CPSC 313: Computer Hardware and Operating Systems

Unit 1: The y86 (as a sequential processor)

Become a CPU designer!

2024 Winter Term 1

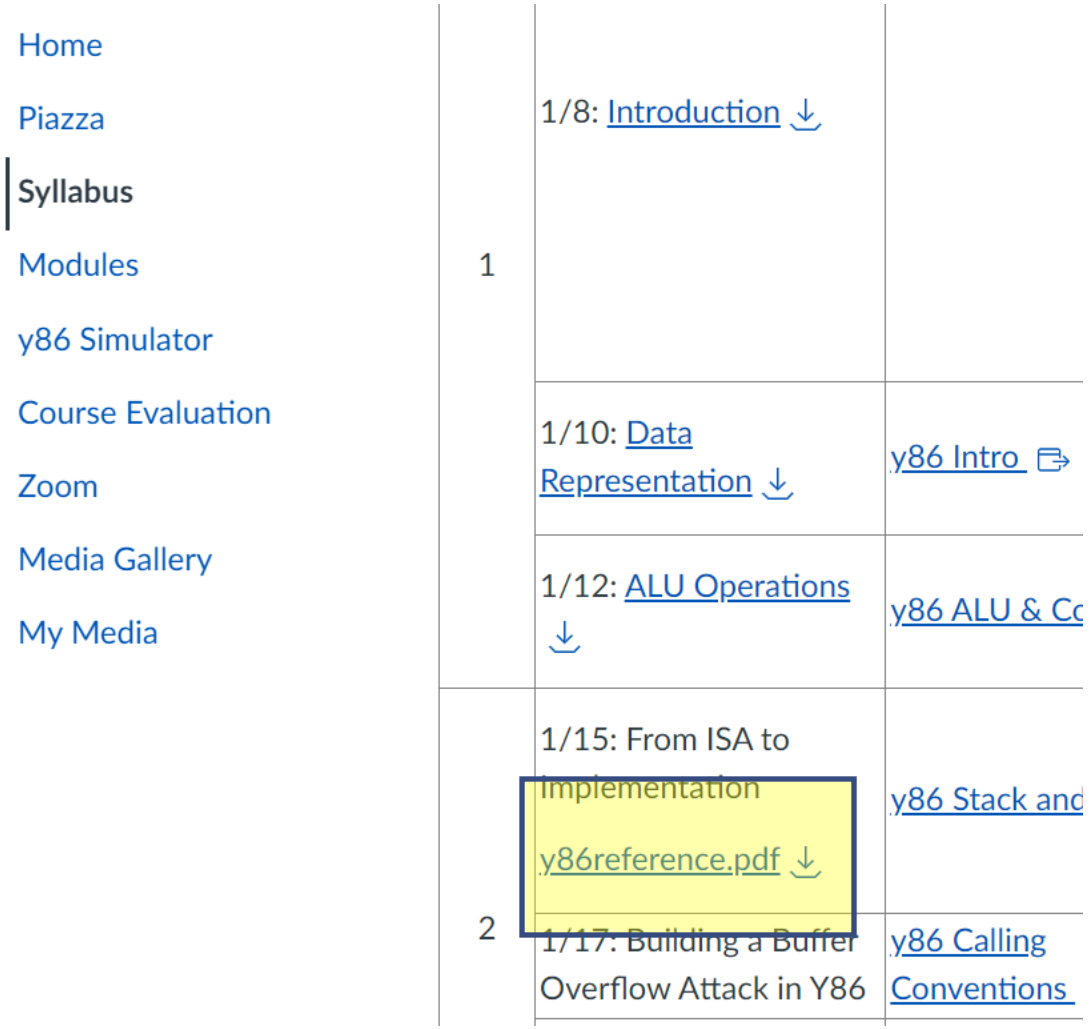
Admin

- Lab 1 due Sunday; Quiz 0 due Sep 18
- More pre- and in-class exercises coming (as always!)
- Be sure to **read all "pinned posts" on Piazza** for key course info!

Today

- Develop some intuition as to how an instruction gets executed
  - Another of the *many* tools we use to think about the meaning and implementation of an instruction.
- Prepare yourself to think about implementing y86 as a CPU

Y86 Instruction Cheat Sheet



Home
Piazza
Syllabus
Modules
y86 Simulator
Course Evaluation
Zoom
Media Gallery
My Media

1	1/8: Introduction	
	1/10: Data Representation	y86 Intro
	1/12: ALU Operations	y86 ALU & Cc
2	1/15: From ISA to Implementation y86reference.pdf	y86 Stack and
	1/17: Building a Buffer Overflow Attack in Y86	y86 Calling Conventions

Y86 Instruction Cheat Sheet

Y86-64 Instructions Encoding

Byte	0		1		2	3	4	5	6	7	8	9
halt	0	0										
nop	1	0										
rrmovq **rA**, **rB**	2	0	**rA**	**rB**								
cmovXX **rA**, **rB**	2	fn	**rA**	**rB**								
irmovq **V**, **rB**	3	0	F	**rB**	**V**							
rmmovq **rA**, **D**(**rB**)	4	0	**rA**	**rB**	**D**							
mrmovq **D**(**rB**), **rA**	5	0	**rA**	**rB**	**D**							
OPq **rA**, **rB**	6	fn	**rA**	**rB**								
jXX **Dest**	7	fn	**Dest**									
call **Dest**	8	0	**Dest**									
ret	9	0										
pushq **rA**	A	0	**rA**	F								
popq **rA**	B	0	**rA**	F								

Instruction	Semantics	Example
rrmovq %rs, %rd	r[rd] ← r[rs]	rrmovq %rax, %rbx
cmovXX %rs, %rd	r[rd] ← r[rs] if last ALU XX 0 (XX is le/l/e/ne/ge/g)	cmovle %rax, %rbx
irmovq $i, %rd	r[rd] ← i	irmovq $100, %rax
rmmovq %rs, D(%rd)	m[D + r[rd]] ← r[rs]	rmmovq %rax, 100(%rbx)
mrmovq D(%rs), %rd	r[rd] ← m[D + r[rs]]	mrmovq 100(%rbx), %rax
OPq %rs, %rd	r[rd] ← r[rd] OP r[rs]	addq %rax, %rbx
jmp D	goto D	jmp foo
jXX D	goto D if last ALU result XX 0 (XX is le/l/e/ne/ge/g)	jle foo
call D	pushq PC; jmp D	call foo
ret	popq PC	ret
pushq %rs	m[r[rsp] - 8] ← r[rs]; r[rsp] = r[rsp] - 8	pushq %rax
popq %rd	r[rd] ← m[r[rsp]]; r[rsp] = r[rsp] + 8	popq %rax

