CPSC 313: Computer Hardware and Operating Systems

Unit 1: The y86 (as a sequential processor)

Sequential Execution WrapUp

2024 Winter Term 1

Overview

- We are officially wrapping up Module 1. By the end, you should be able to:
  - Read and write simple y86 programs
  - Write y86 programs that implement simple C statements
  - Translate y86 assembly to/from its binary representation
  - Explain the behavior of y86 instructions in terms of how they might be implemented in hardware
  - Break apart y86 instruction execution into different phases
  - Use and explain the y86 calling conventions

Admin

- Lab 2 due Sunday
- Lab 3 released at 17:00 on Friday.
- Quiz 1 is this week
  - Quiz 1 “information” and “practice” on PrairieLearn
  - What’s the main focus? The unit we’re finishing today! ⏪

Admin

- Tutorial 2 is this week!
  - As always, open the tutorial placeholder on PrairieLearn and submit it so your TA can give you attendance credit. (You'll get 0 until your TA changes it.)
- You get a break from tutorials next week
  - Instead tutorial times will be treated as office hours
  - Lab 3 is challenging, so these will be great times to ask questions.

Understanding Execution

- First we understood each instruction in logical form – what it did.
- Then, we saw how to realize instructions in hardware, using wiring diagrams.
- Next, we examined execution as a phased process routing signals.
- Today: Execution as a set of formal specification statements (what must happen in each phase for each instruction).

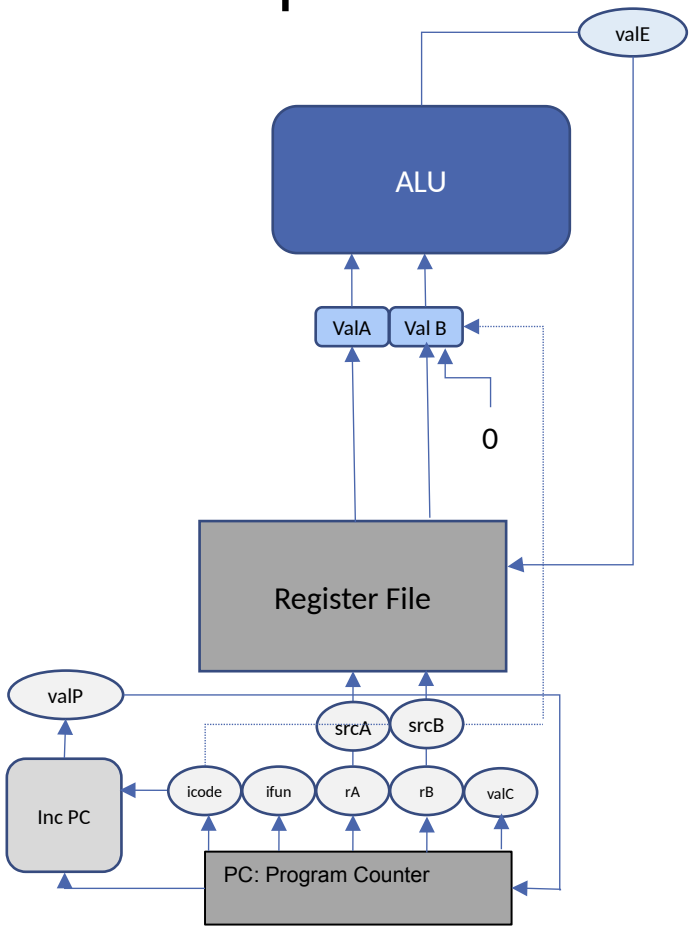
Why? Another useful perspective. Also…
…what if our conceptual phases are much more important than we yet realize?
More soon. .

