CS 3330: SEQ part 1

13 September 2016

Changelog

Corrections made in this version not in first posting:

16 Sep 2016: Slide 26: Added missing execute stage.

State in Y86-64

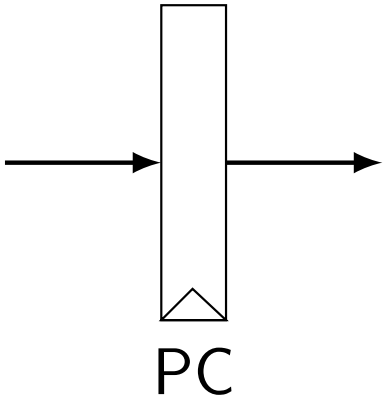
program counter (register)

register file (15 registers: `%rax`, `%rdx`, ...)

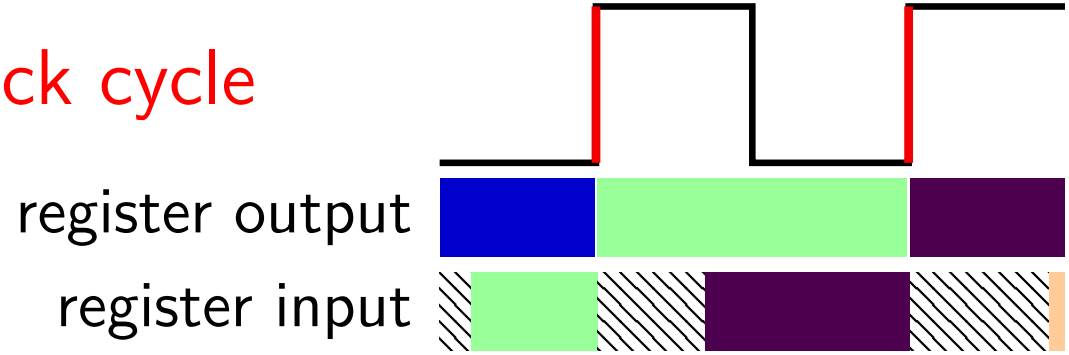
condition codes (ZF, SF)

status register (is the processor still running?)

