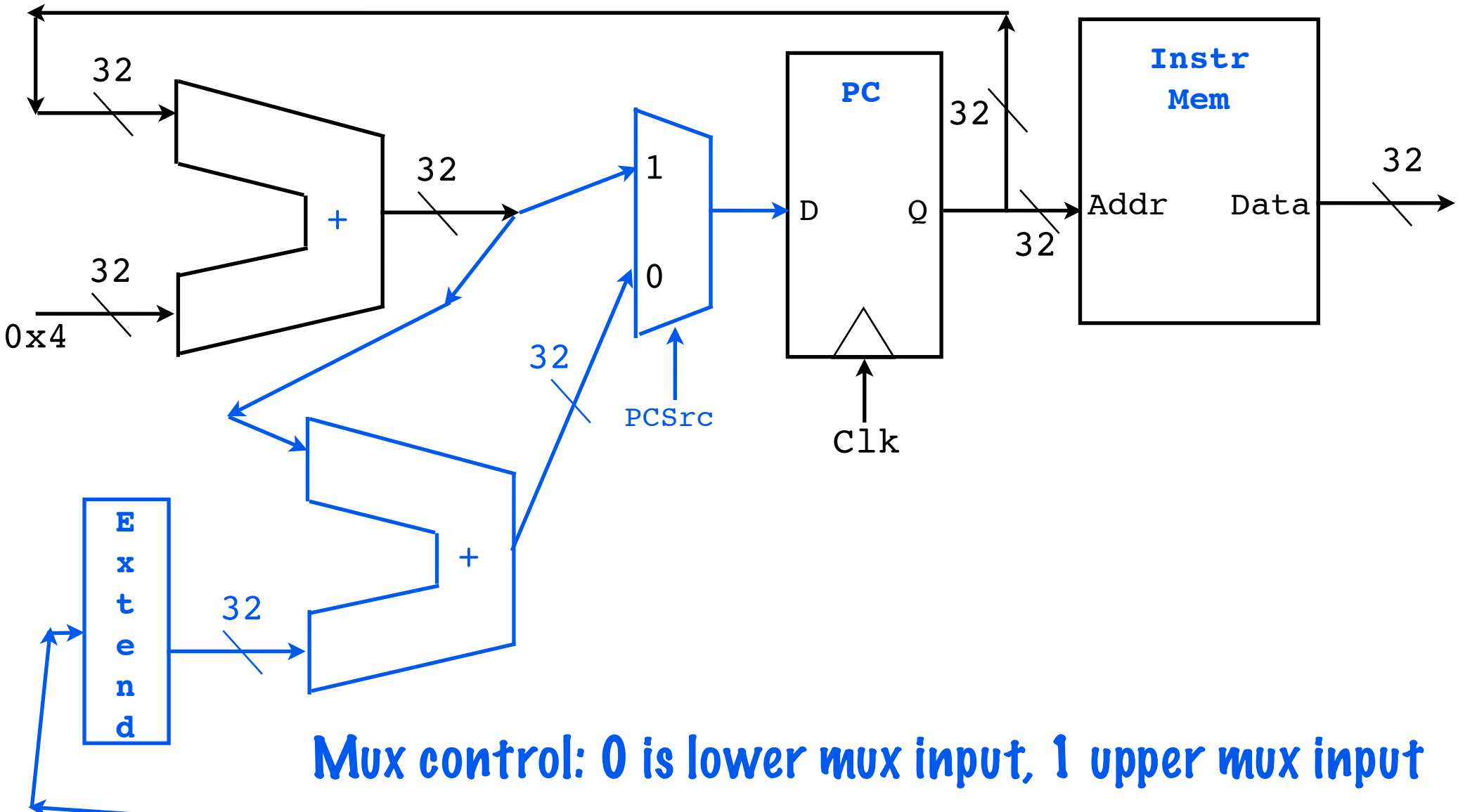
Q3: Single-Cycle Branch Delay Slot

3 Branch Delay Slot (10 points)

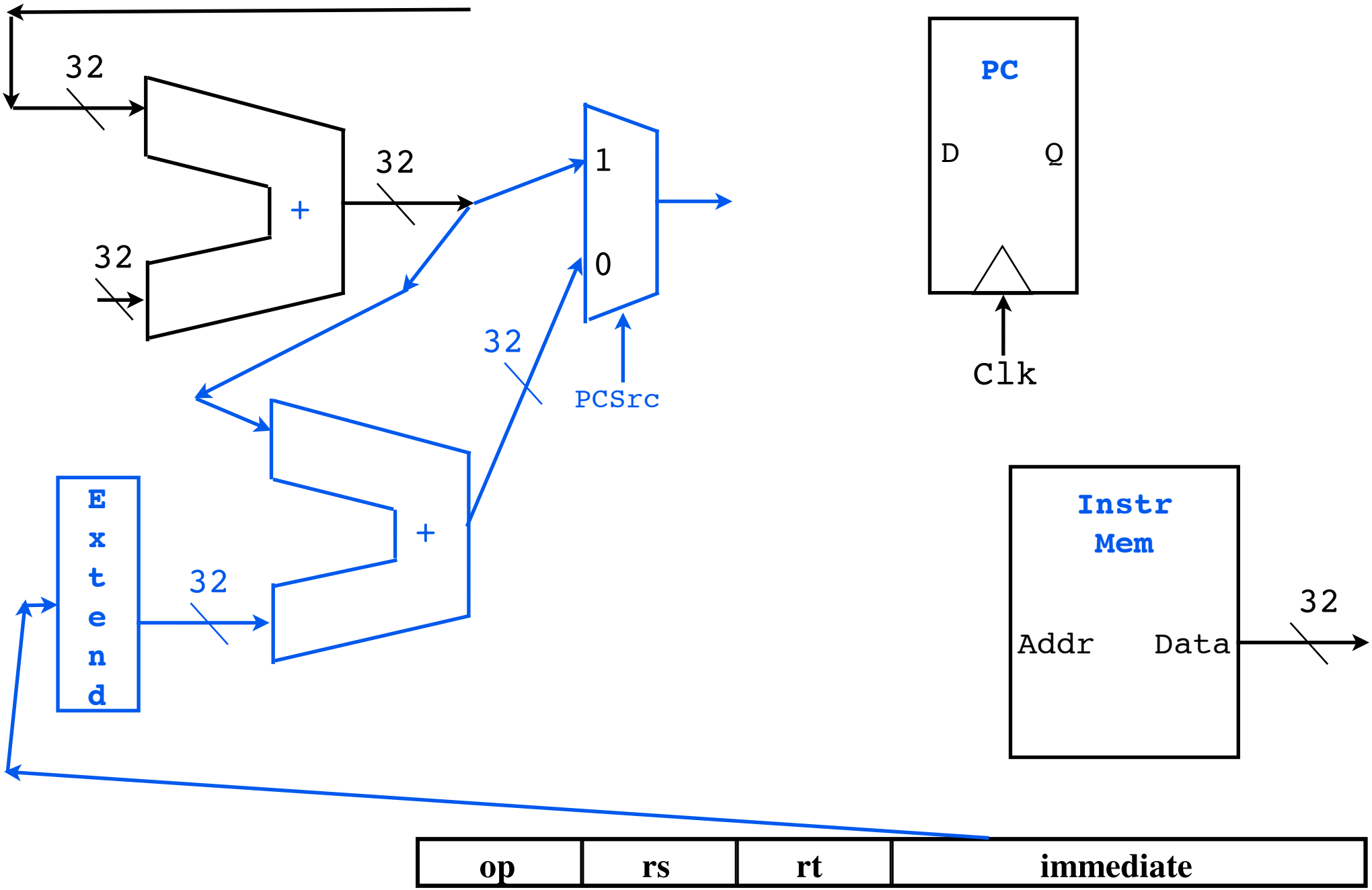
On the top slide on the next page, we show the branch logic for the single-cycle processor showed in Lecture 1-2. Recall that this processor does not use the branch delay slot.