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation

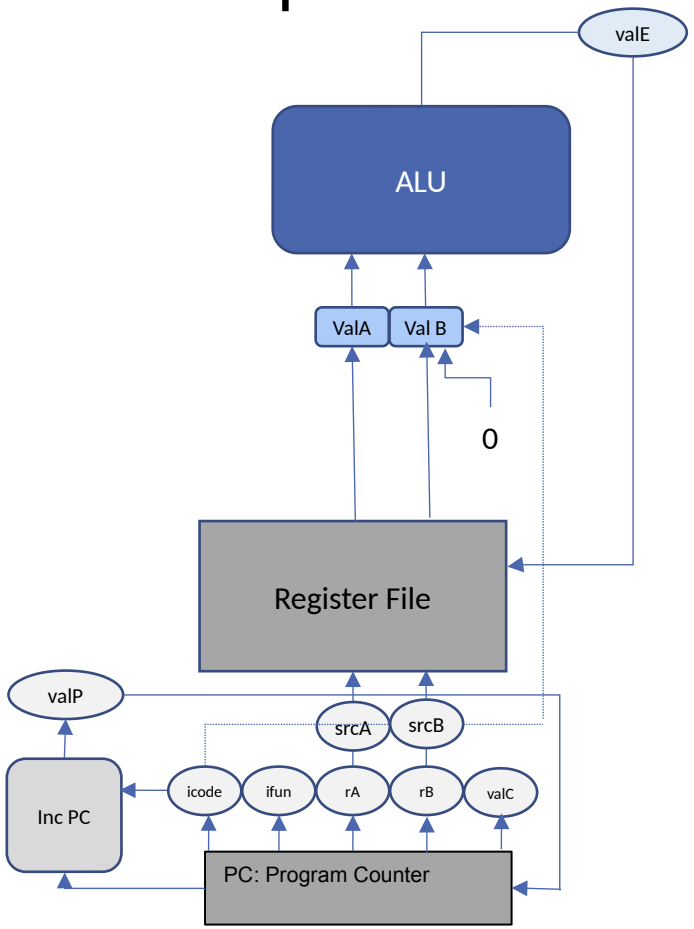


rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



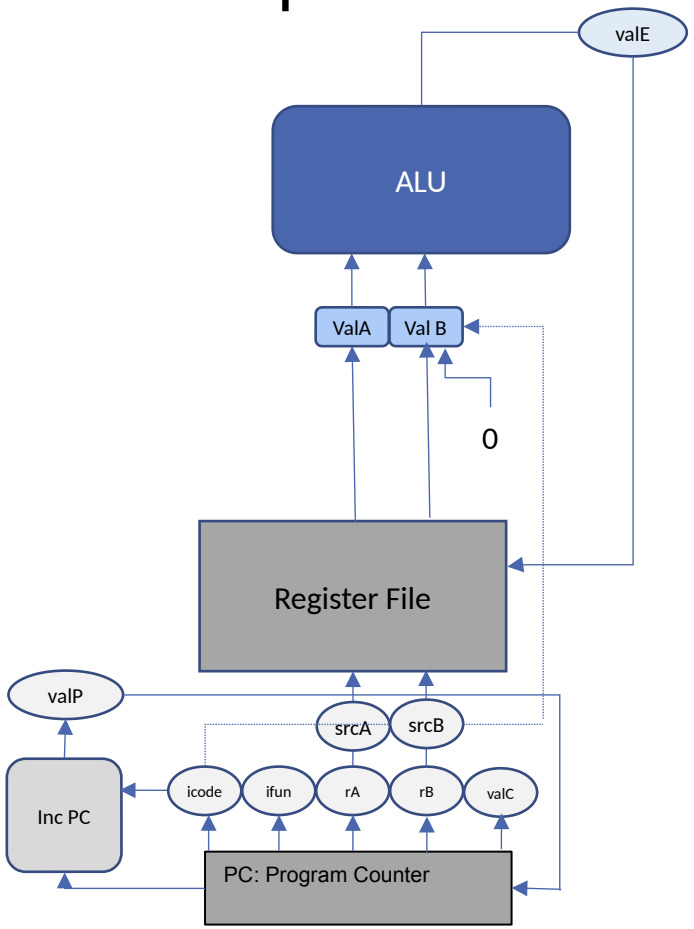
By Specification