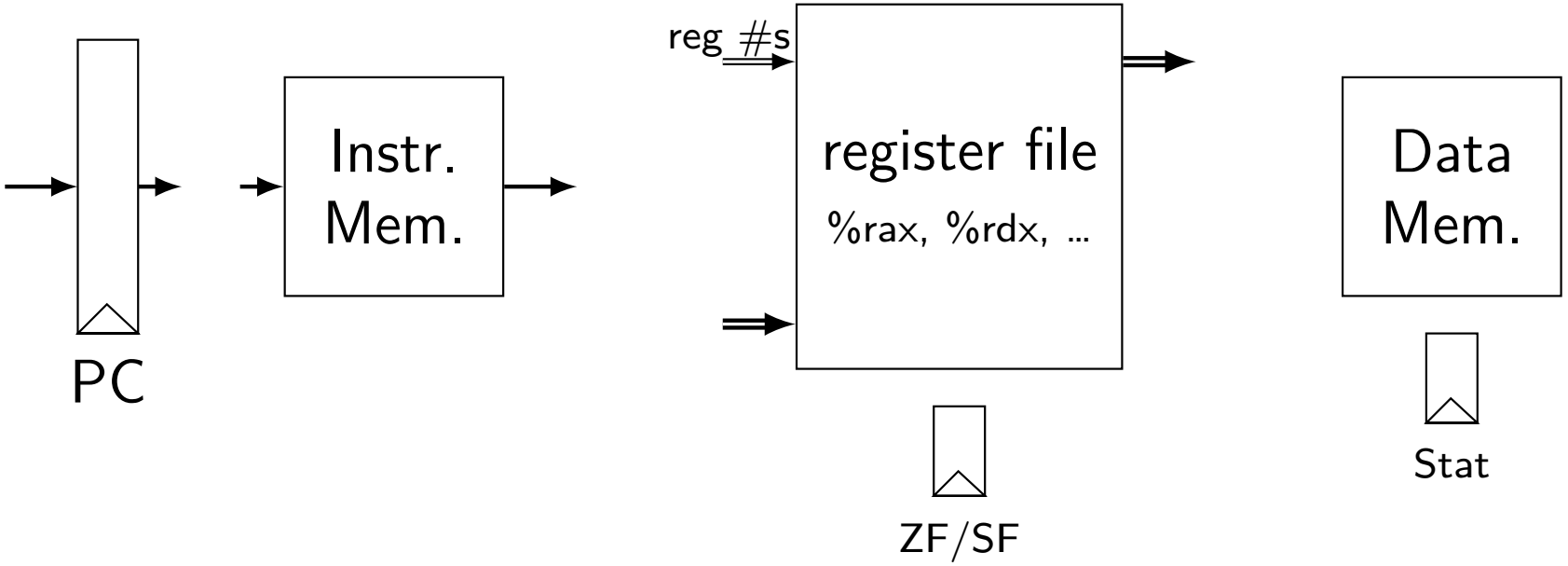
Registers



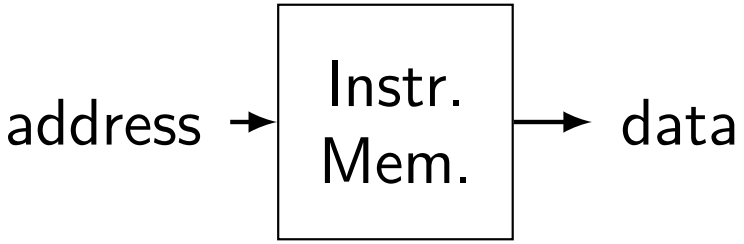
updates every clock cycle

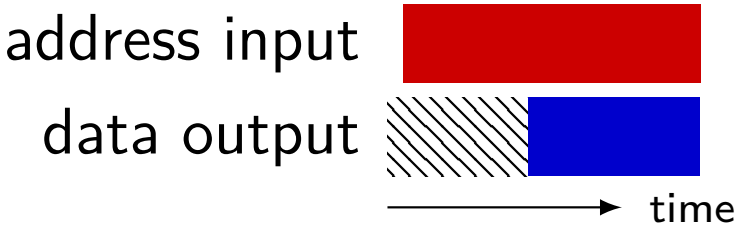


State in Y86-64

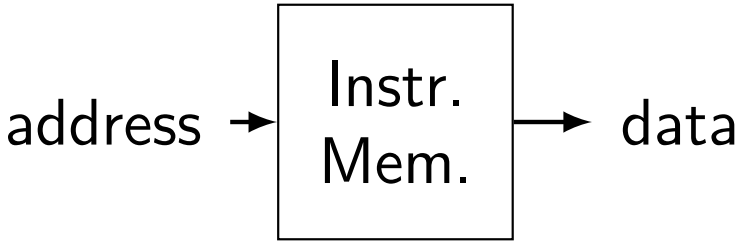


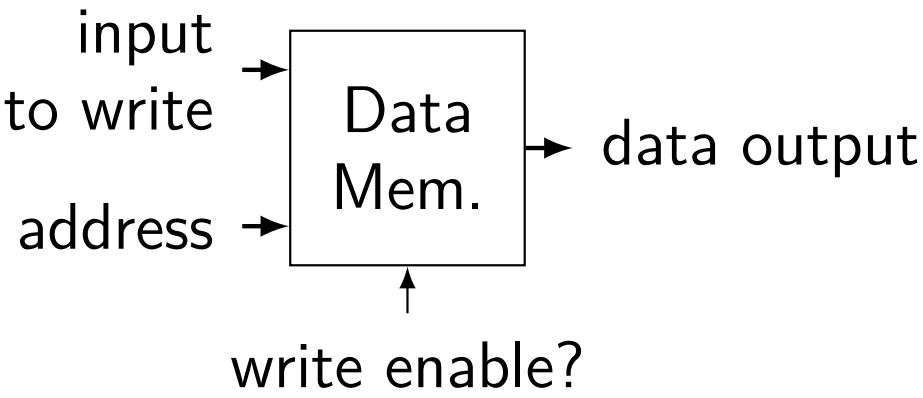
Memories

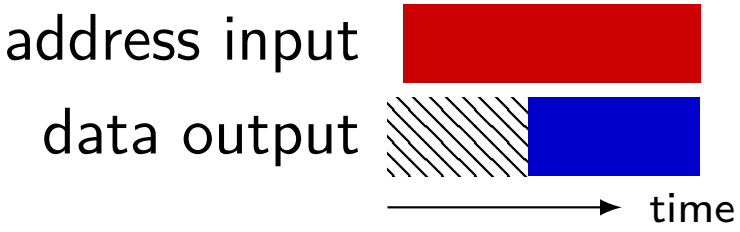




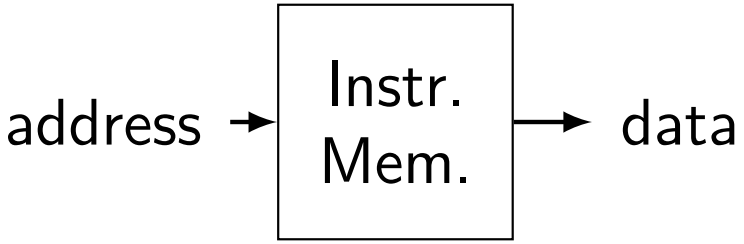
Memories

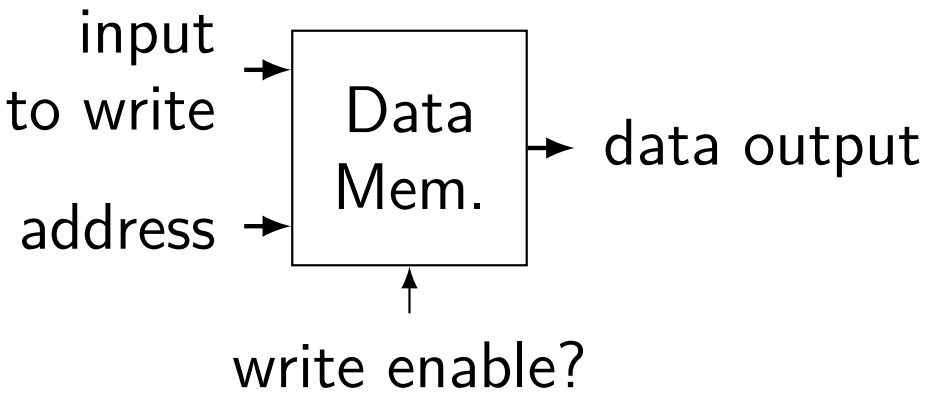

address
Instr.
Mem.
data

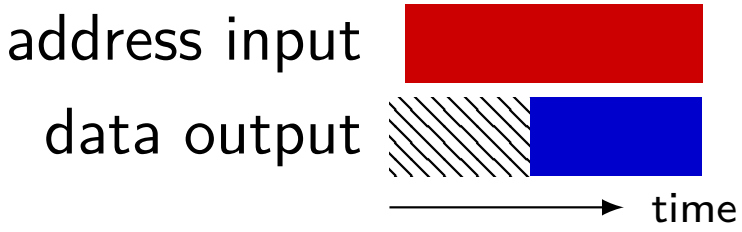

input
to write
address
Data
Mem.
data output
write enable?

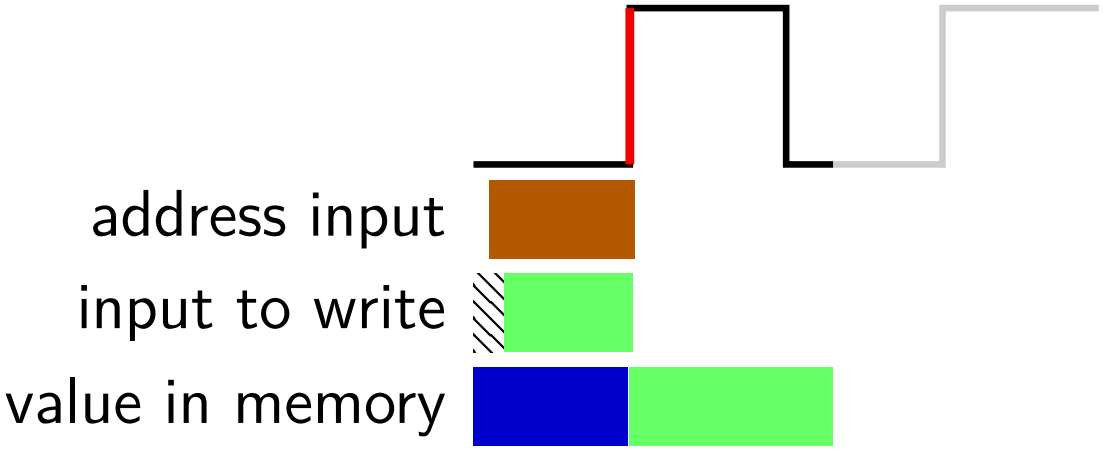

address input
data output
time

Memories

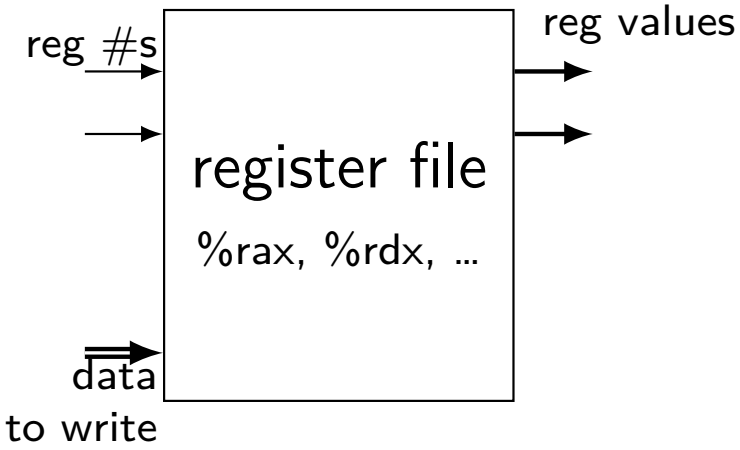

address
Instr.
Mem.
data


input
to write
address
Data
Mem.
data output
write enable?

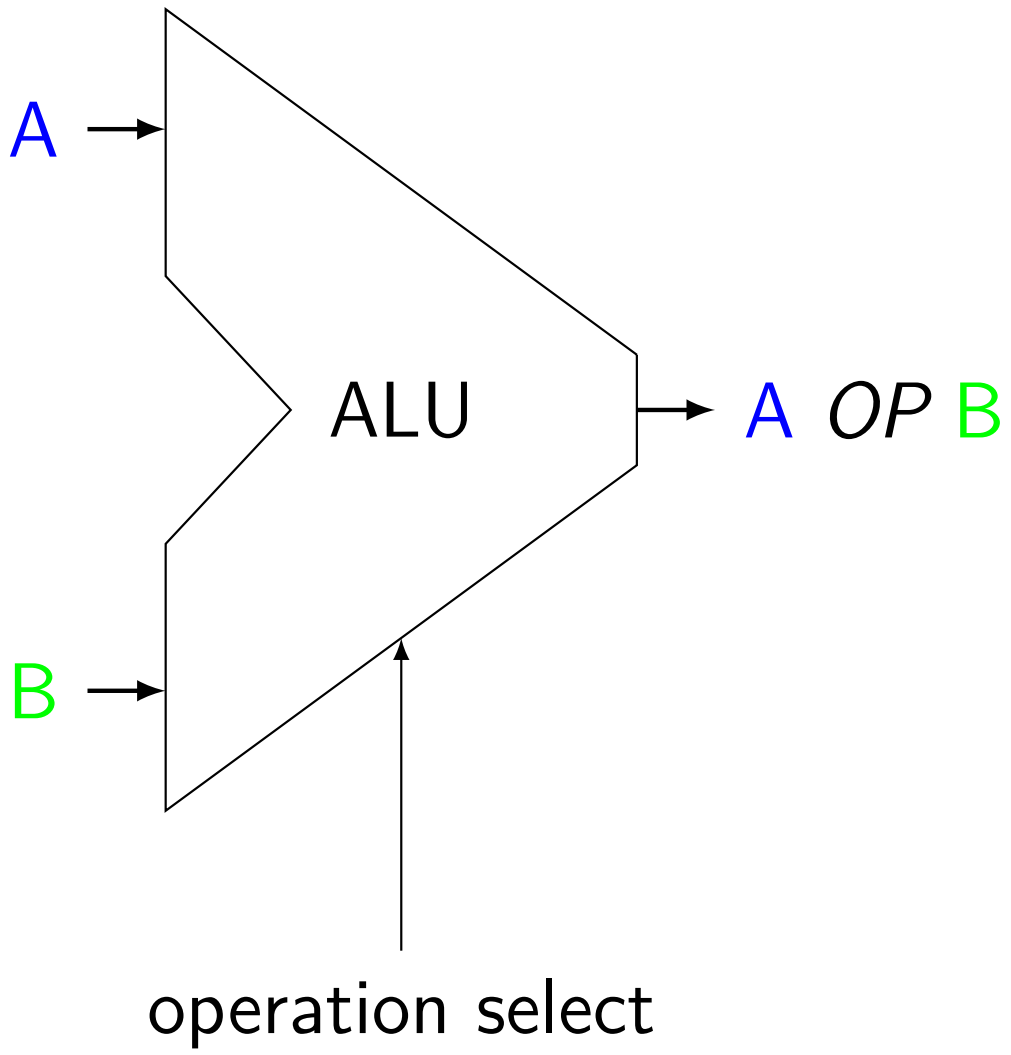

address input
data output
time


address input
input to write
value in memory

Register file



ALUs



Operations needed:
add — **`addq`**, addresses
sub — **`subq`**
xor — **`xorq`**
and — **`andq`**
more?