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.

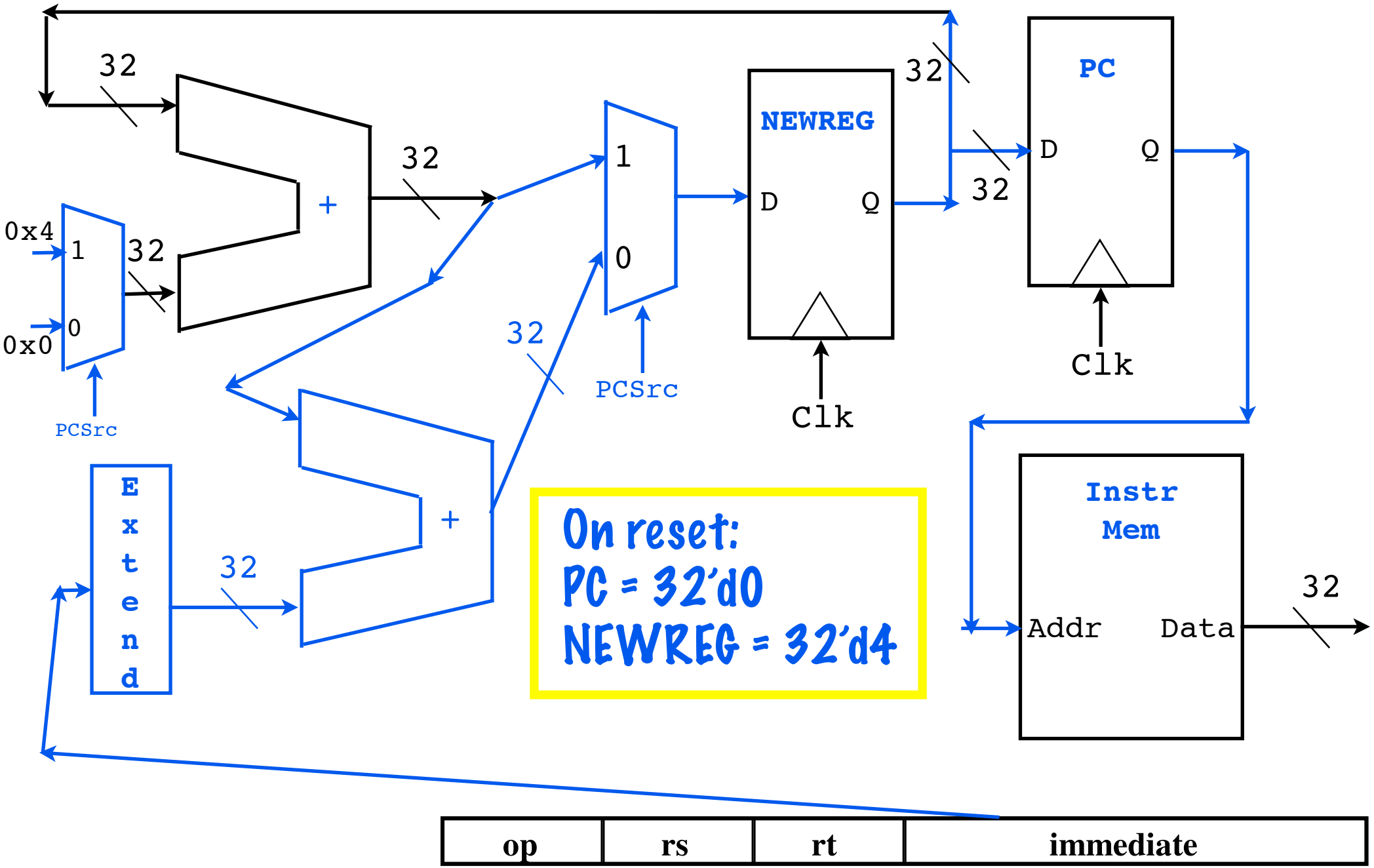
Single-Cycle I-Fetch (no delay slot)



Design Instruction Fetch WITH delay slot



Design Instruction Fetch WITH delay slot



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 ...