Hexadecimal conversions

Hex	Bin	Hex	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

ifun values

ifun	OPq	jXX/cmovXX
0	add	no condition
1	sub	le
2	and	l
3	xor	e
4	mul	ne
5	div	ge
6	mod	g

Register Names

#	Name	#	Name
0	%rax	8	%r8
1	%rcx	9	%r9
2	%rdx	A	%r10
3	%rbx	B	%r11
4	%rsp	C	%r12
5	%rbp	D	%r13
6	%rsi	E	%r14
7	%rdi	F	NONE

Function argument registers: %rdi %rsi %rdx %rcx %r8 %r9 Function return register: %rax

Recall: The y86 in a single slide

Registers (RF: Register File)

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

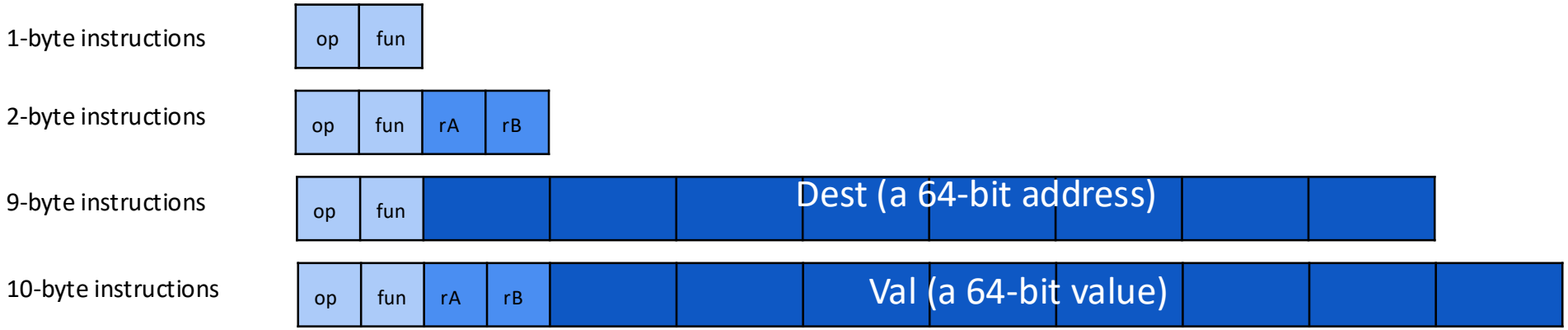
Condition Codes (CC)

ZF	SF	OF

Stat: Status Register

PC: Program Counter

DMEM: Memory



Recall: The y86 in a single slide

Registers (RF: Register File)

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

Condition Codes (CC)