Simple ISA 1: addq

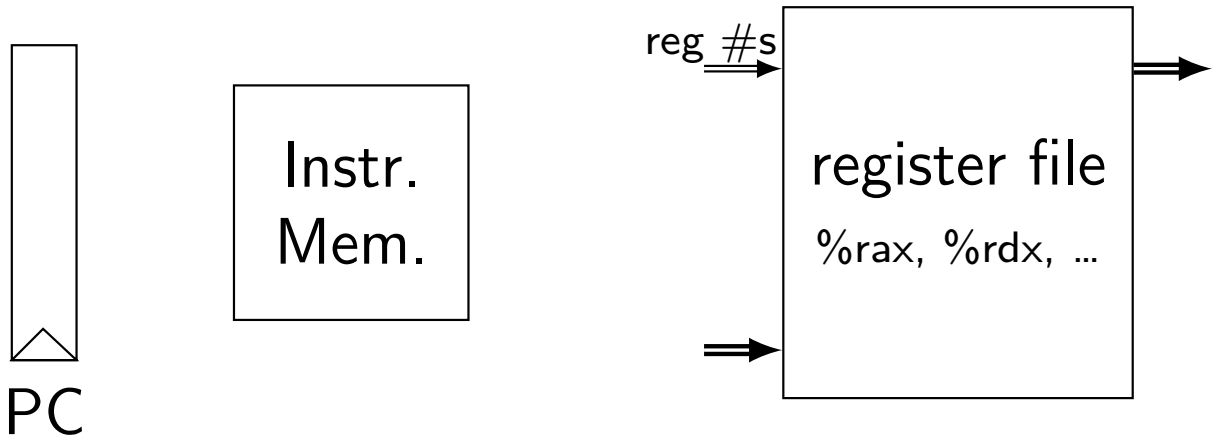
`addq %rXX, %rYY`

encoding: *4-bit register #*, *4-bit register #*

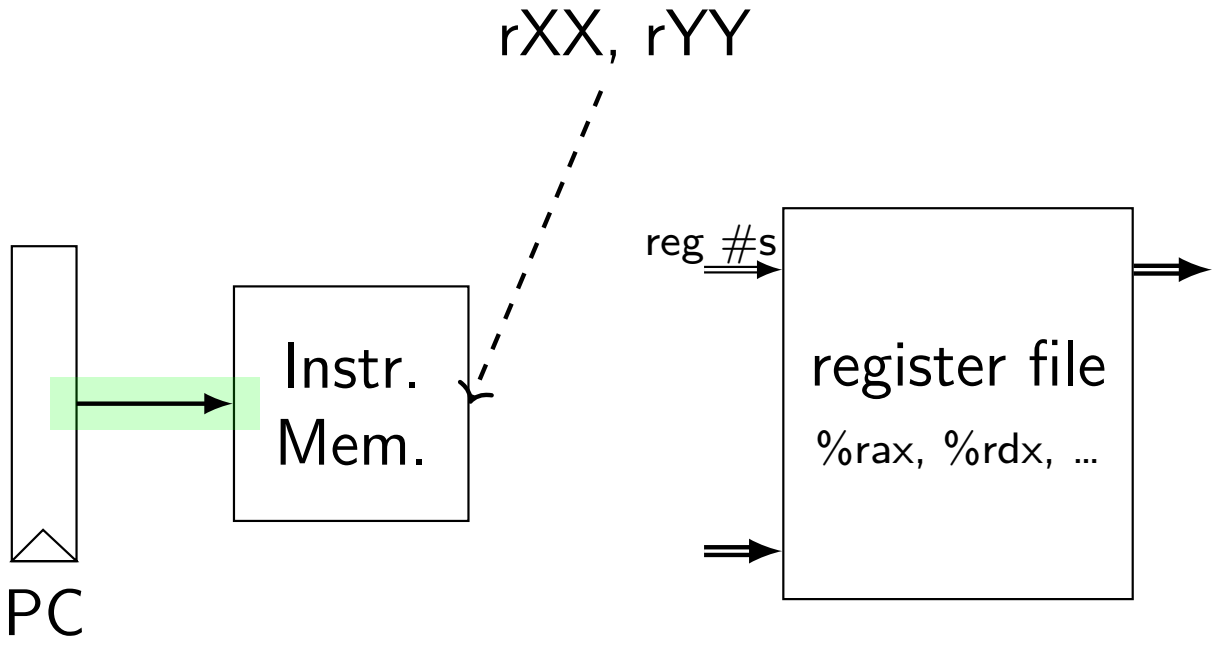
1 byte instructions, no opcode

no other instructions

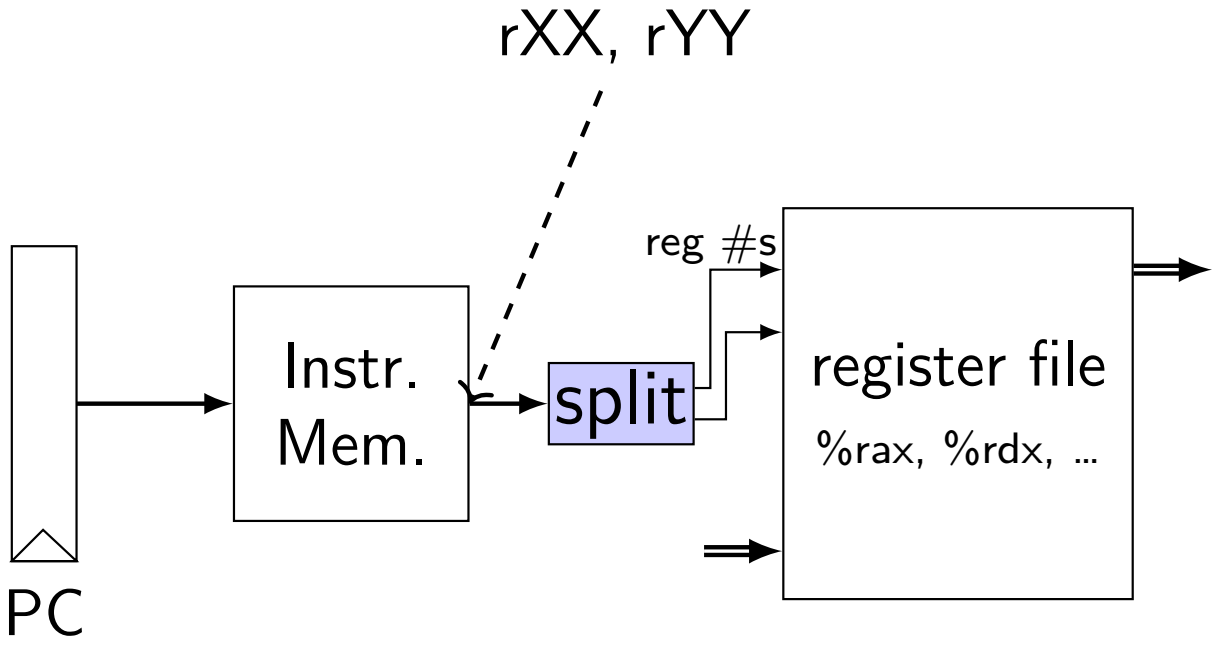
addq CPU



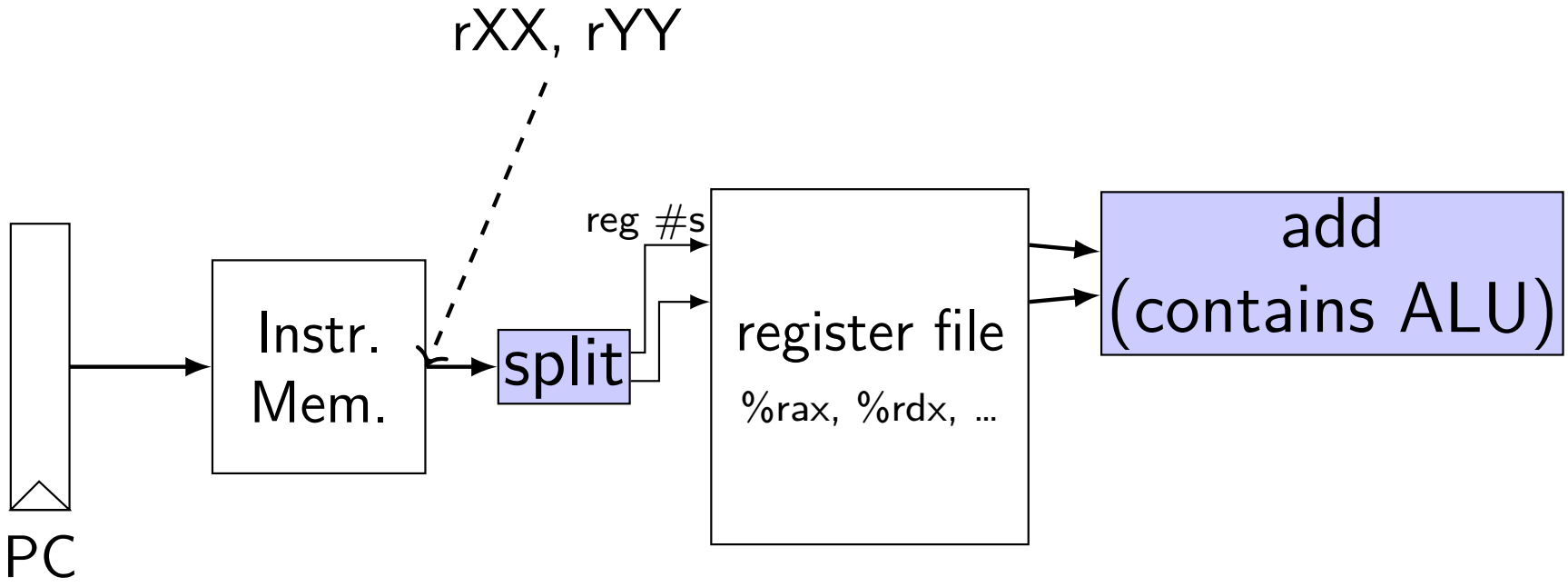