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

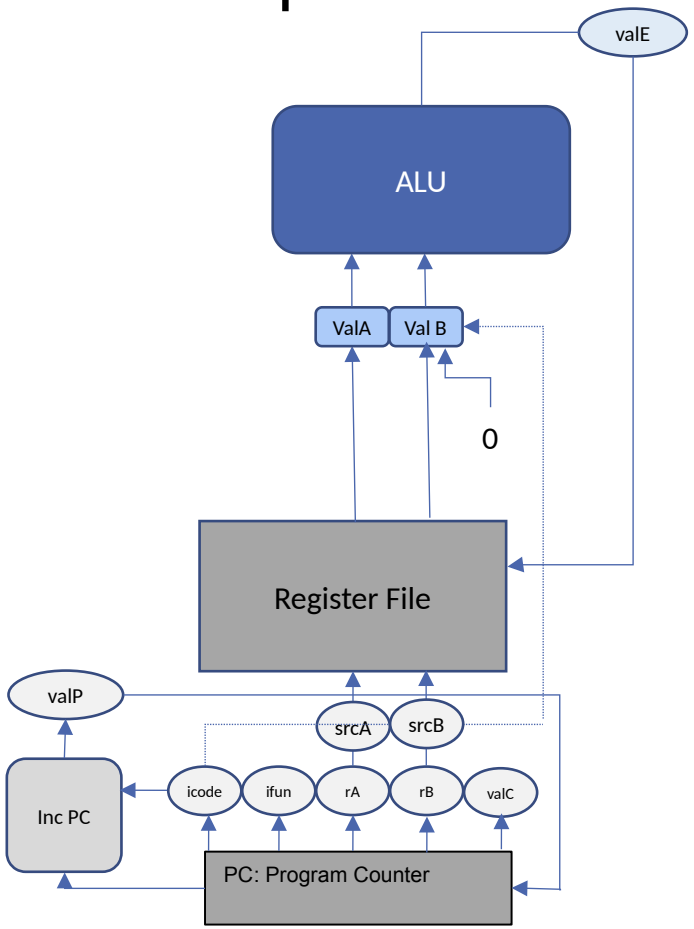
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

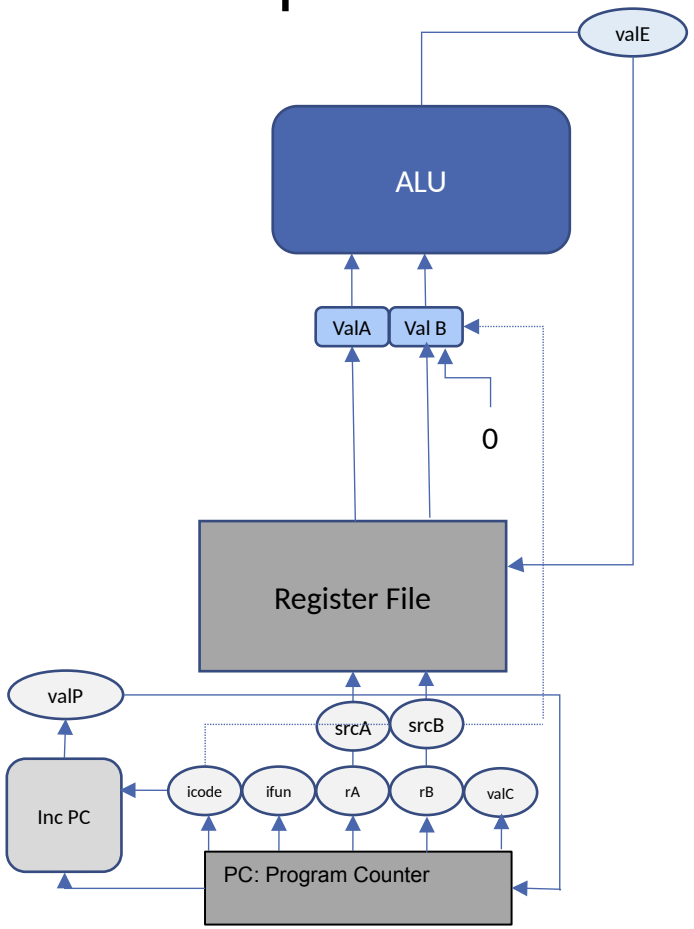
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