ZF	SF	OF

Stat: **AOK**

PC: Program Counter

DMEM: Memory

Y86 Implementation: Steps in Execution

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

ZF	SF	OF

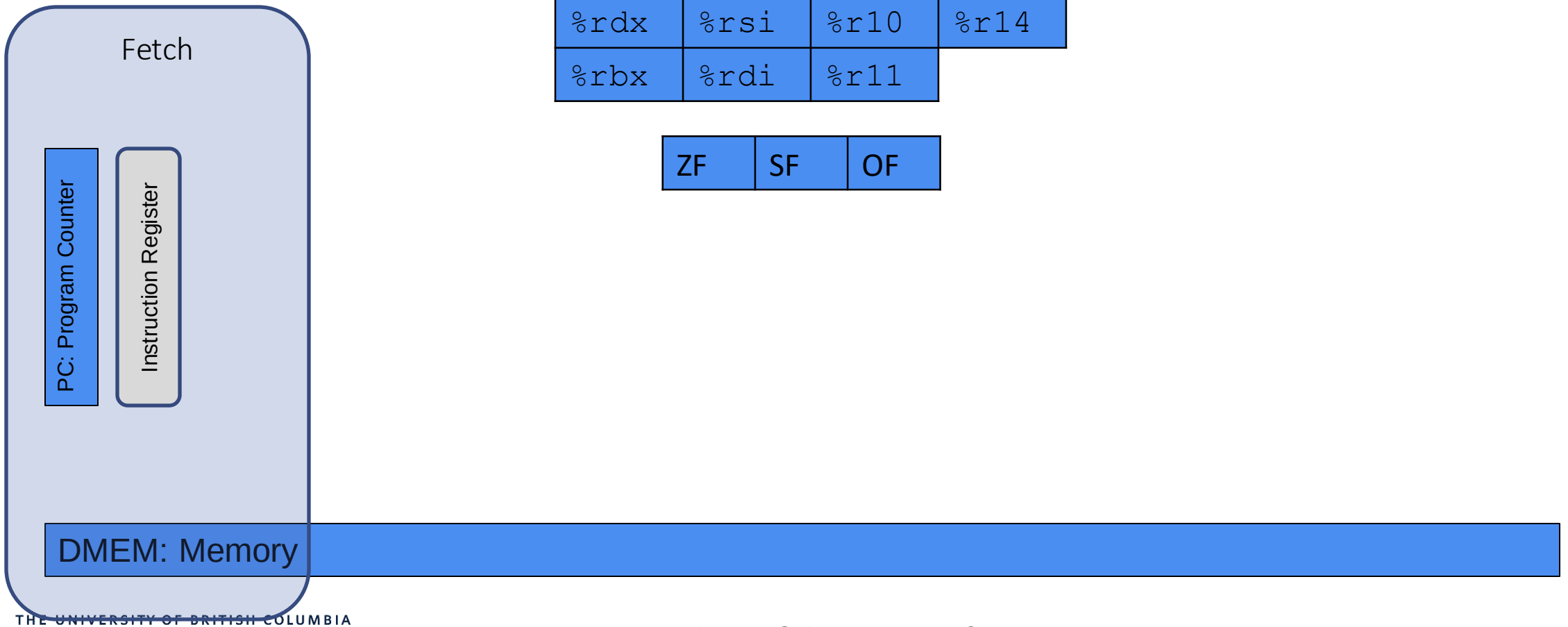
PC: Program Counter

DMEM: Memory

Y86 Implementation: Steps in Execution

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