addq CPU



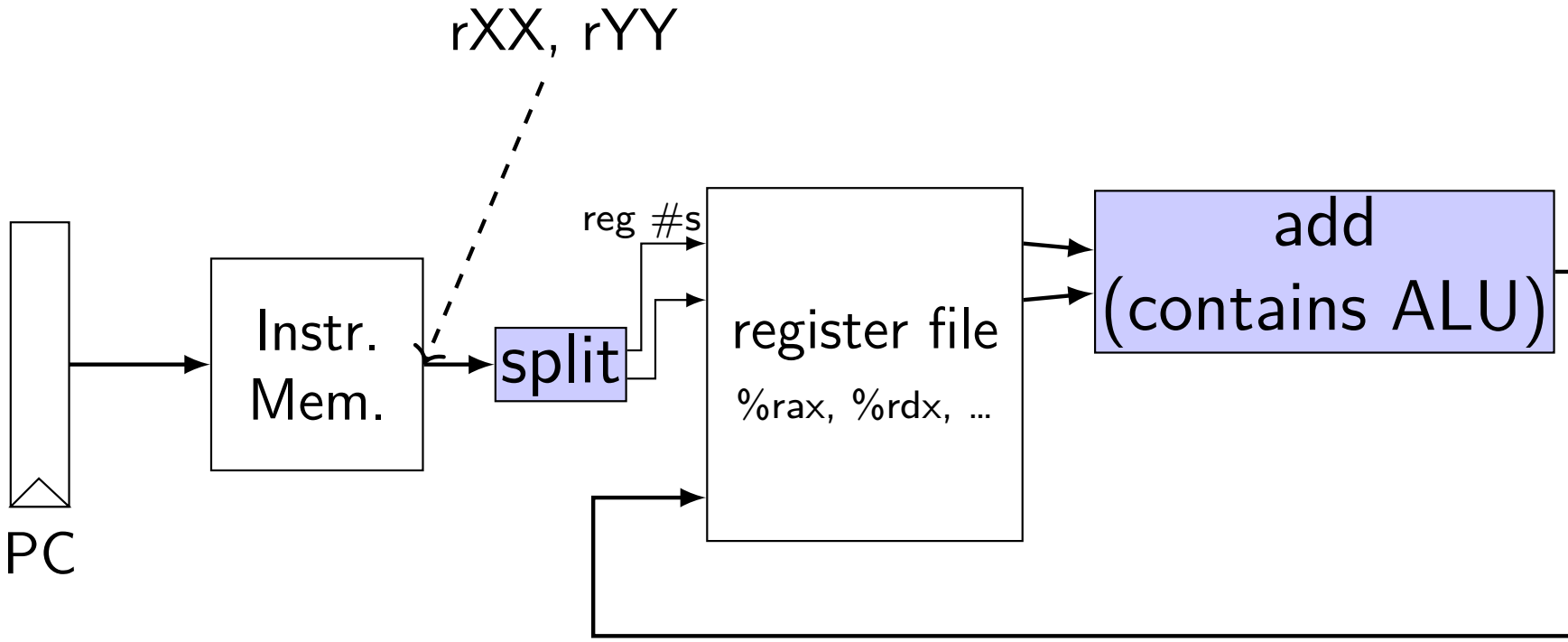
addq CPU



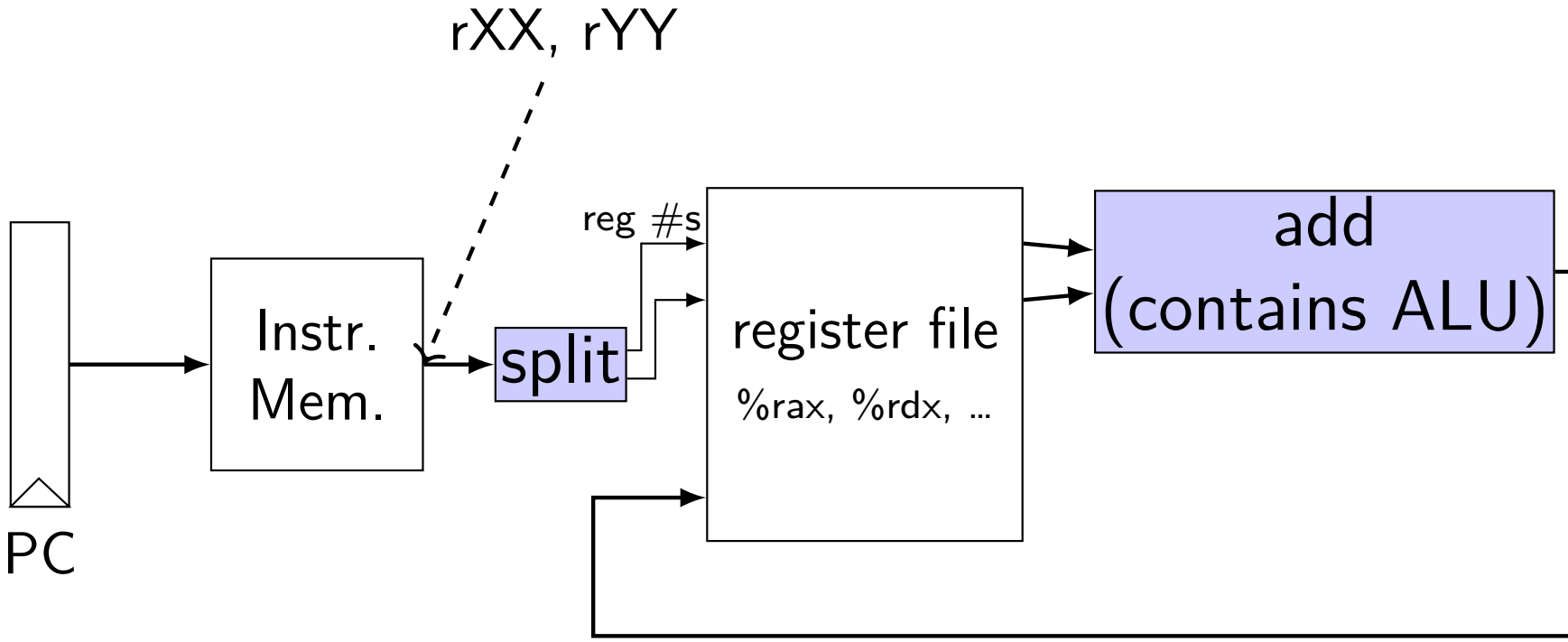
addq CPU


rXX, rYY
reg #s
Instr.
Mem.
split
register file
%rax, %rdx, ...
add
(contains ALU)
PC

addq CPU


rXX, rYY
reg #s
Instr.
Mem.
split
register file
%rax, %rdx, ...
add
(contains ALU)
PC