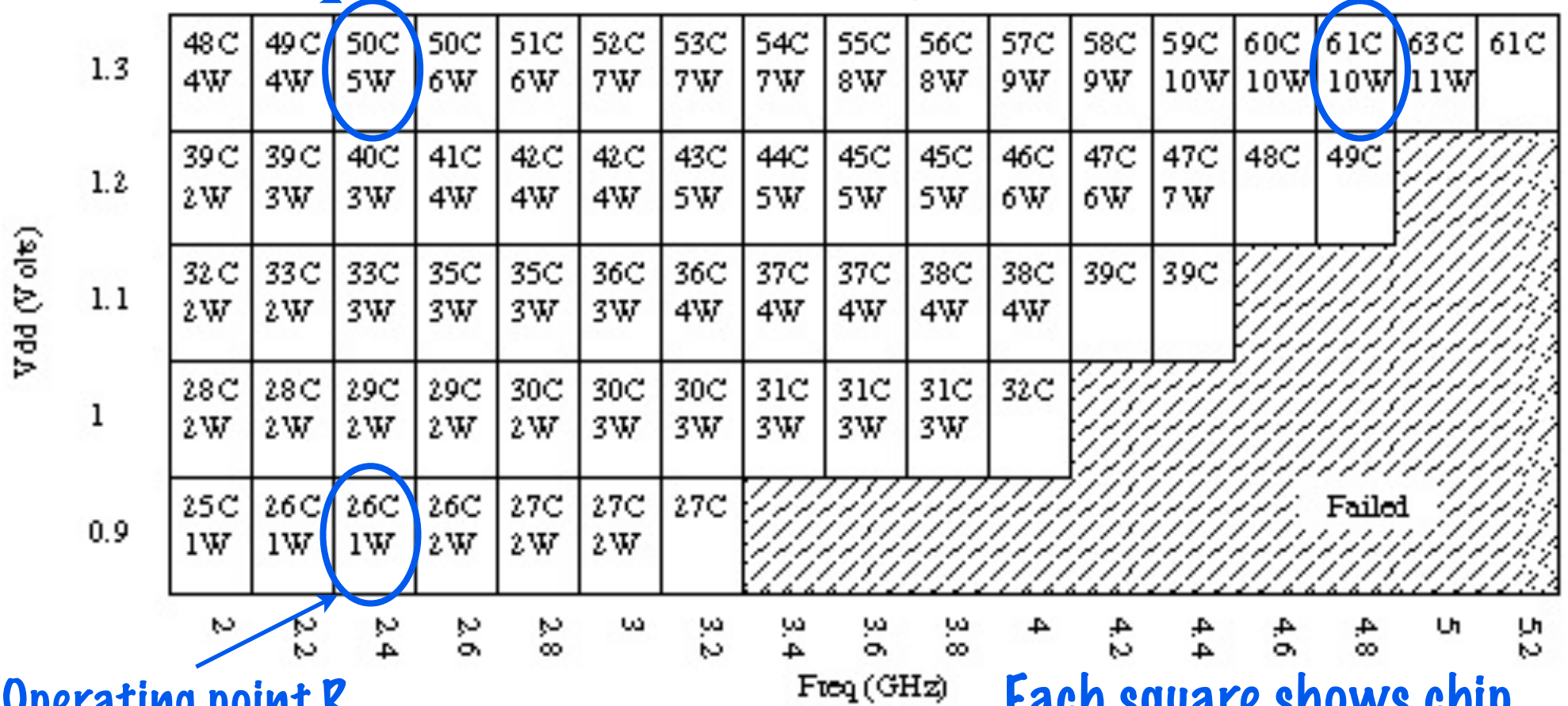
The energy equations: $E_{0->1} = \frac{1}{2} C V_{dd}^2$ $E_{1->0} = \frac{1}{2} C V_{dd}^2$

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".

Q5: Visualizing Stalls and Kills

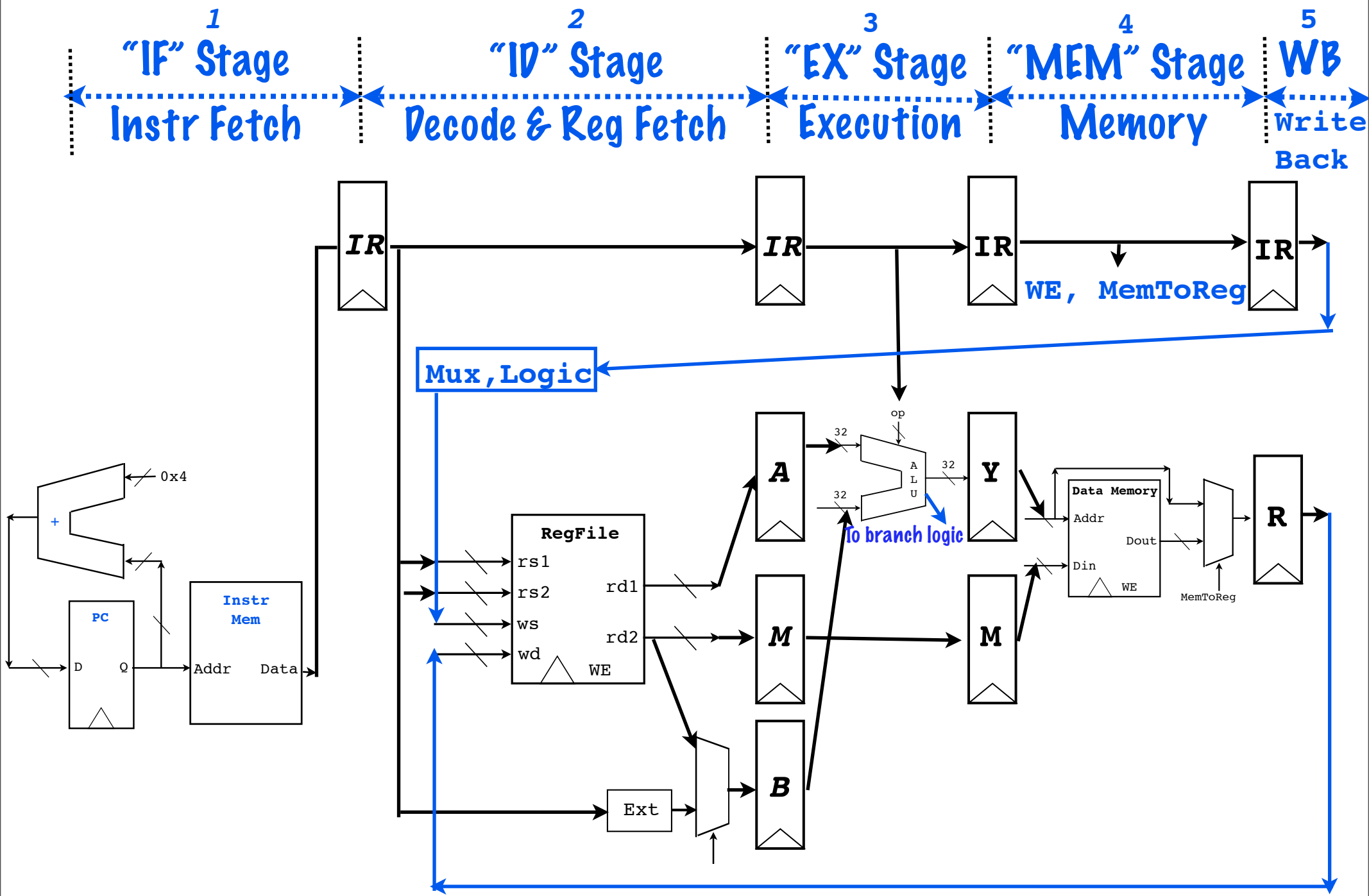
On the next page, we show the simple 5-stage MIPS pipelined processor from Lecture 4-1. This processor does not have forwarding paths and muxes, and does not have a comparator ALU for branches. The processor does have the ability to stall and to kill instructions at each stage of the pipeline. For clarity,