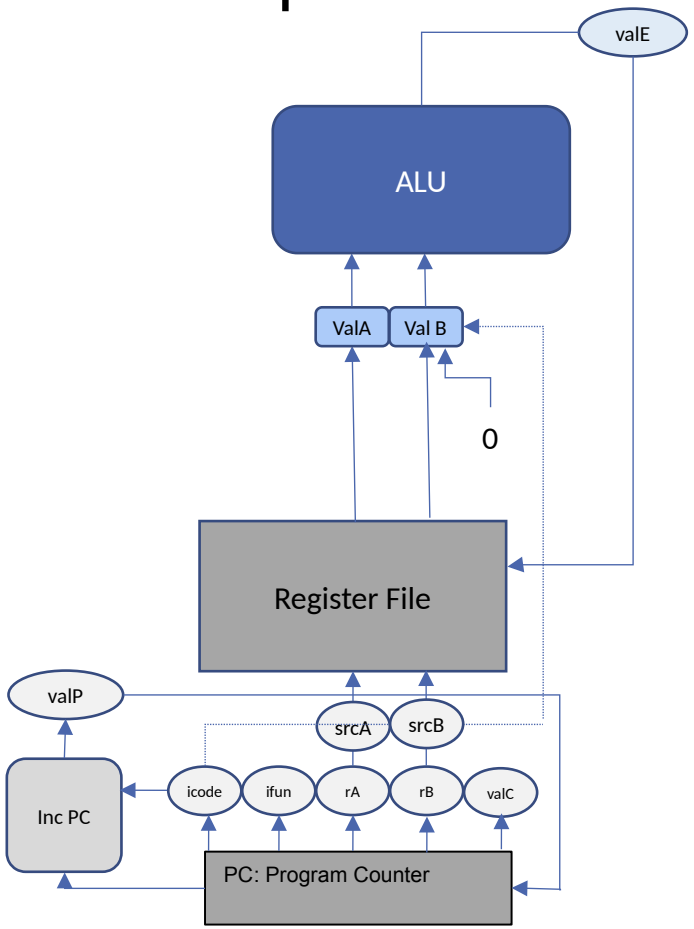
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

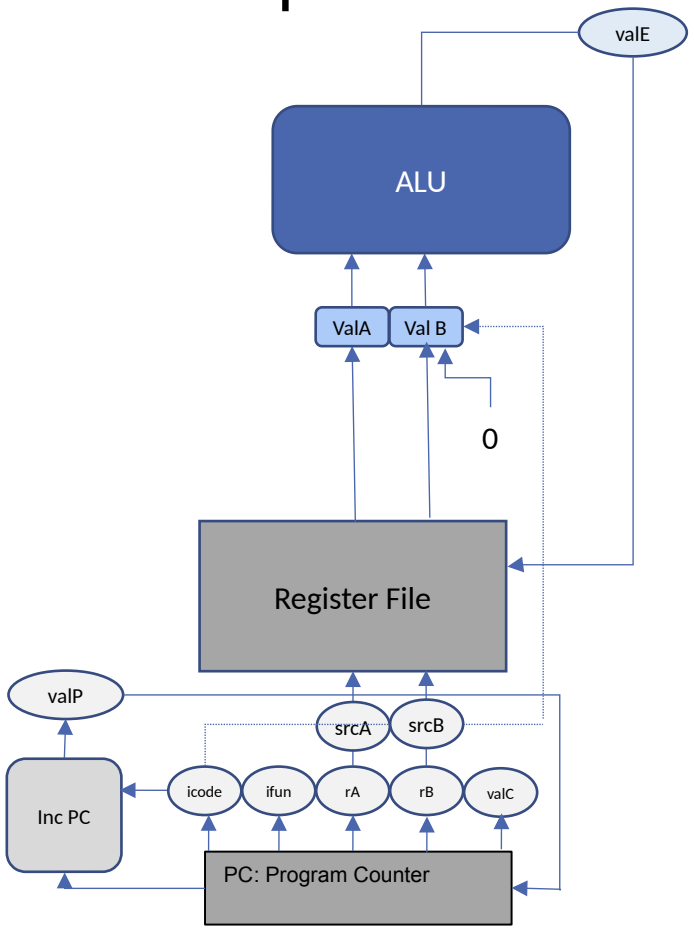
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