ZF	SF	OF

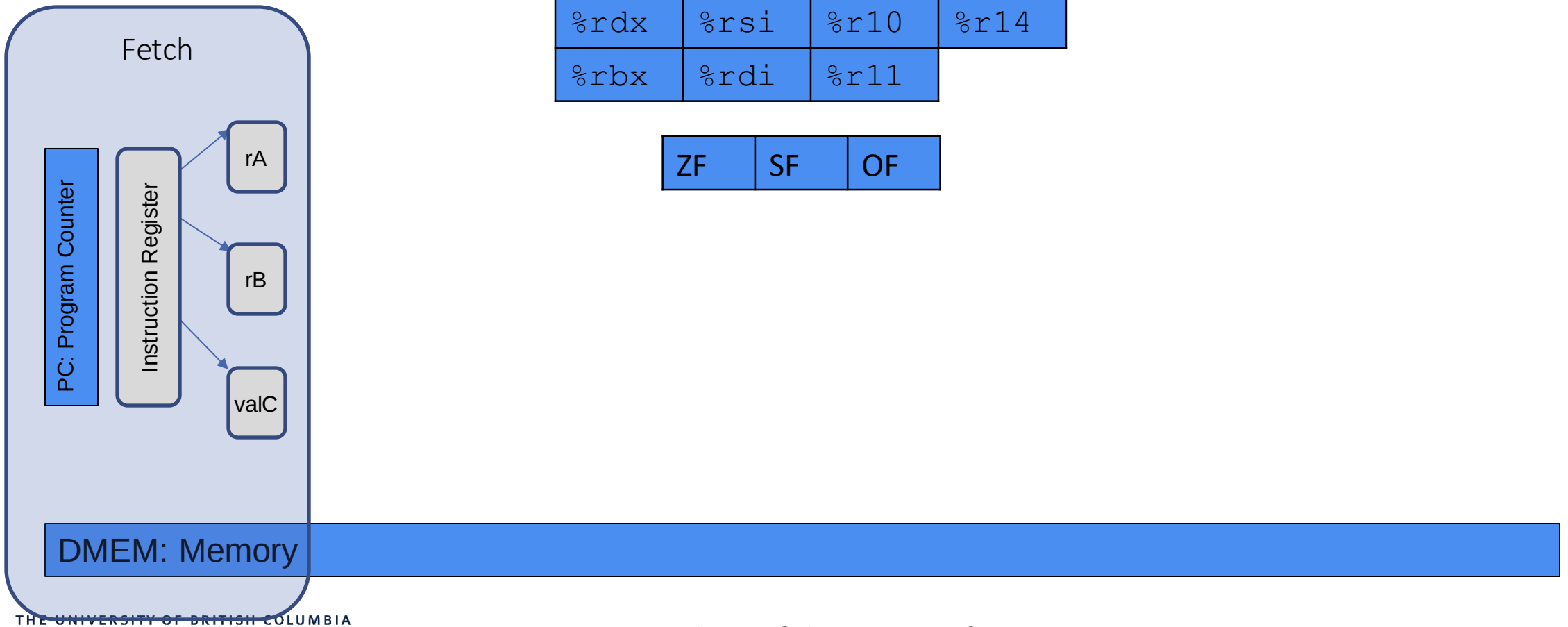



```
mrmovq   4(%r8), %r9
```

Y86 Implementation: Steps in Execution

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

ZF	SF	OF

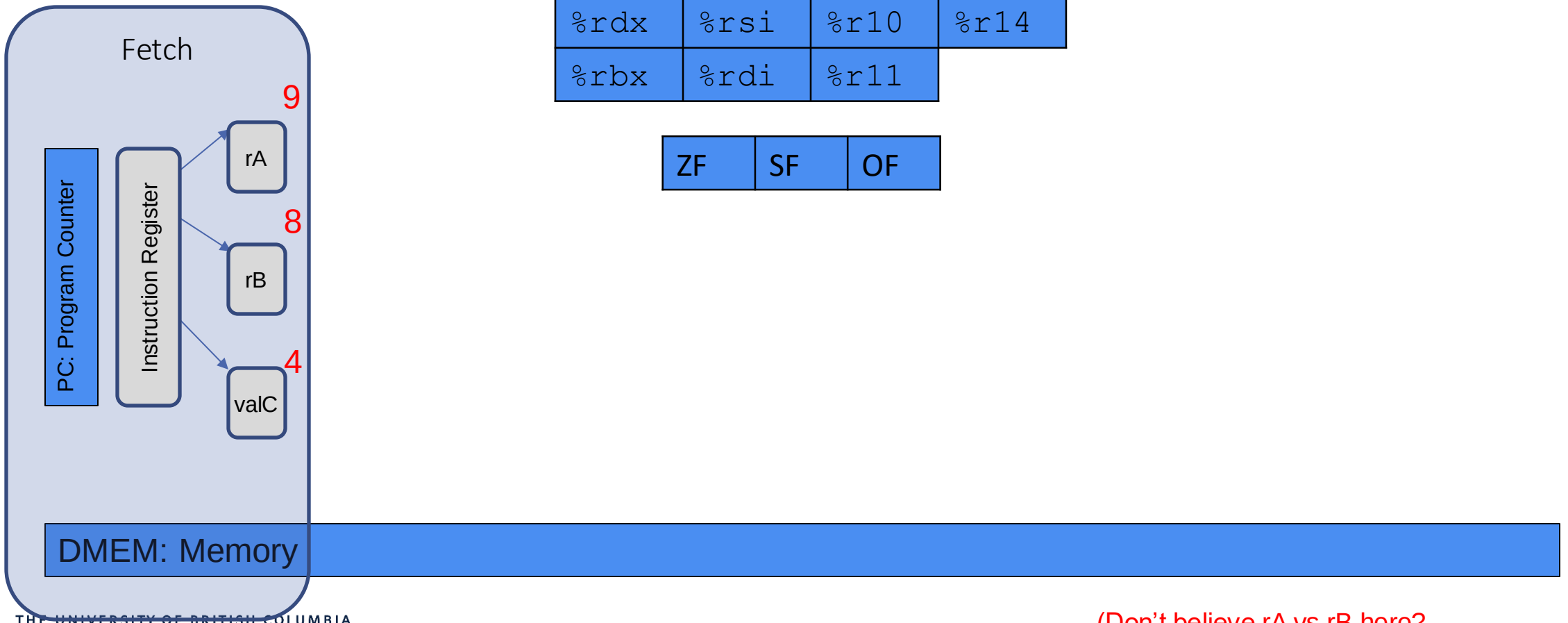


mrmovq 4(%r8), %r9

Y86 Implementation: Steps in Execution