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 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												
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

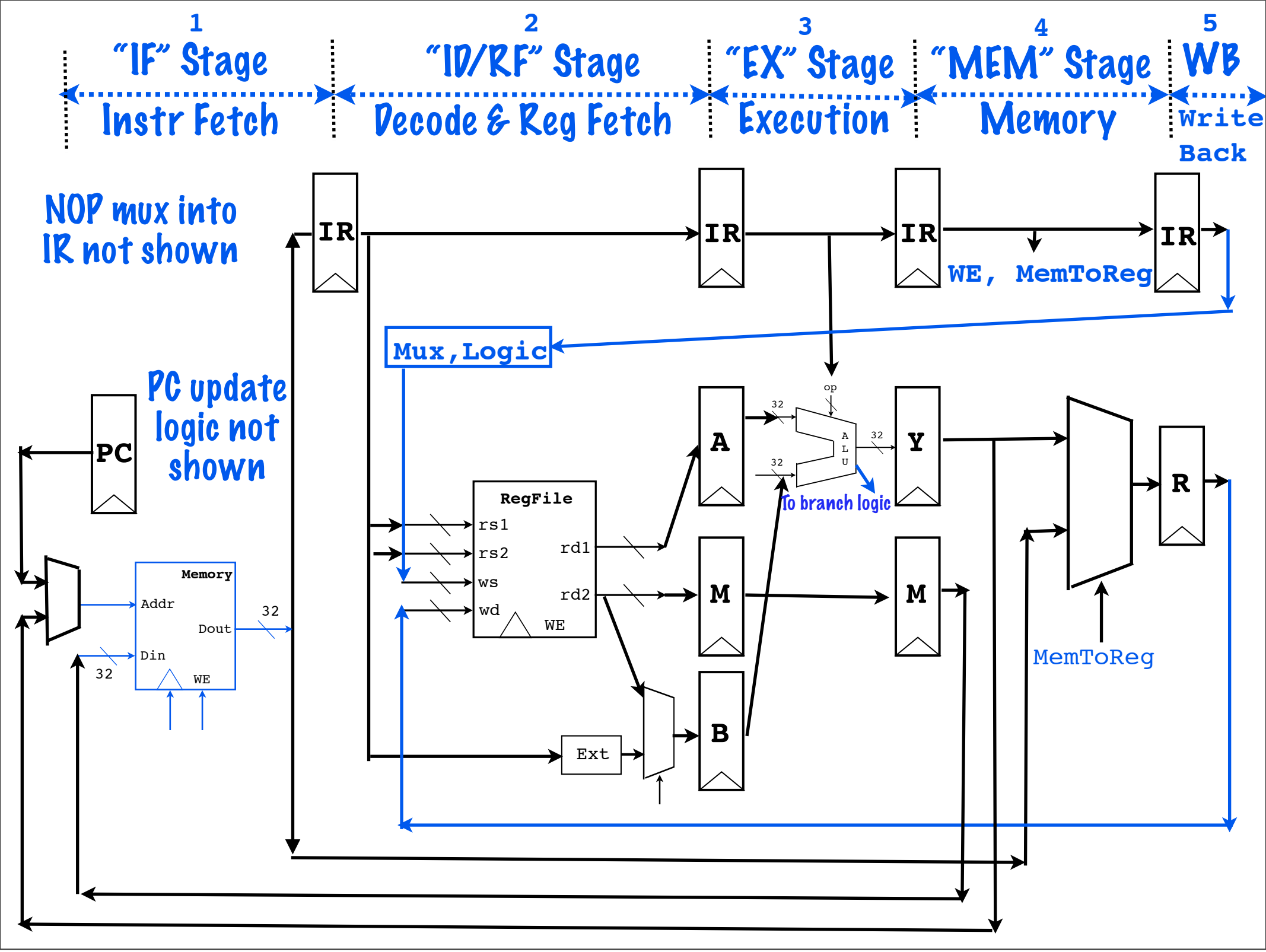
Q6: Unified Memory and Pipelines

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
RegFile
rs1
rs2
ws
wd
rd1
rd2
WE
A
B
op
32
32
32
ALU
Y
To branch logic
M
M
R
Memory
Addr
Dout
Din
WE
32
32
Ext
MemToReg

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

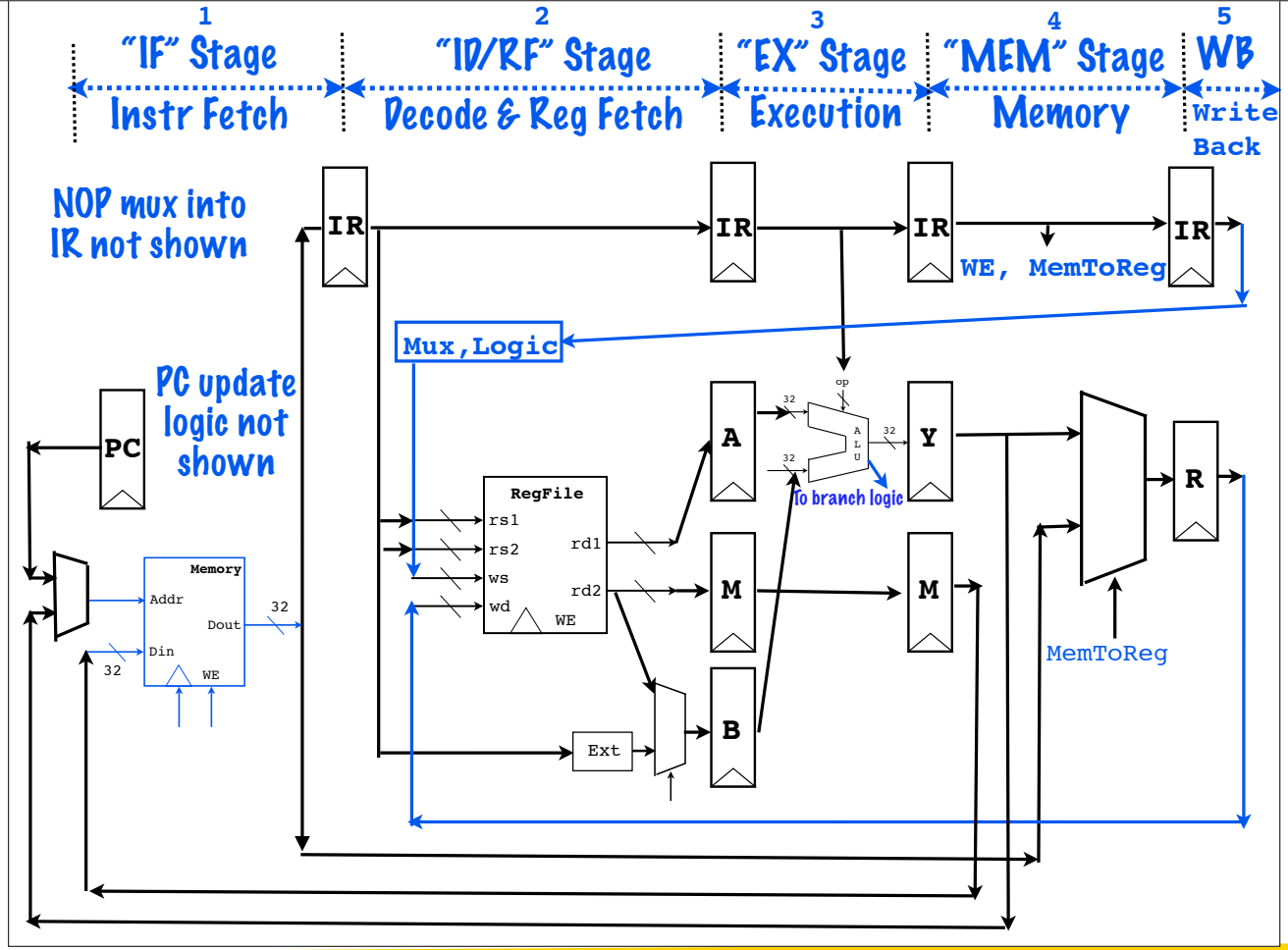
Use N to denote a stage holding a NOP.