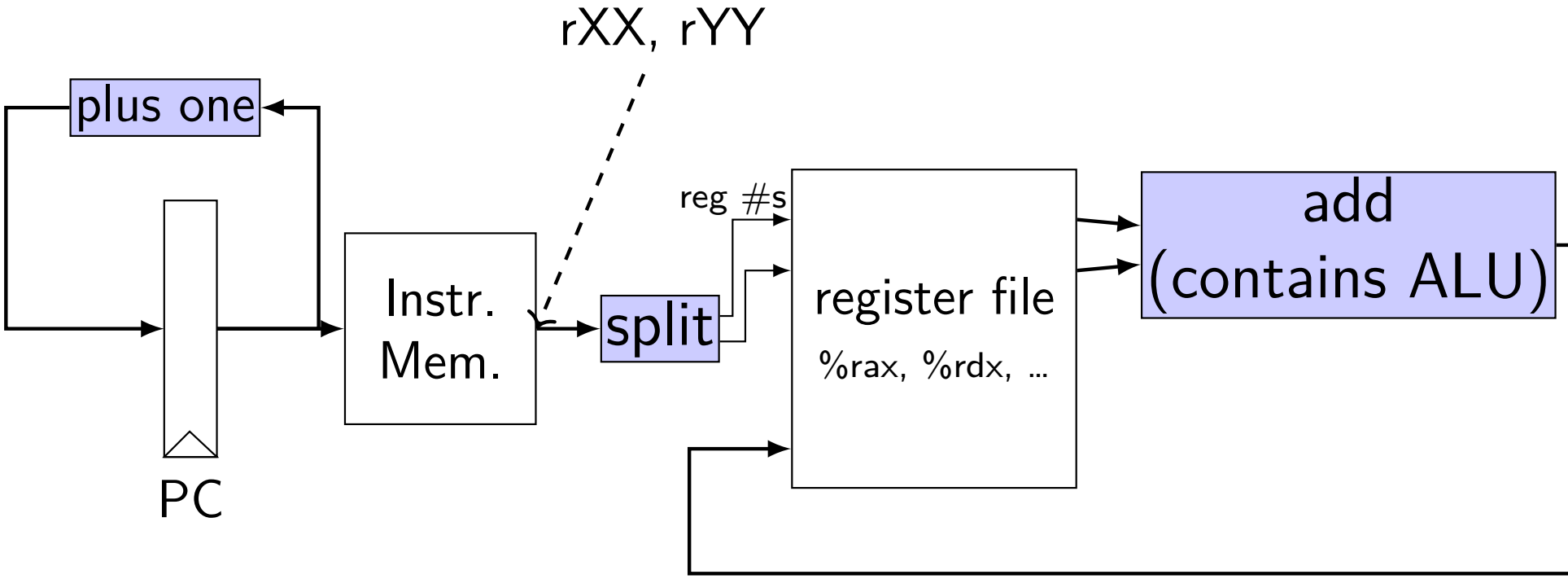
addq CPU



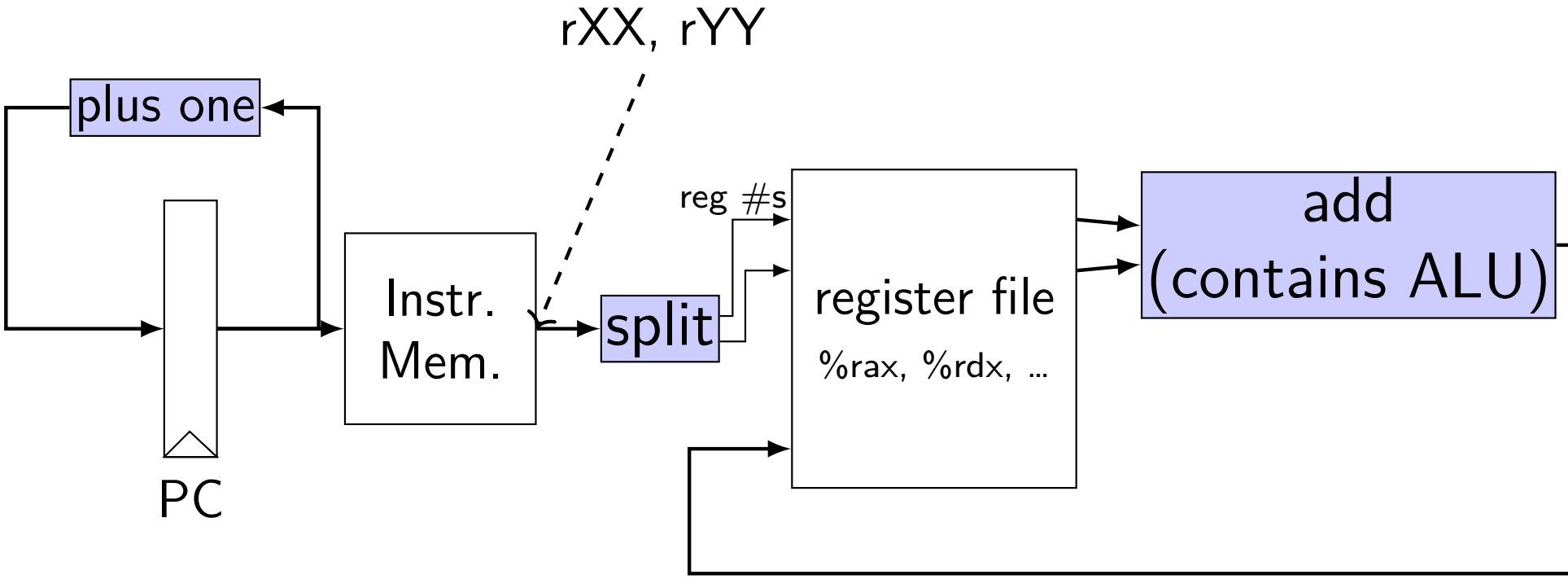

```
/* 0x00: */ addq %rax, %rdx
/* 0x01: */ addq %rbx, %rdx
```

initially:	PC = 0x00,	rax = 1,	rbx = 2,	rdx = 3
after cycle 1:	PC = ????,	rax = 1,	rbx = 2,	rdx = 4
after cycle 2:	PC = ????,	rax = ??,	rbx = ??,	rdx = ??

addq CPU


rXX, rYY
plus one
reg #s
add
(contains ALU)
Instr.
Mem.
split
register file
%rax, %rdx, ...
PC

addq CPU




```
/* 0x00: */ addq %rax, %rdx
/* 0x01: */ addq %rbx, %rdx
```

initially:	PC = 0x00,	rax = 1,	rbx = 2,	rdx = 3
after cycle 1:	PC = 0x01,	rax = 1,	rbx = 2,	rdx = 4
after cycle 2:	PC = 0x02,	rax = 1,	rbx = 2,	rdx = 6

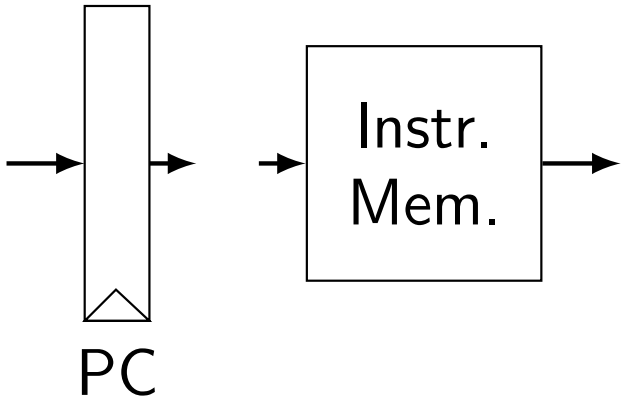
Simple ISA 2: jmp

```
jmp label
```

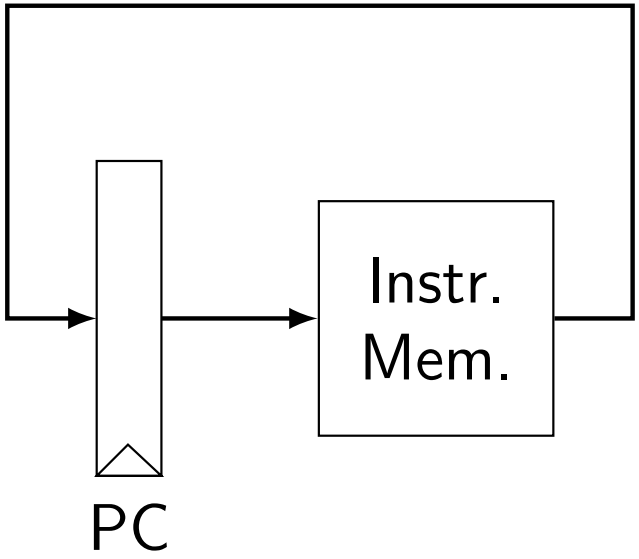
encoding: *8-byte little-endian address*

8 byte instructions, no opcode

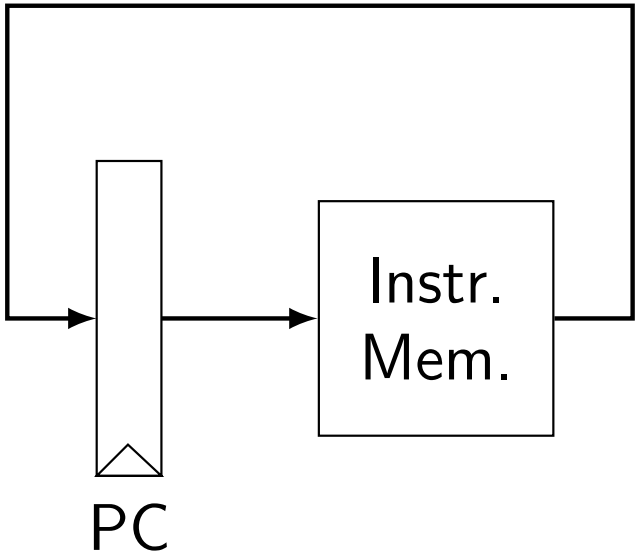
jmp CPU



jmp CPU



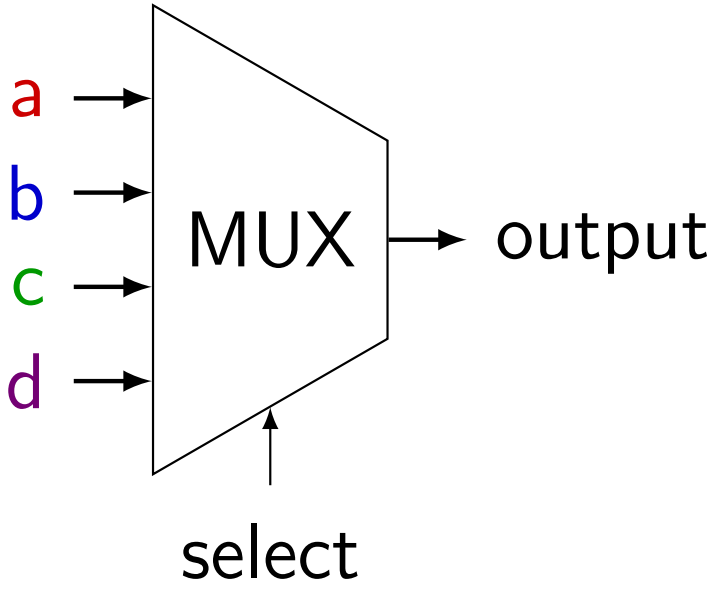
jmp CPU



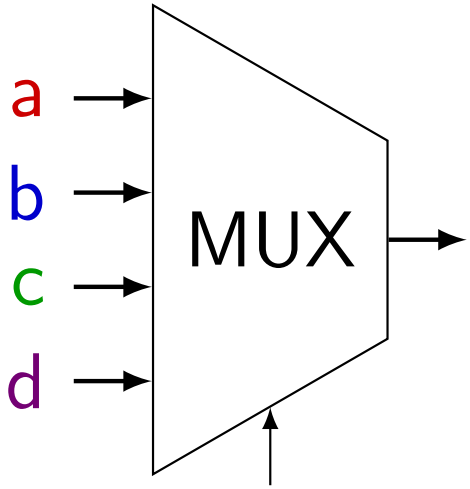

```
/* 0x00: */ jmp 0x10
/* 0x08: */ jmp 0x00
/* 0x10: */ jmp 0x08
```

initially: PC = 0x00
after cycle 1: PC = 0x10
after cycle 2: PC = 0x08
after cycle 3: PC = 0x00