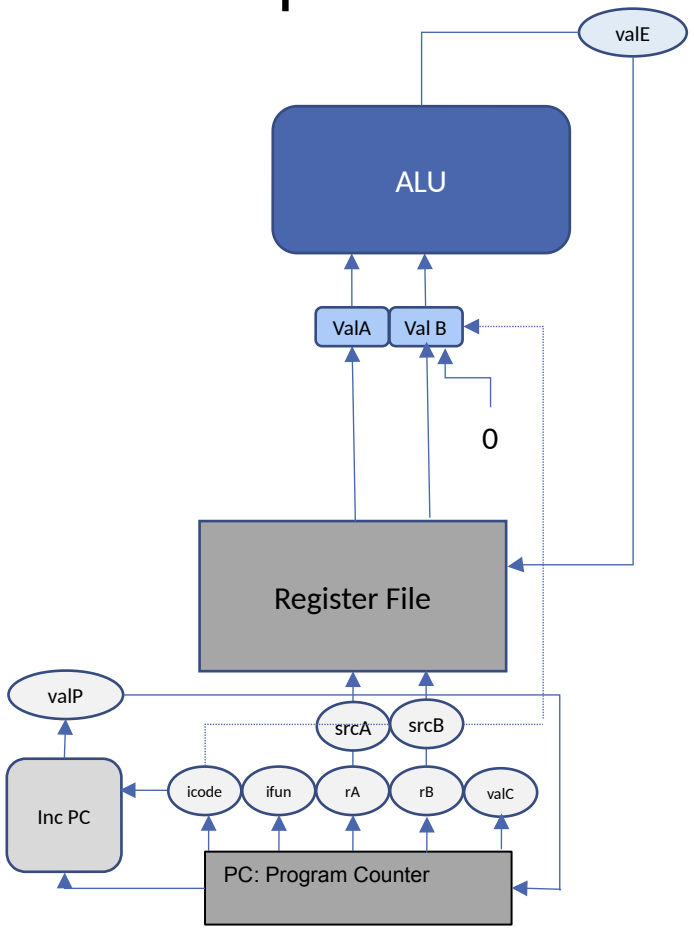
icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK

rrmovq three ways

From video 1

```
R[rB] <- R[rA]
```

From implementation



By Specification

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

From rrmovq to cmovxx

RRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

From rrmovq to cmovxx

RRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

CMOVXX

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

From rrmovq to cmovxx

RRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

CMOVXX

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0 Cnd = cond(CC, ifun)	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