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								



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

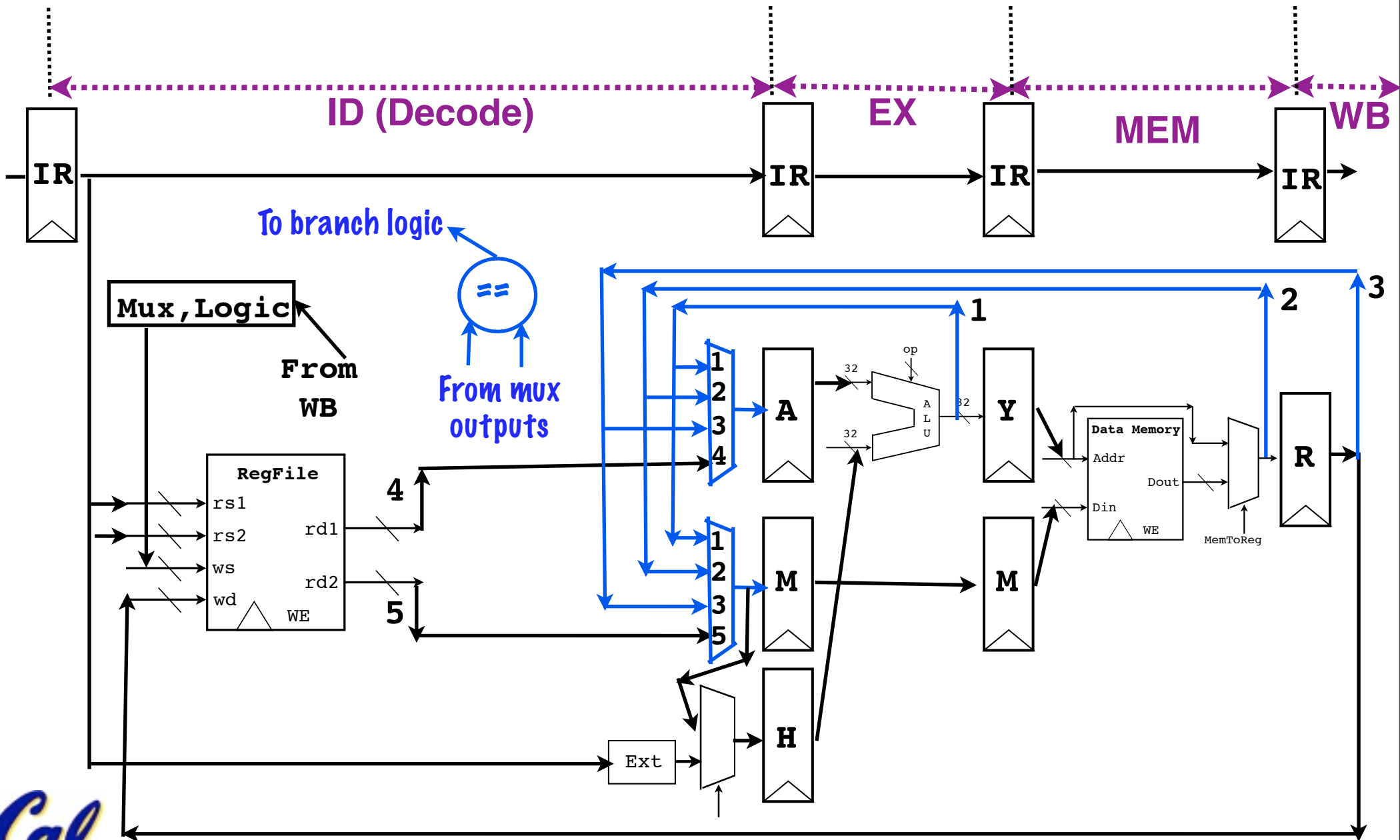
Q7: Forwarding Networks

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.