Multiplexers



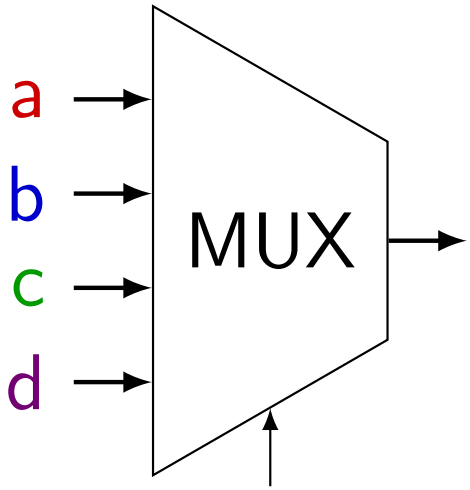
Multiplexers



output $=$ a or b or c or d

select $=$ 0 or 1 or 2 or 3

Multiplexers



output = a or b or c or d

select = 0 or 1 or 2 or 3

truth table:

select bit 1	select bit 0	output (many bits)
0	0	a
0	1	b
1	0	c
1	1	d

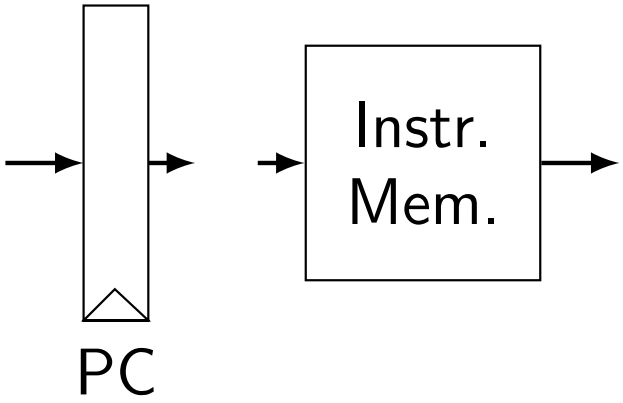
Simple ISA 3: Jmp or No-Op

actual subset of Y86-64

`jmp LABEL` — encoded as `0x70` + address

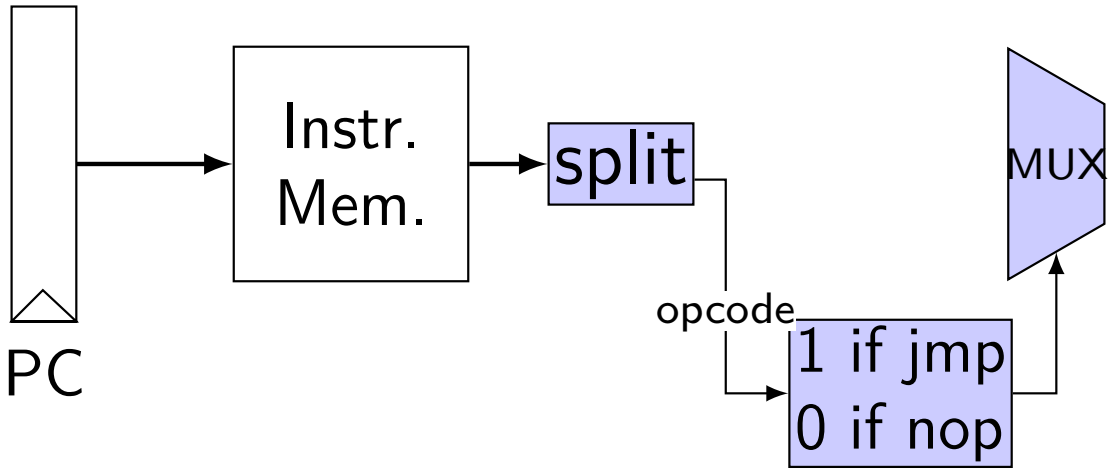
`nop` — encoded as `0x10`

jmp+nop CPU



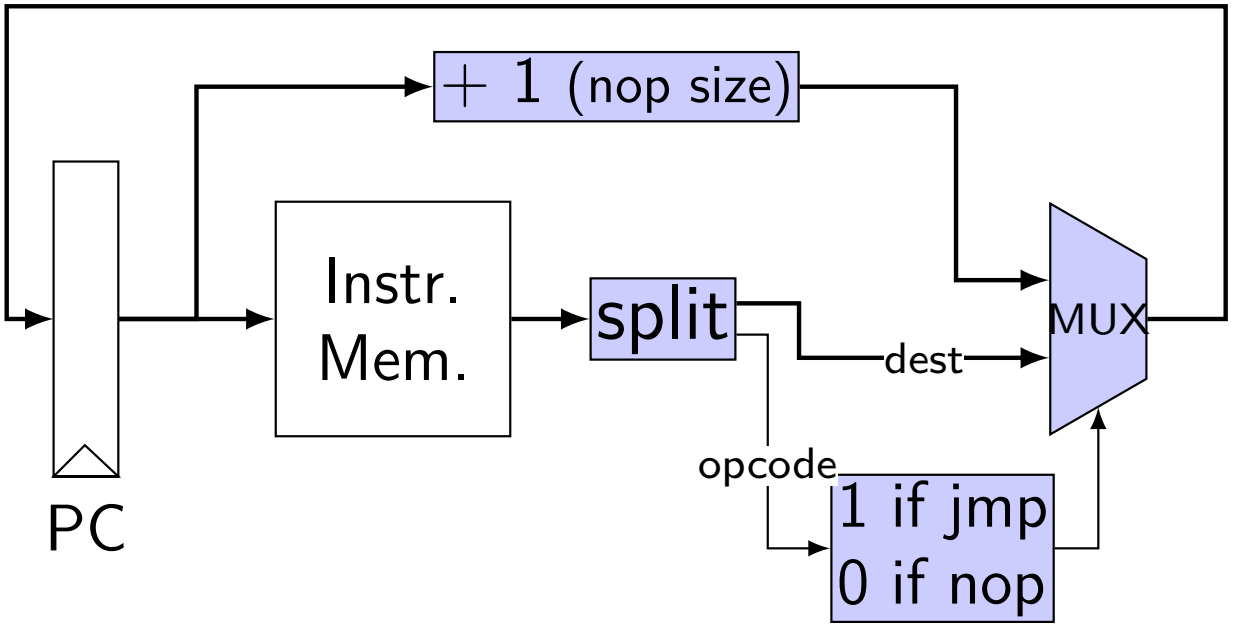
nop	1	0	
jmp *Dest*	7	0	*Dest*