From rrmovq to cmovxx

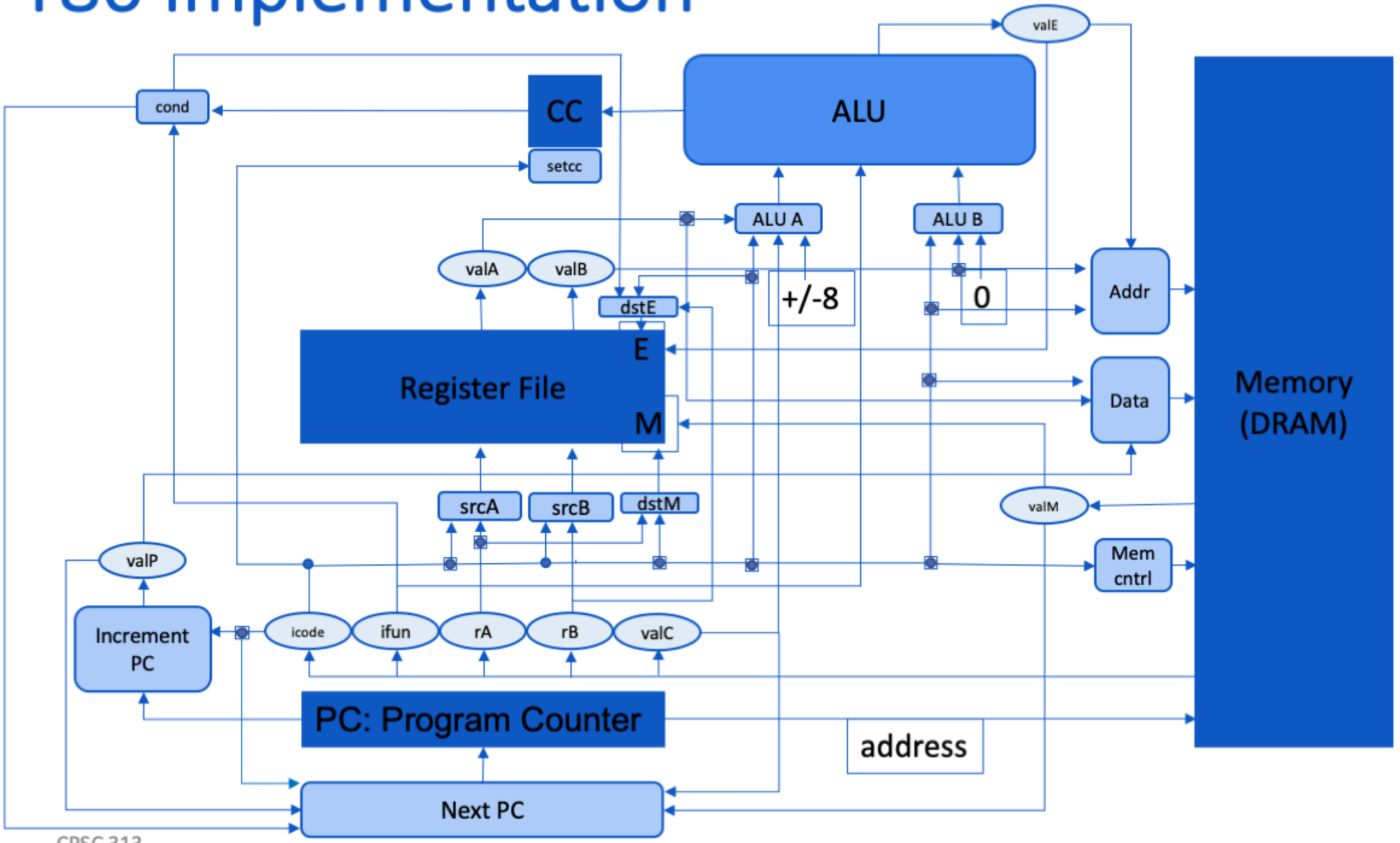
RRMOVQ

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0	EXECUTE
	MEMORY
R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

CMOVXX

icode:ifun = M_1[PC] rA:rB = M_1[PC+1] valP = PC + 2	FETCH
valA = R[rA]	DECODE
valE = valA + 0 Cnd = cond(CC, ifun)	EXECUTE
	MEMORY
if (Cnd) R[rB] = valE	WRITEBACK
PC = valP	PC UPDATE

Y86 Implementation



CMOVXX

icode:ifun = M_1[PC]

rA:rB = M_1[PC+1]

valP = PC + 2

valA = R[rA]

valE = valA + 0

Cnd = cond(CC, ifun)

If (Cnd) R[rB] = valE

PC = valP

Allowed in specifications:

- Fetch: icode, ifun, rA, rB, valC, valP, PC, Memory (for instruction fetch)
- Decode: rA, rB, valA, valB
- Execute: valC, valE, valA, valB, CC, Cnd, cond, 0, 8, -8, ifun
  - **CC** are the condition codes
  - **cond** is logic that combines the condition codes and the ifun to produce a 0/1 **Cnd** signal that is used to enable/disable other operations.
- Memory: valA, valB, valE, valM, valP, Memory
- Writeback: Cnd, rA, rB, valE, valM
- PC Update: PC, Cnd, valC, valM, valP

Anywhere you use rA/rB, you might also use %rsp explicitly.
Feel free to assume a function that helps with performing the appropriate ALU operation and computing CC values.

Let’s try it!

- In each group - one of you should:
  - Sign into your google account (if nobody has one, we’ll figure it out!)
  - Make a copy of the spreadsheet in google
  - Share the sheet with everyone in your group
- Collaboratively discuss what happens at each phase and record it in the spreadsheet
- Each of you is to save and submit the PDF version of the spreadsheet
- We will release a completed copy of the spreadsheet before class Friday.

Next…

- Friday
  - No new material because of the quiz; I will be in class to answer questions.
- Monday
  - Pipelining (module 2)