%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

ZF	SF	OF



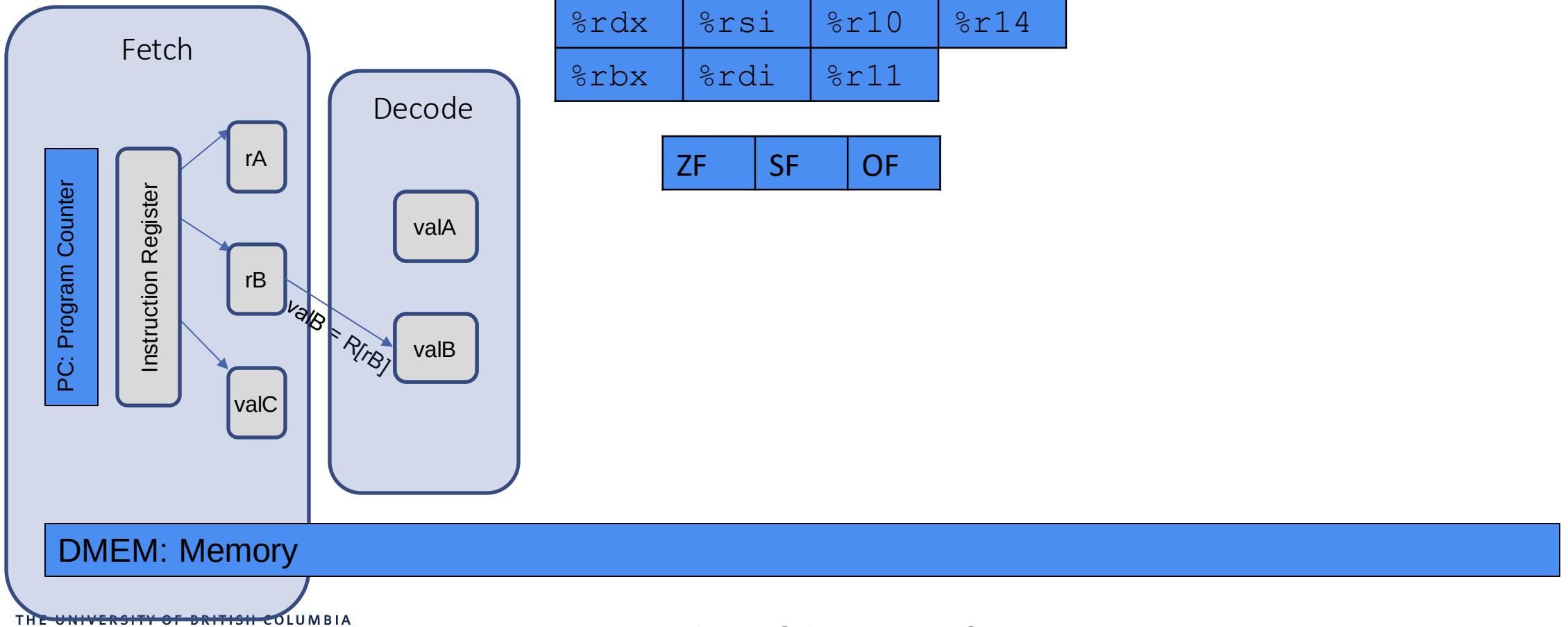
mrmovq 4(%r8), %r9

(Don't believe rA vs rB here? Check the ref sheet!)

Y86 Implementation: Steps in Execution

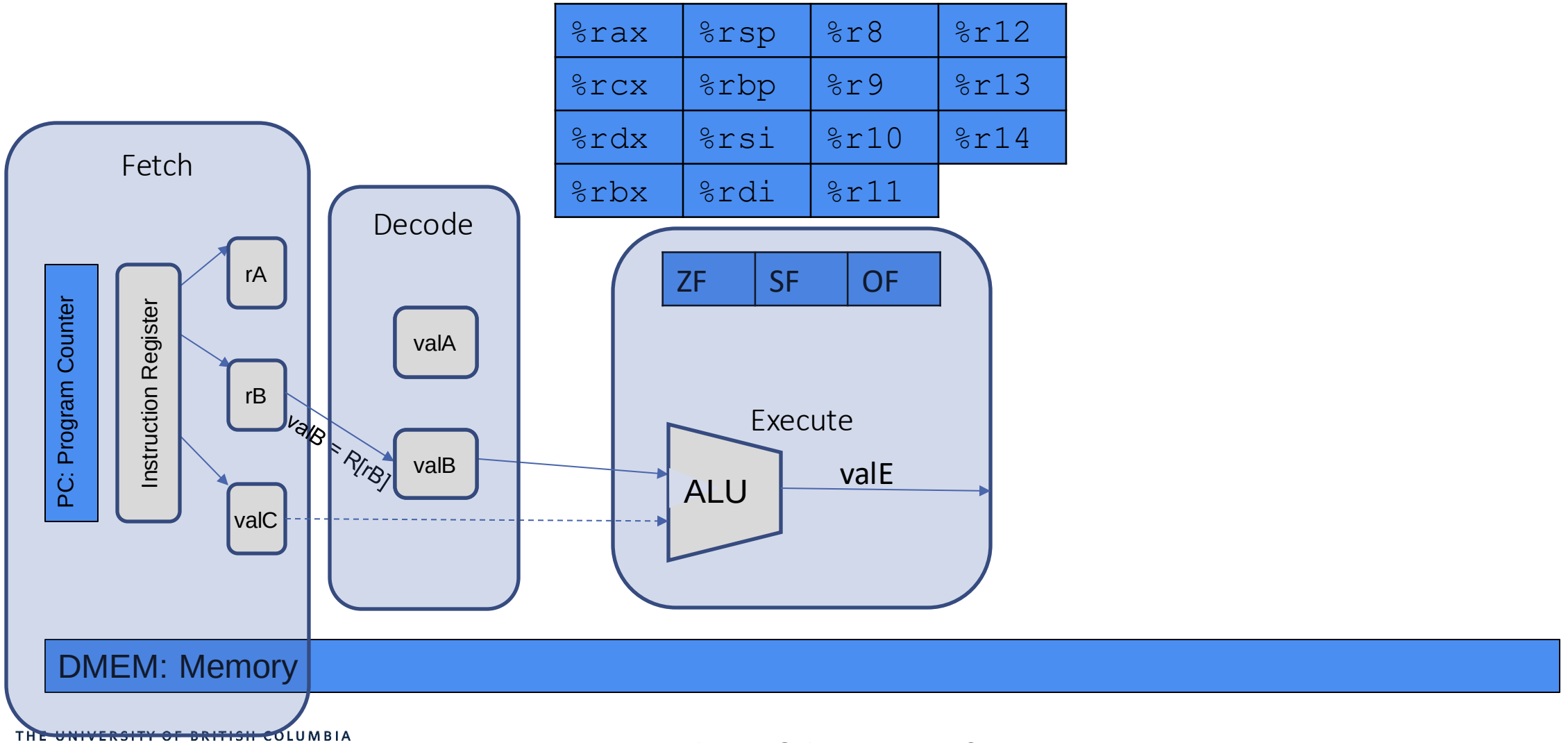
%rax	%rsp	%r8	%r12
%rcx	%rbp	%r9	%r13
%rdx	%rsi	%r10	%r14
%rbx	%rdi	%r11	