jmp+nop CPU



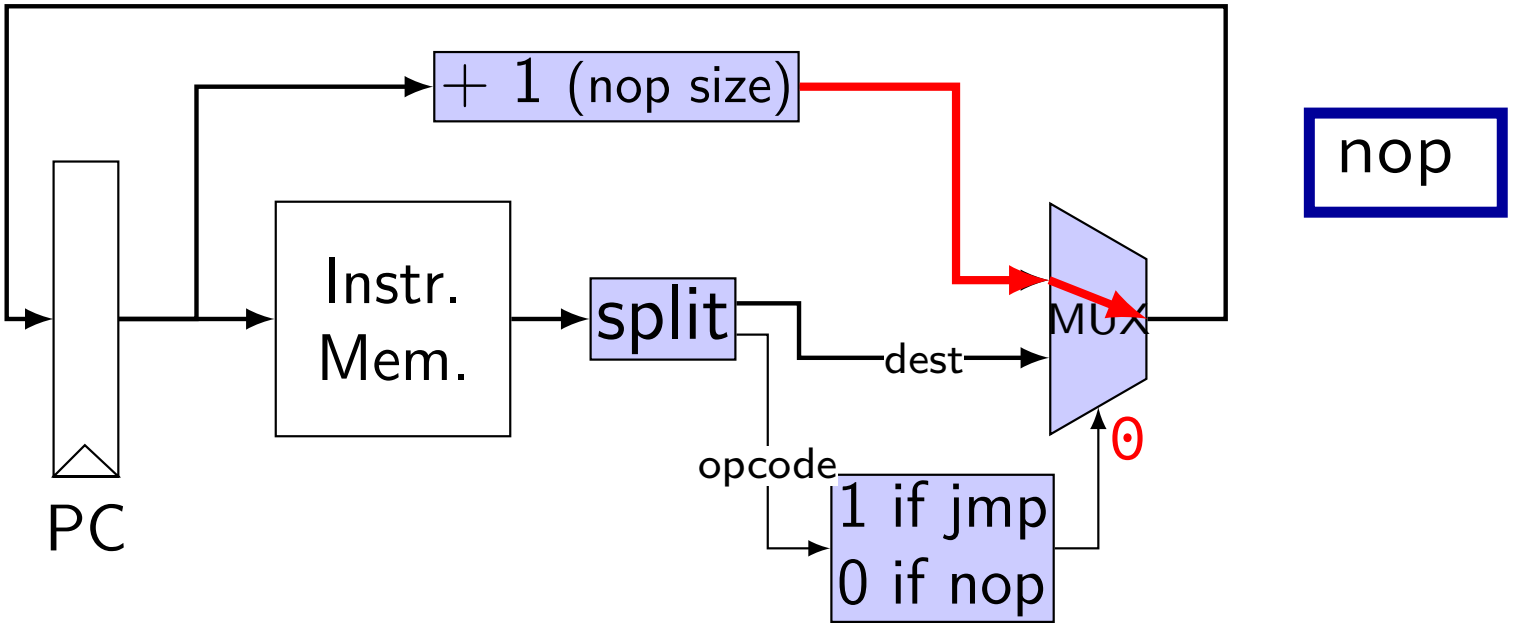
nop	1	0	
jmp *Dest*	7	0	*Dest*

jmp+nop CPU



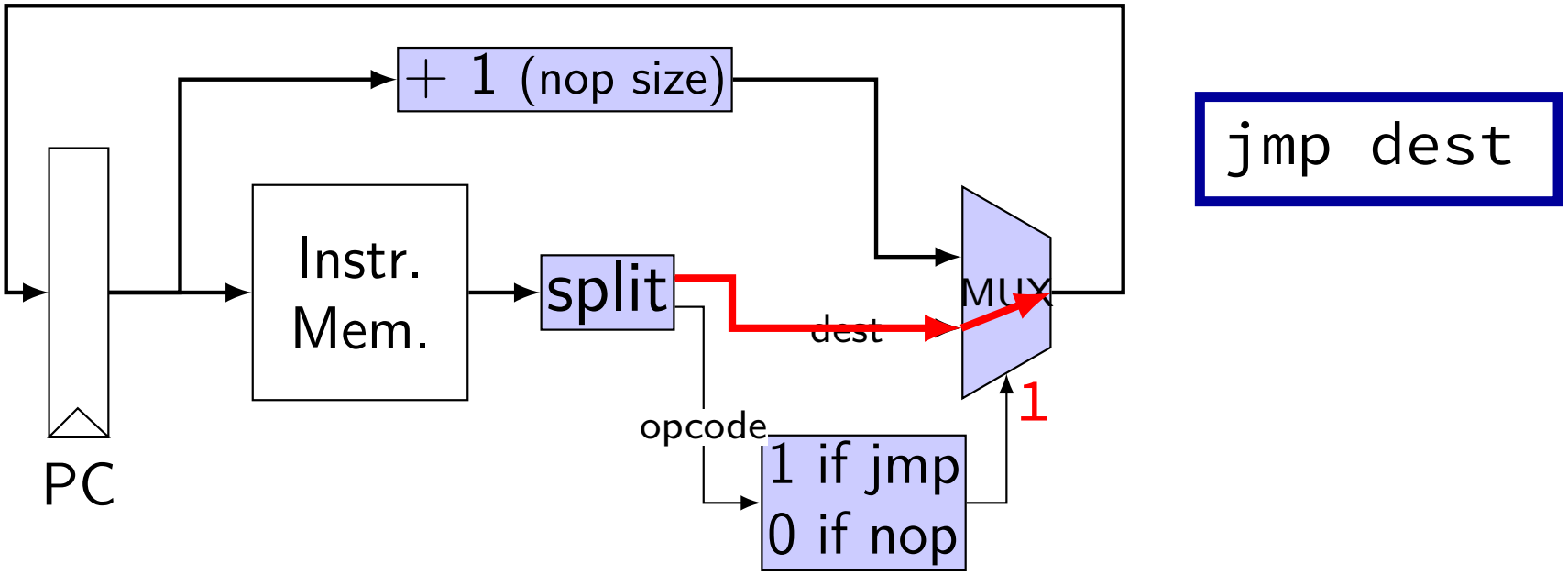
nop	1	0	
jmp *Dest*	7	0	*Dest*

jmp+nop CPU



nop	1	0	
jmp *Dest*	7	0	*Dest*

jmp+nop CPU

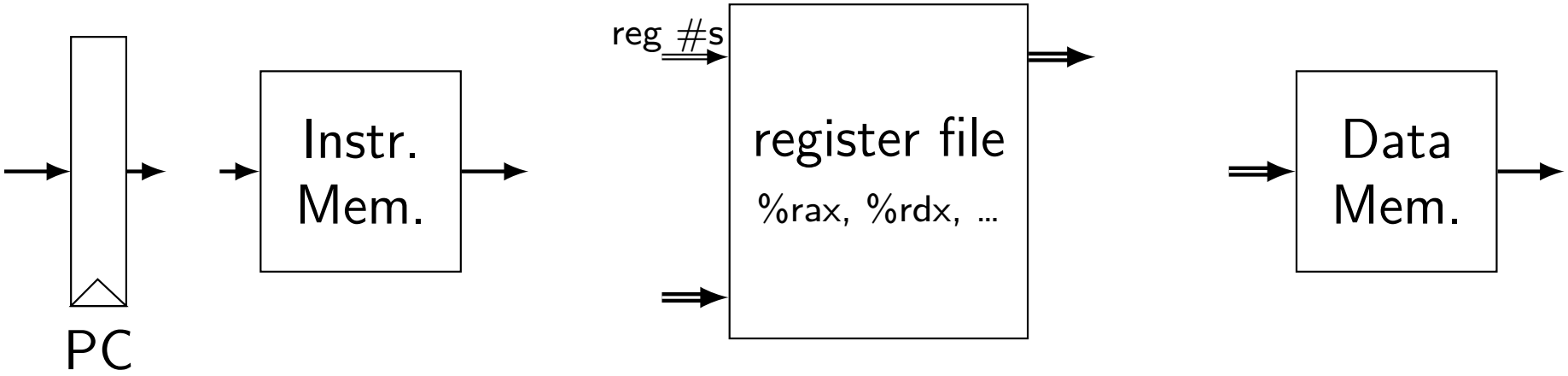


nop	1	0	
jmp *Dest*	7	0	*Dest*

Simple ISA 4: Mov-to-register

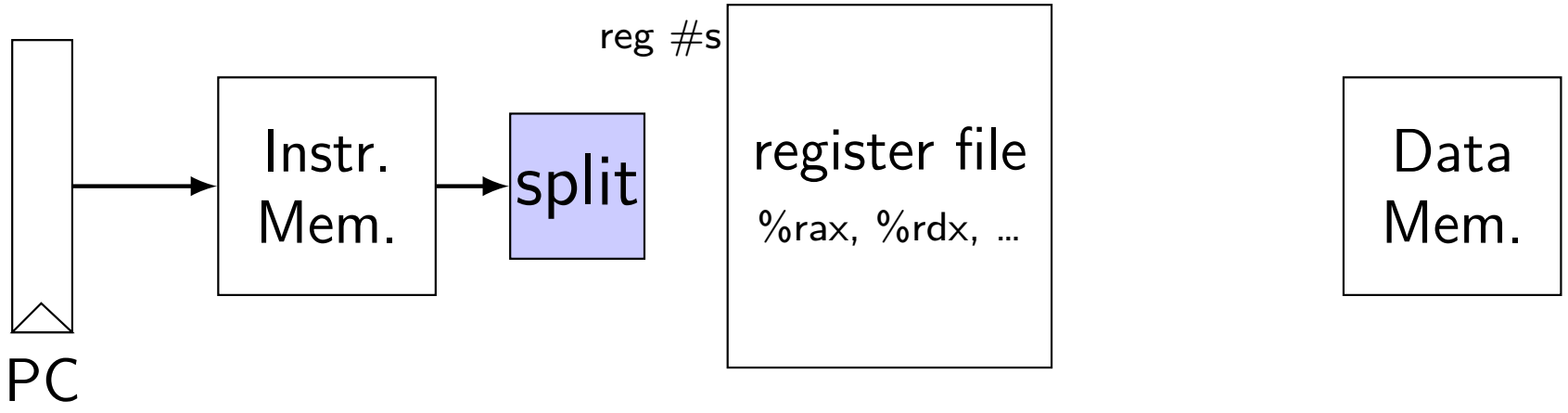
```
irmovq $constant, %rYY

rrmovq %rXX, %rYY

mrmovq 10(%rXX), %rYY
```

mov-to-register CPU



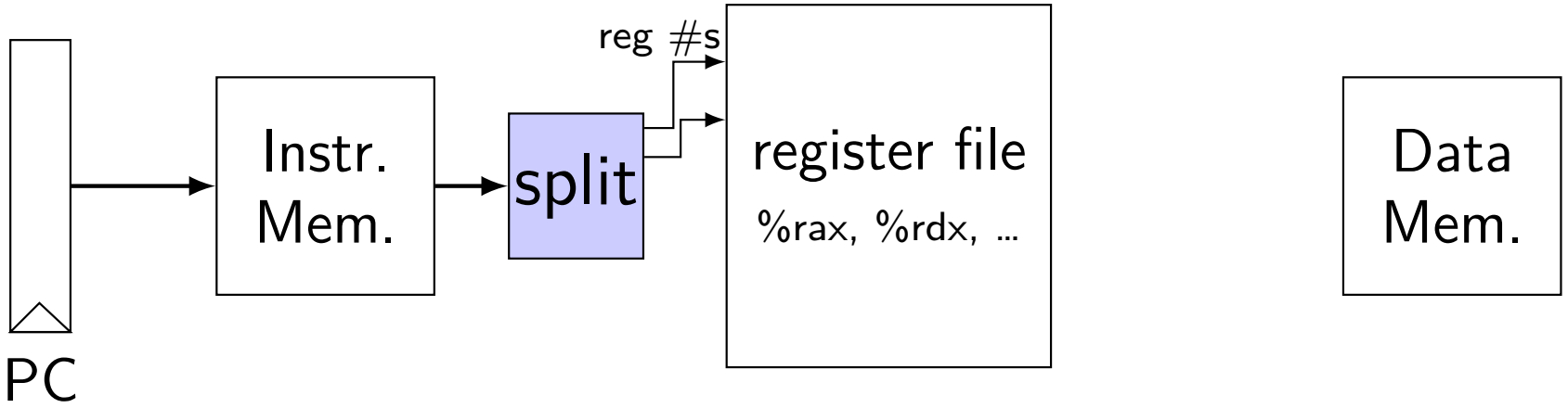
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU