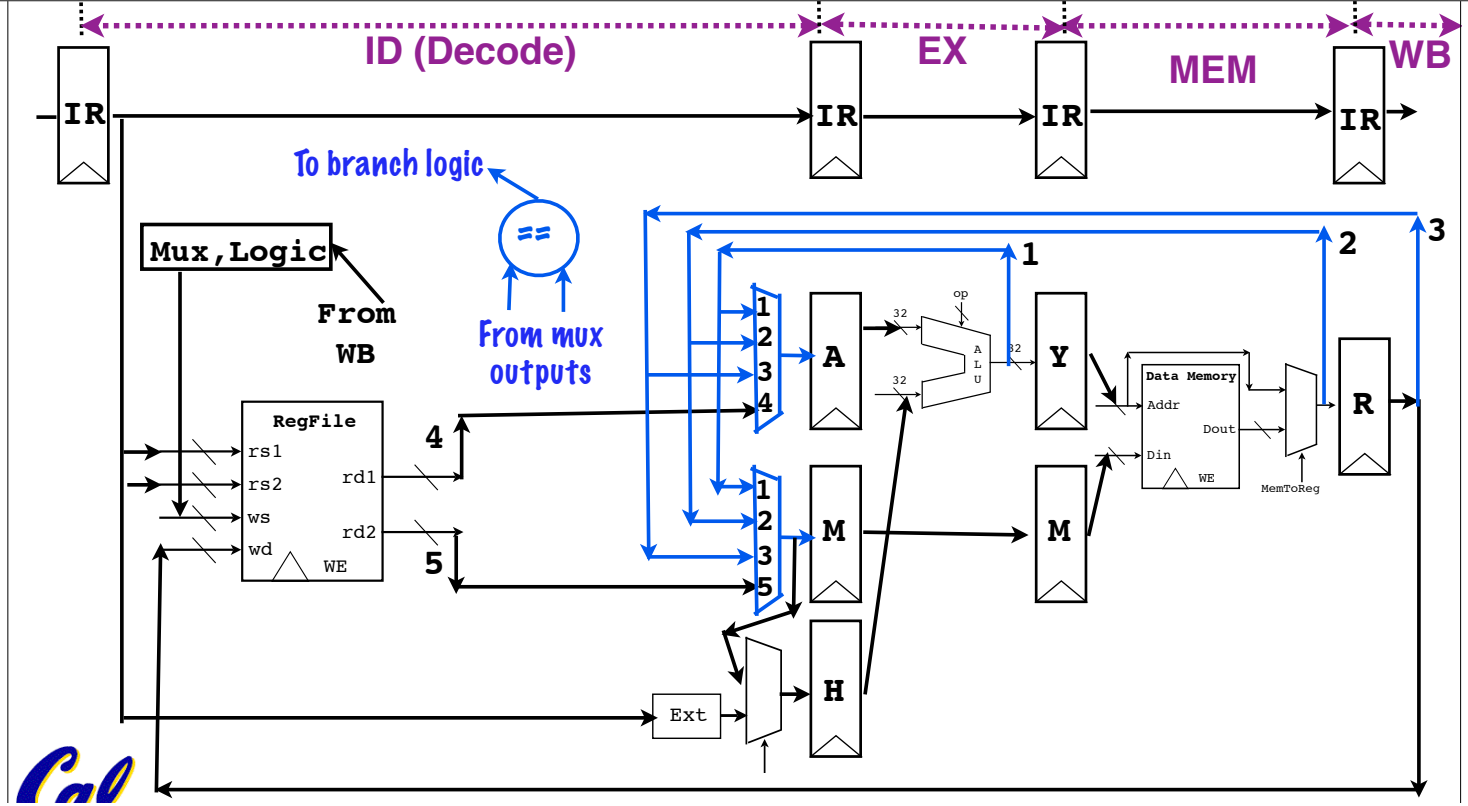
(2) Fill in A# with the selected input of the mux driving the A register needed to fulfill the programmers contract (1,2,3, 4, or X for don't care).

(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									



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

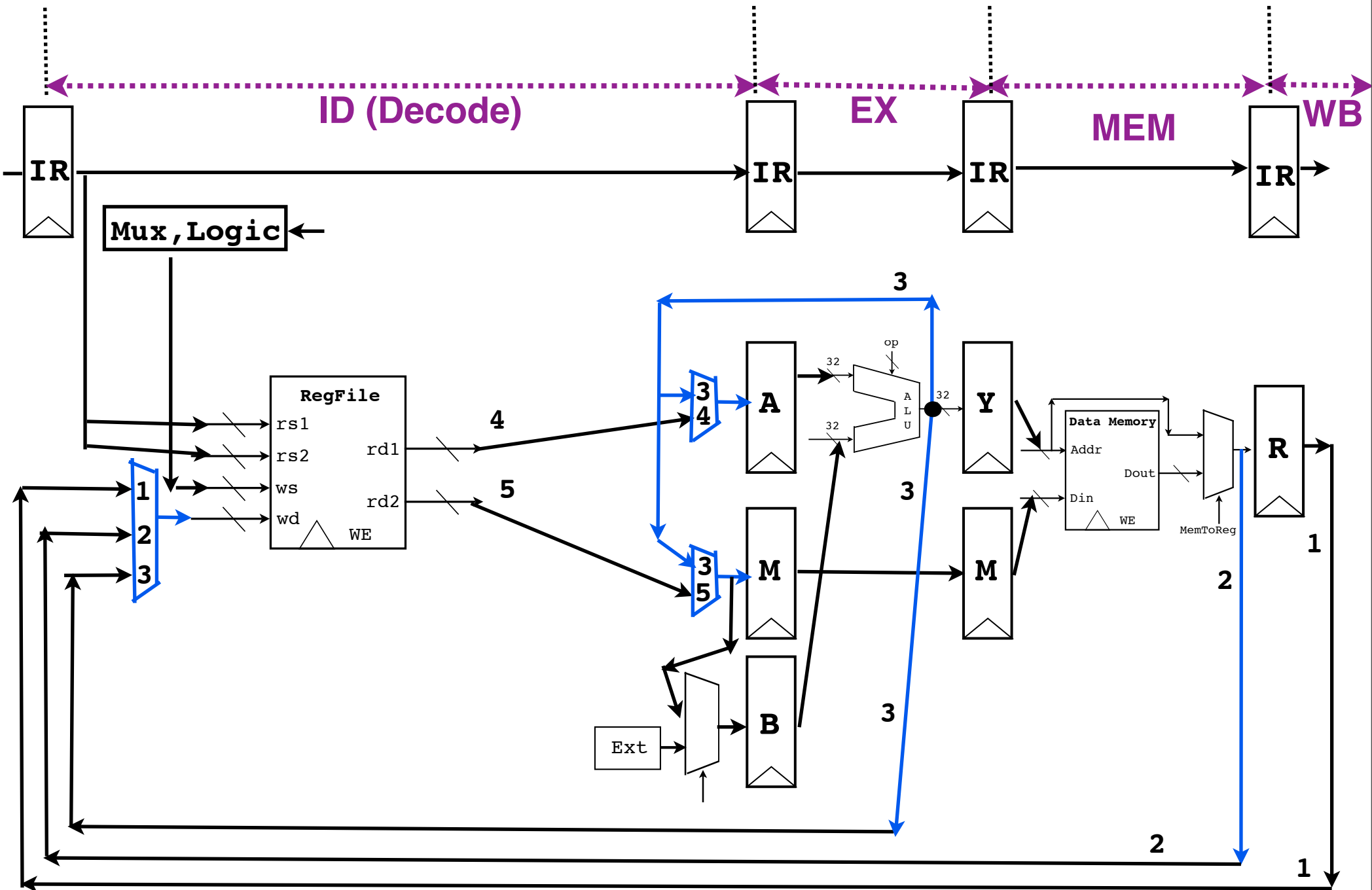
Q8: Forwarding Through Registers

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).