ZF	SF	OF



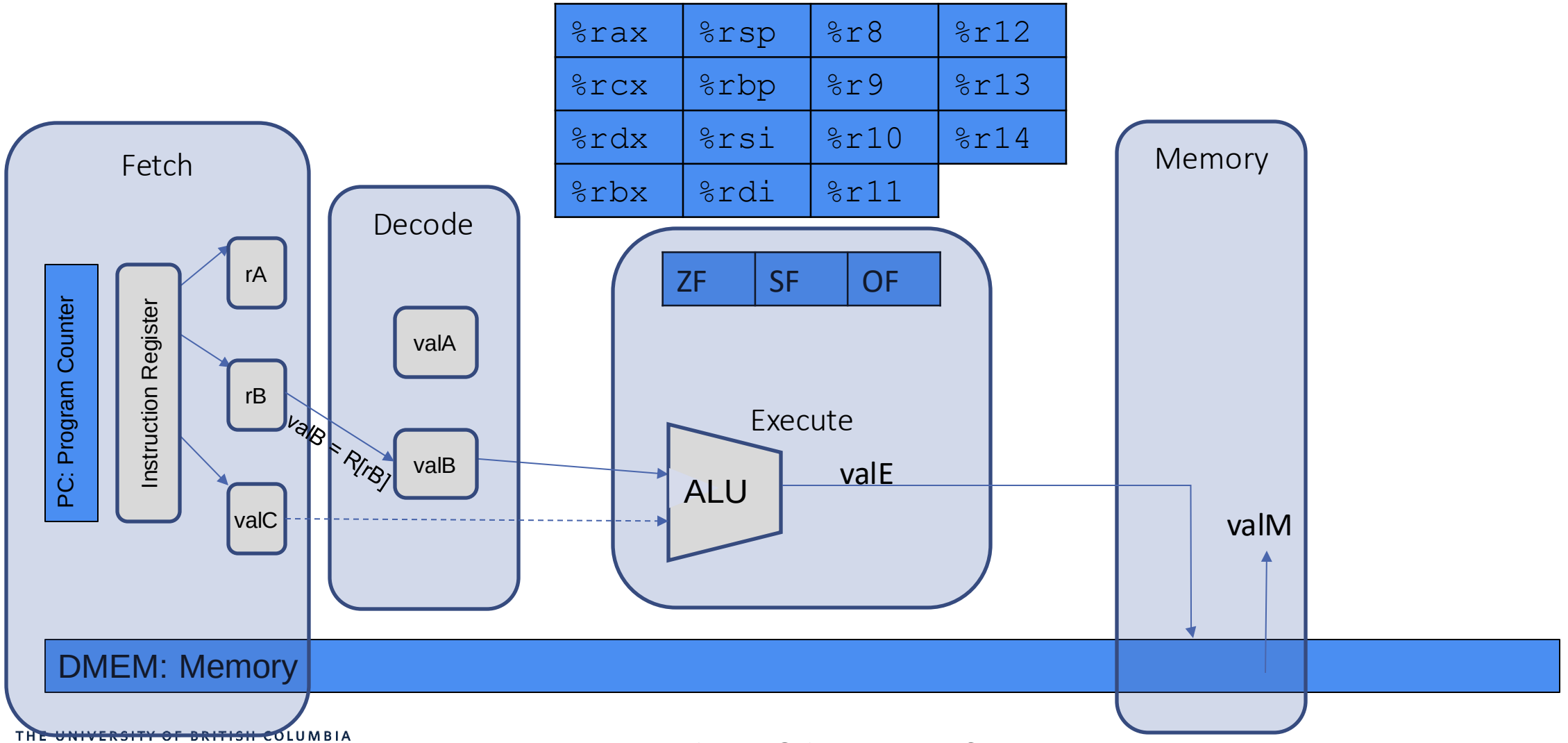
mrmovq 4(%r8), %r9

Y86 Implementation: Steps in Execution



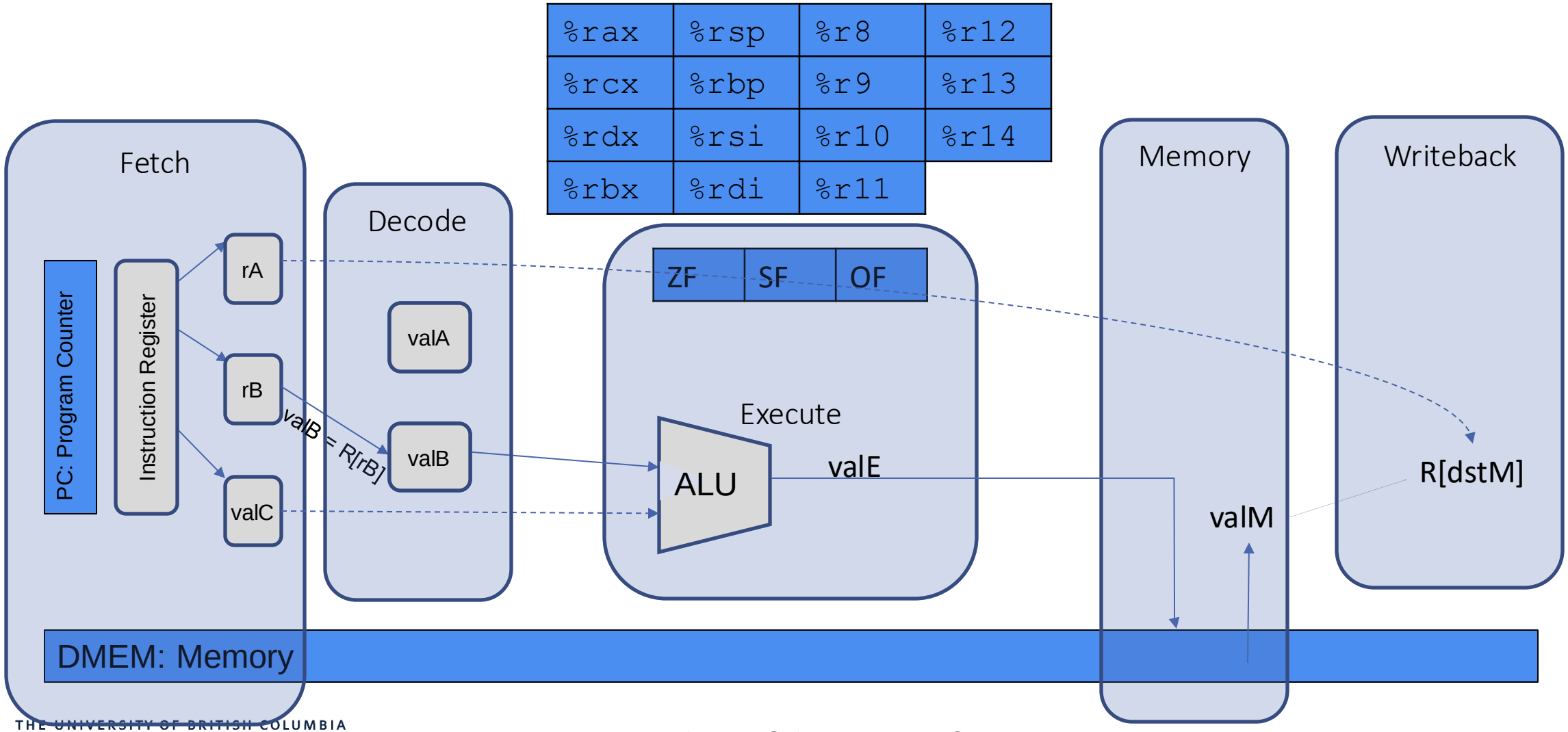

```
mrmovq   4(%r8), %r9
```

Y86 Implementation: Steps in Execution



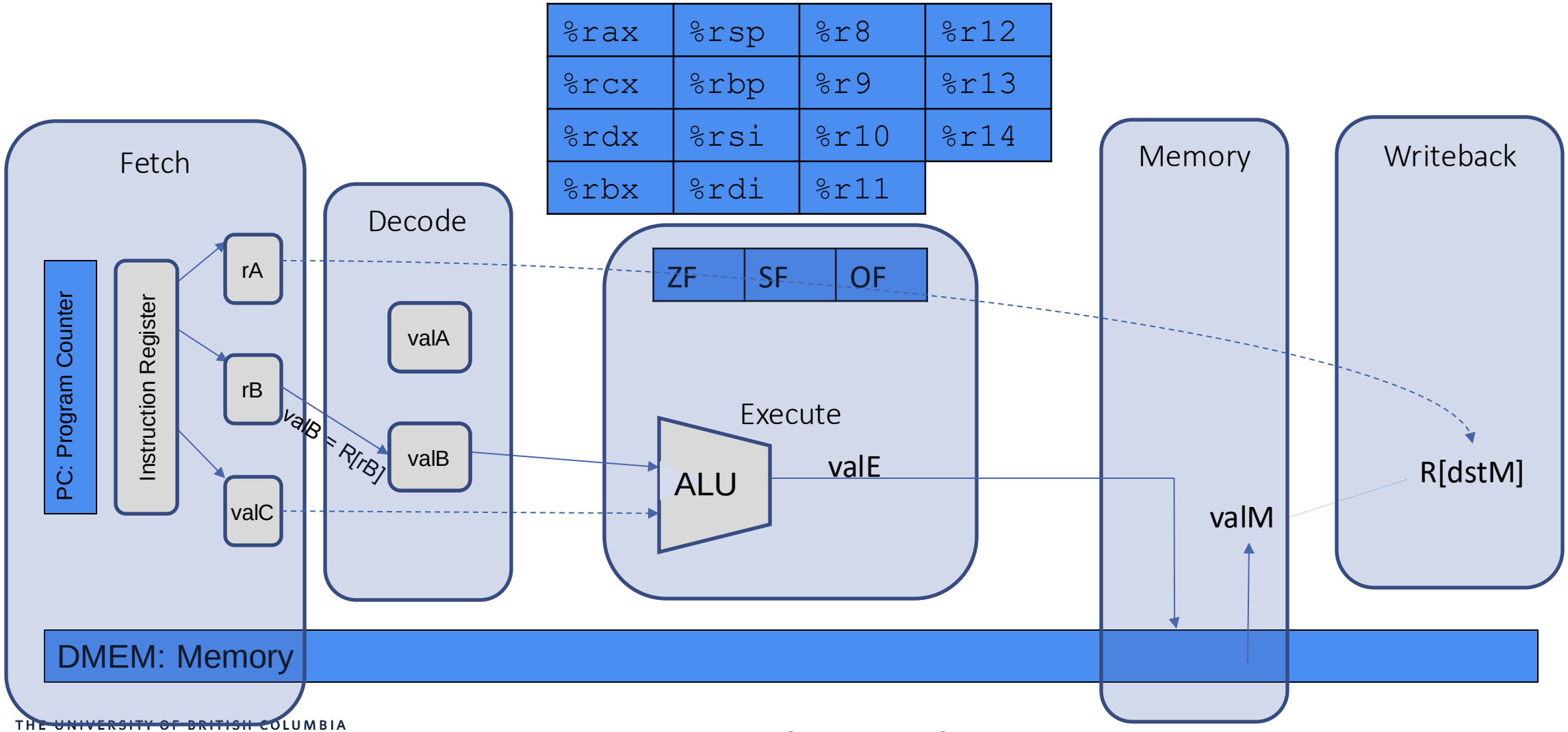

```
mrmovq   4(%r8), %r9
```

Y86 Implementation: Steps in Execution



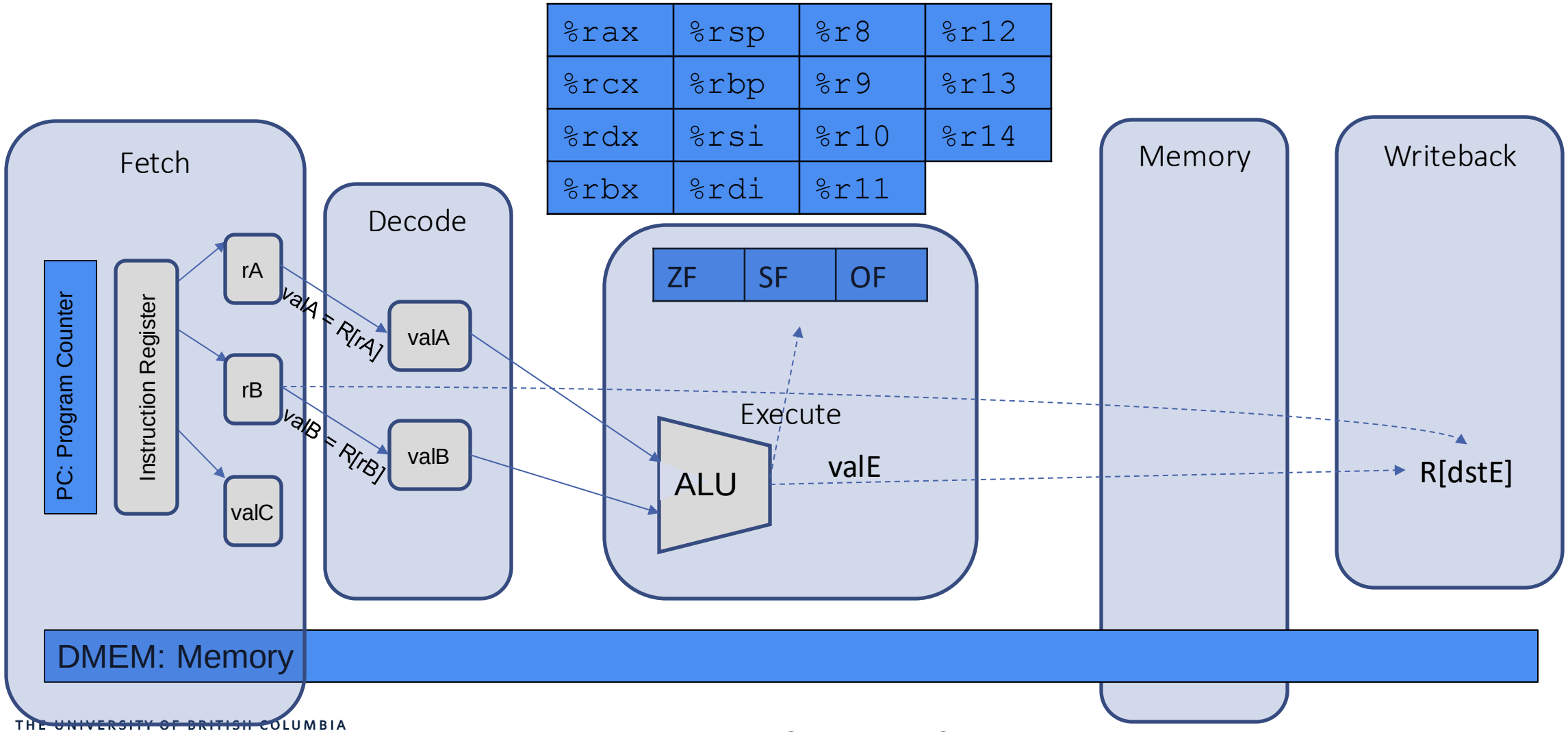

```
mrmovq   4(%r8), %r9
```

What do we change from mrmovq to addq?




```
addq  %r8, %r9
```

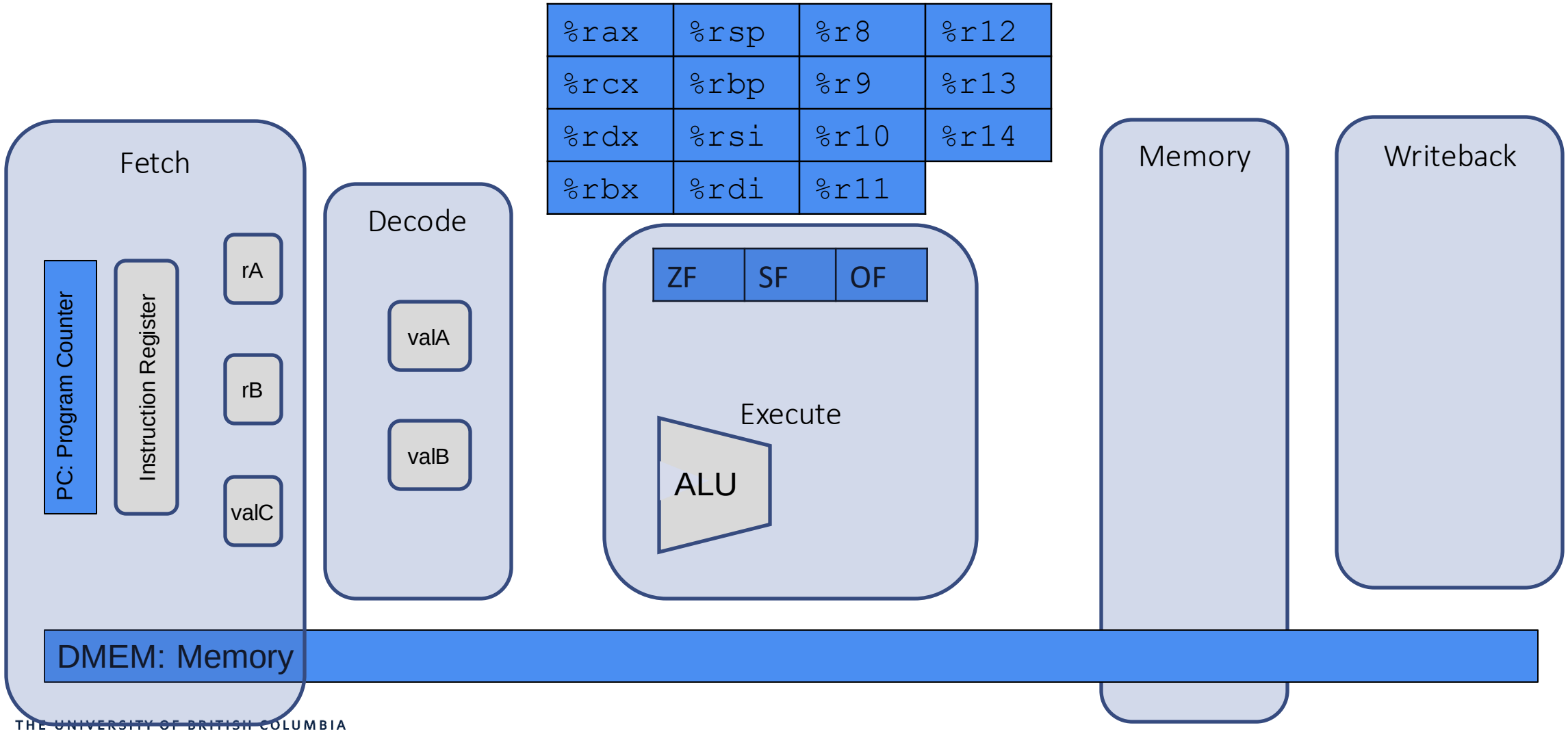
Y86 Implementation: addq



addq %r8, %r9

Inclass Exercise

If there's time, try another instruction!



Wrapping up

- The implementation of the y86 can be viewed in five steps
- Don’t forget Quiz 0 (due tonight!)
- Next time: Fun exercise to practice your y86 expertise.
  - Even more than always: Do the pre-class exercise!
  - Bring a laptop and some scratch paper!

- In the class after that, we’ll return to today’s ideas and..