`LW ws,imm(rs1)`

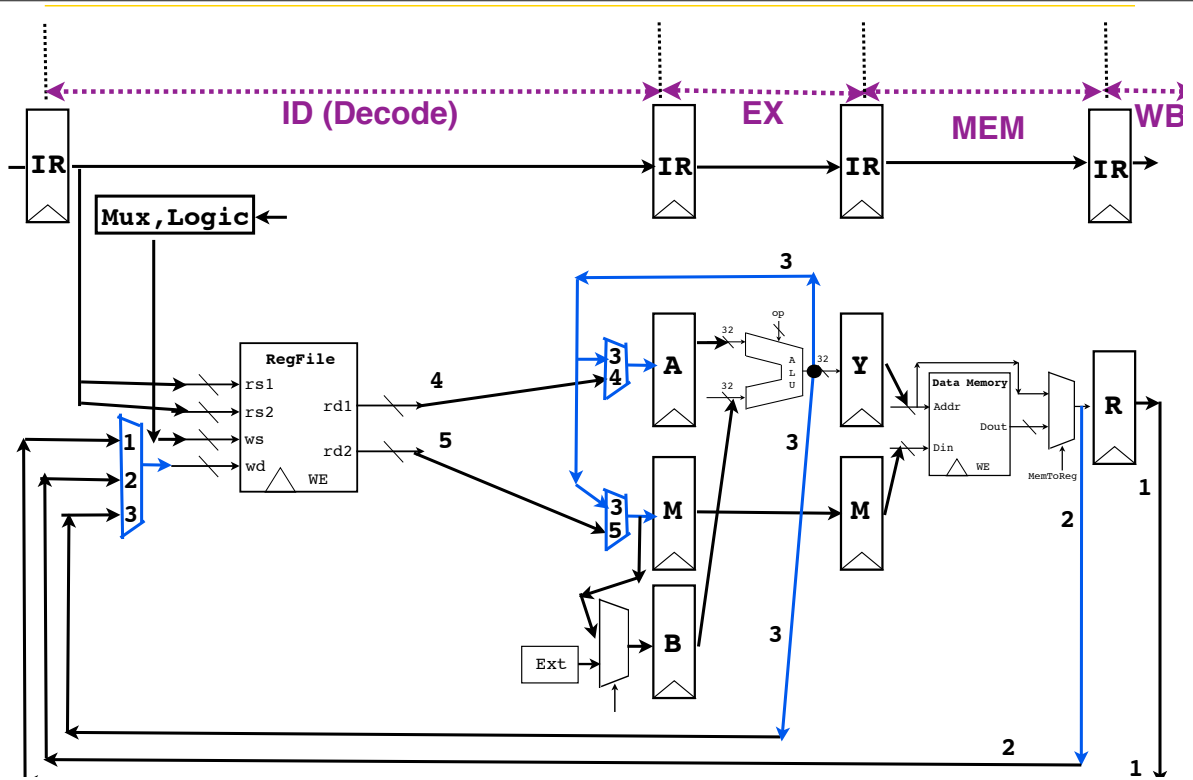
Fill in M# with the selected input of the mux driving the M register needed to fufill the programmers contract (3, 5, or X for don't care).

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												



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

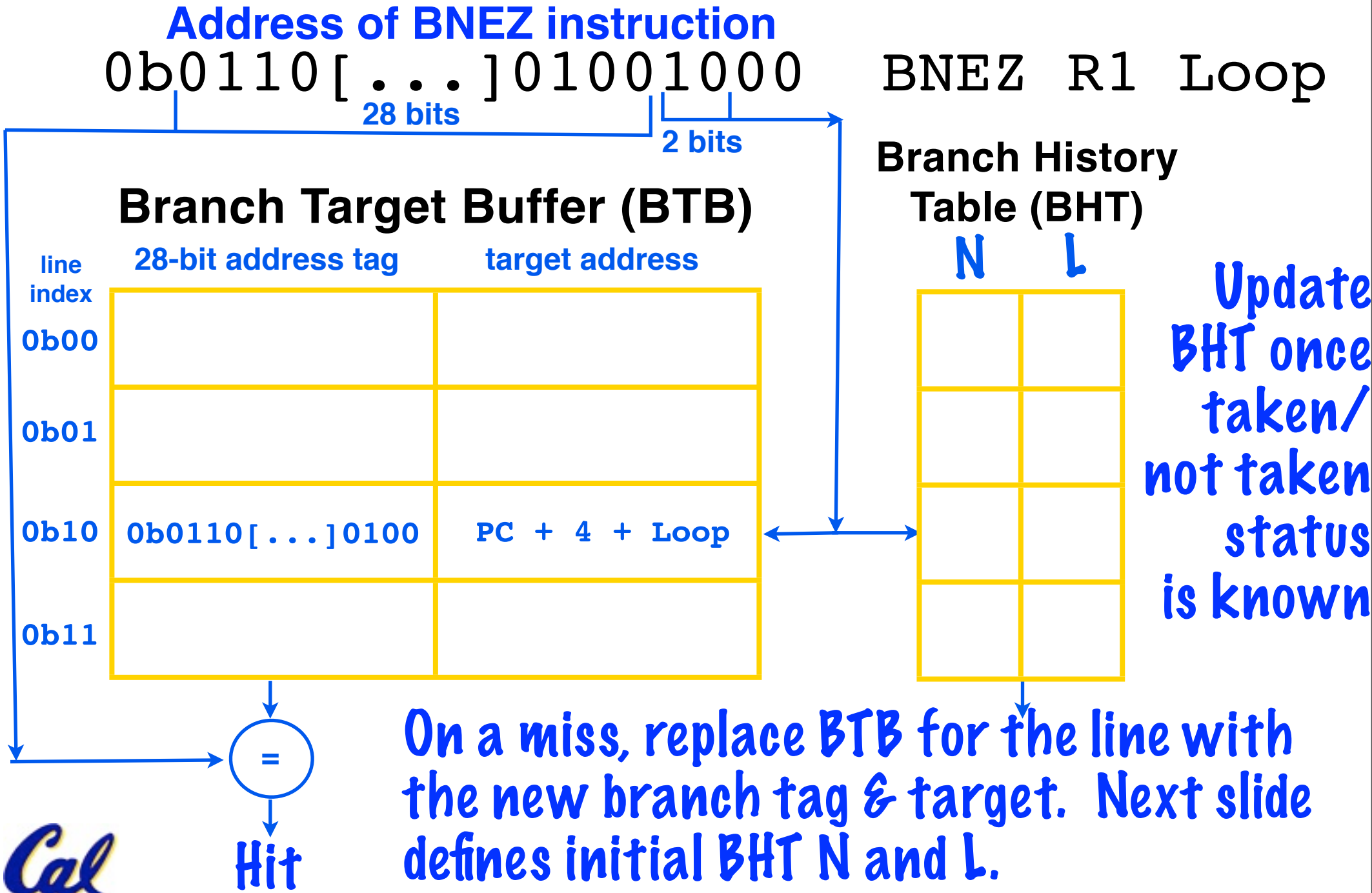
For (at least) one of the "OR RX, RY, RZ" commands in the program, it is possible to change RY or RZ to a different register number, in such a way that it becomes impossible to use the forwarding network to execute the instruction stream in a stall-free way. What is the label of this instruction ($I1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

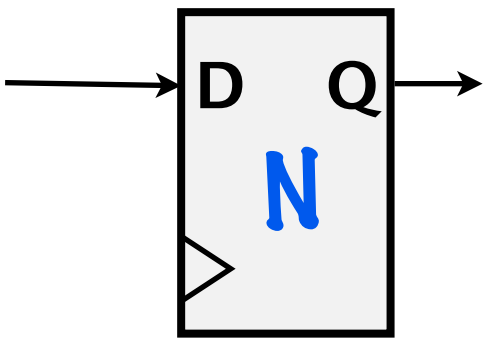
a stall is impossible to avoid.

Q9: Simple branch predictor

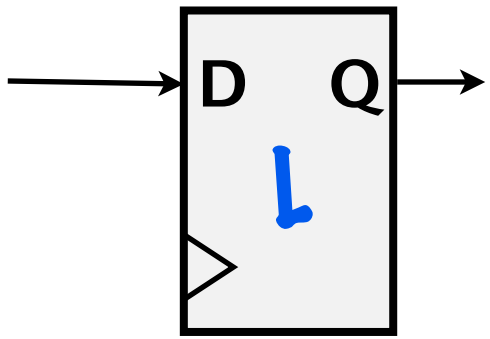


Simple ("2-bit") Branch History State

"N bit"
Prediction for Next branch (1 = take, 0 = not take)



"L bit"
Was Last prediction correct? (1 = yes, 0 = no)



old N	old L	branch	new N	new L
0	0	not taken	0	1
0	0	taken	1	1
0	1	not taken	0	1
0	1	taken	0	0
1	0	not taken	0	1
1	0	taken	1	1
1	1	not taken	1	0
1	1	taken	1	1

When replacing the tag value for a line, initialize branch history state to (N = 1, L = 1) (for taken branches) or to (N = 0, L = 1) (for "not taken" branches).

Branch predictor state before first inst. in trace executes

28-bit address tag	target address	N	T	line index
0x 0000 000	PC + 4 + Lab1	0	0	0b00
0x 0000 003	PC + 4 + Lab4	1	0	0b01
0x 0000 005	PC + 4 + Lab6	0	1	0b10
0x 0000 007	PC + 4 + Lab8	1	1	0b11

1	0x 0000 0000	BEQ R1 R2 Lab1	Taken

0x 0000 000	PC + 4 + Lab1	1	1	0b00
0x 0000 003	PC + 4 + Lab4	1	0	0b01
0x 0000 005	PC + 4 + Lab6	0	1	0b10
0x 0000 007	PC + 4 + Lab8	1	1	0b11

0x 0000 000	PC + 4 + Lab1
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 007	PC + 4 + Lab8

1	1	0b00
1	0	0b01
0	1	0b10
1	1	0b11

2	0x 0000 0034	BEQ R7 R8 Lab4	Not Taken

0x 0000 000	PC + 4 + Lab1
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 007	PC + 4 + Lab8

1	1	0b00
0	1	0b01
0	1	0b10
1	1	0b11

0x 0000 000	PC + 4 + Lab1
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 007	PC + 4 + Lab8

1	1	0b00
0	1	0b01
0	1	0b10
1	1	0b11

3	0x 0000 006C	BEQ R13 R14 Lab7	Not Taken

0x 0000 000	PC + 4 + Lab1
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 006	PC + 4 + Lab7

1	1	0b00
0	1	0b01
0	1	0b10
0	1	0b11

0x 0000 000	PC + 4 + Lab1	1 1	0b00
0x 0000 003	PC + 4 + Lab4	0 1	0b01
0x 0000 005	PC + 4 + Lab6	0 1	0b10
0x 0000 006	PC + 4 + Lab7	0 1	0b11

4	0x 0000 0058	BEQ R11 R12 Lab6	Taken

0x 0000 000	PC + 4 + Lab1	1 1	0b00
0x 0000 003	PC + 4 + Lab4	0 1	0b01
0x 0000 005	PC + 4 + Lab6	0 0	0b10
0x 0000 006	PC + 4 + Lab7	0 1	0b11

0x 0000 000	PC + 4 + Lab1
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 006	PC + 4 + Lab7

1	1	0b00
0	1	0b01
0	0	0b10
0	1	0b11

5	0x 0000 0020	BNE R5 R6 Lab3	Taken

0x 0000 002	PC + 4 + Lab3
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 006	PC + 4 + Lab7

1	1	0b00
0	1	0b01
0	0	0b10
0	1	0b11

0x 0000 002	PC + 4 + Lab3
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 006	PC + 4 + Lab7

1	1	0b00
0	1	0b01
0	0	0b10
0	1	0b11

6	0x 0000 0034	BEQ R7 R8 Lab4	Taken

0x 0000 002	PC + 4 + Lab3
0x 0000 003	PC + 4 + Lab4
0x 0000 005	PC + 4 + Lab6
0x 0000 006	PC + 4 + Lab7

1	1	0b00
0	0	0b01
0	0	0b10
0	1	0b11

0x 0000 002	PC + 4 + Lab3	1 1	0b00
0x 0000 003	PC + 4 + Lab4	0 0	0b01
0x 0000 005	PC + 4 + Lab6	0 0	0b10
0x 0000 006	PC + 4 + Lab7	0 1	0b11

7	0x 0000 006C	BEQ R13 R14 Lab7	Not Taken

Q4 Answer:

Branch predictor state after 7 branches complete

0x 0000 002	PC + 4 + Lab3	1 1	0b00
0x 0000 003	PC + 4 + Lab4	0 0	0b01
0x 0000 005	PC + 4 + Lab6	0 0	0b10
0x 0000 006	PC + 4 + Lab7	0 1	0b11

On Tuesday

Mid-term I ...

T 3/18	Midterm I

Good luck !