  try to translate them into an implementation!

If there’s time, we’ll quickly review call/ret together.
If not, you should review the video and the next two slides for next class!

First, let's *quickly* review CALL..

CALL:

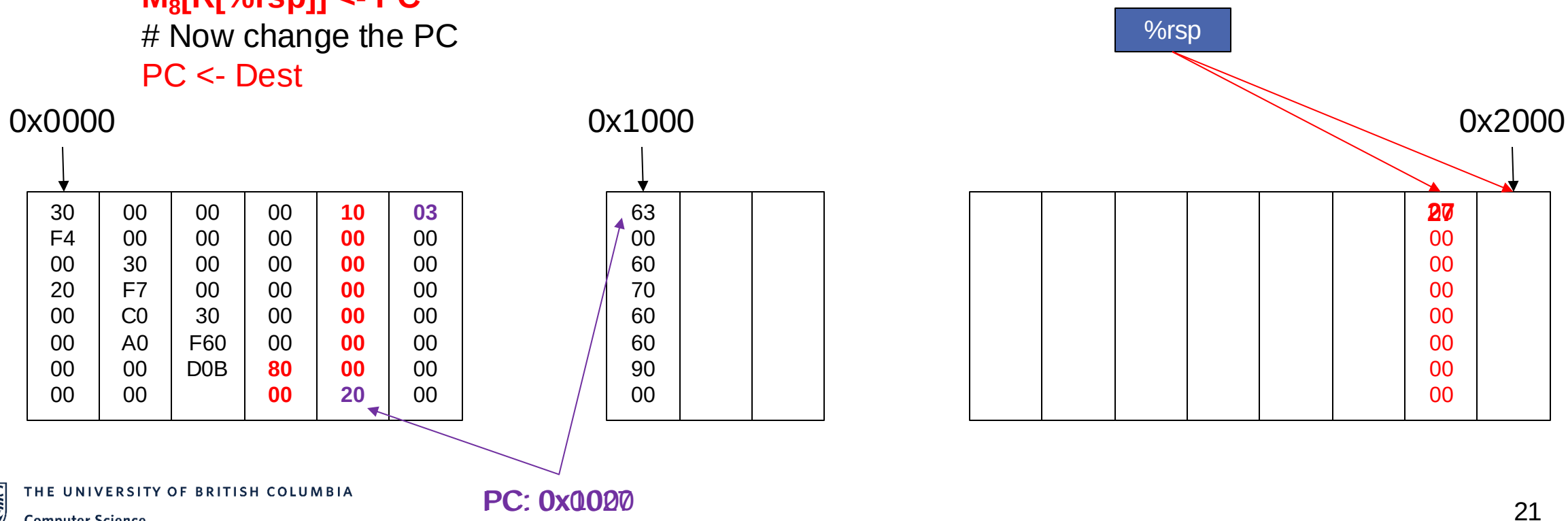
This part is conceptually PUSHQ PC

R[%rsp] <- R[%rsp] – 8

M_8[R[%rsp]] <- PC

Now change the PC

PC <- Dest



First, let's *quickly* review CALL and RET

RET:

Conceptually POPQ PC

PC <- M_8[R[%rsp]]

R[%rsp] <- R[%rsp] + 8