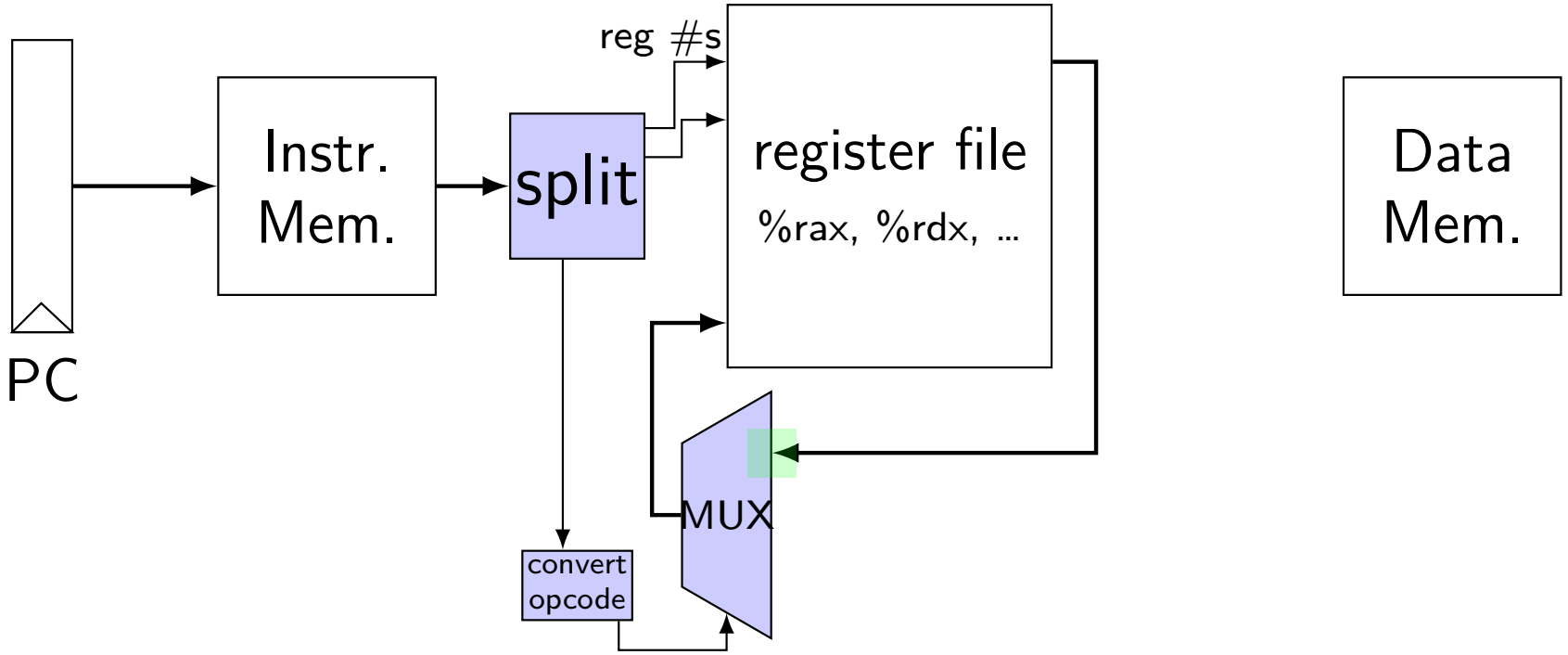
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU



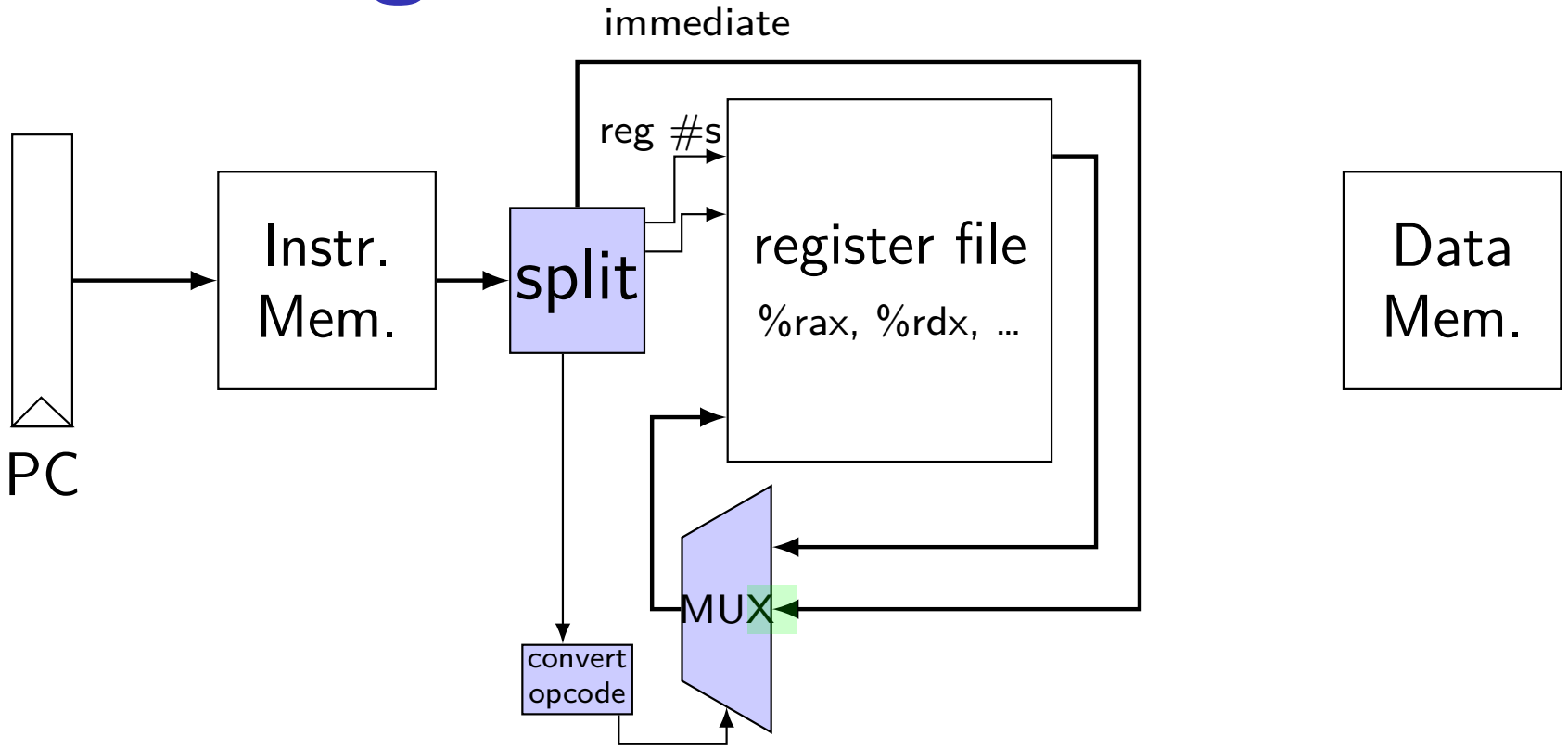
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU



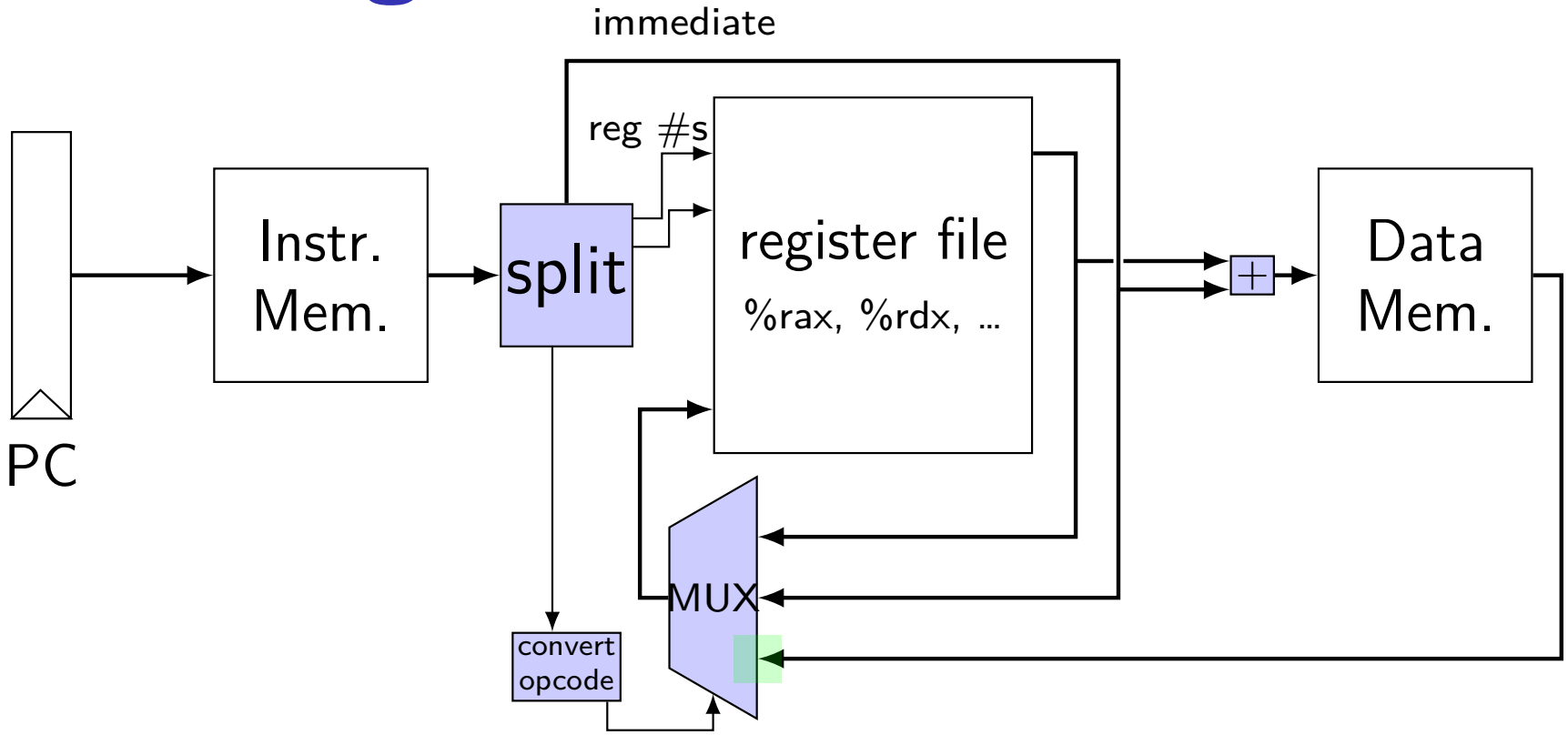
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU



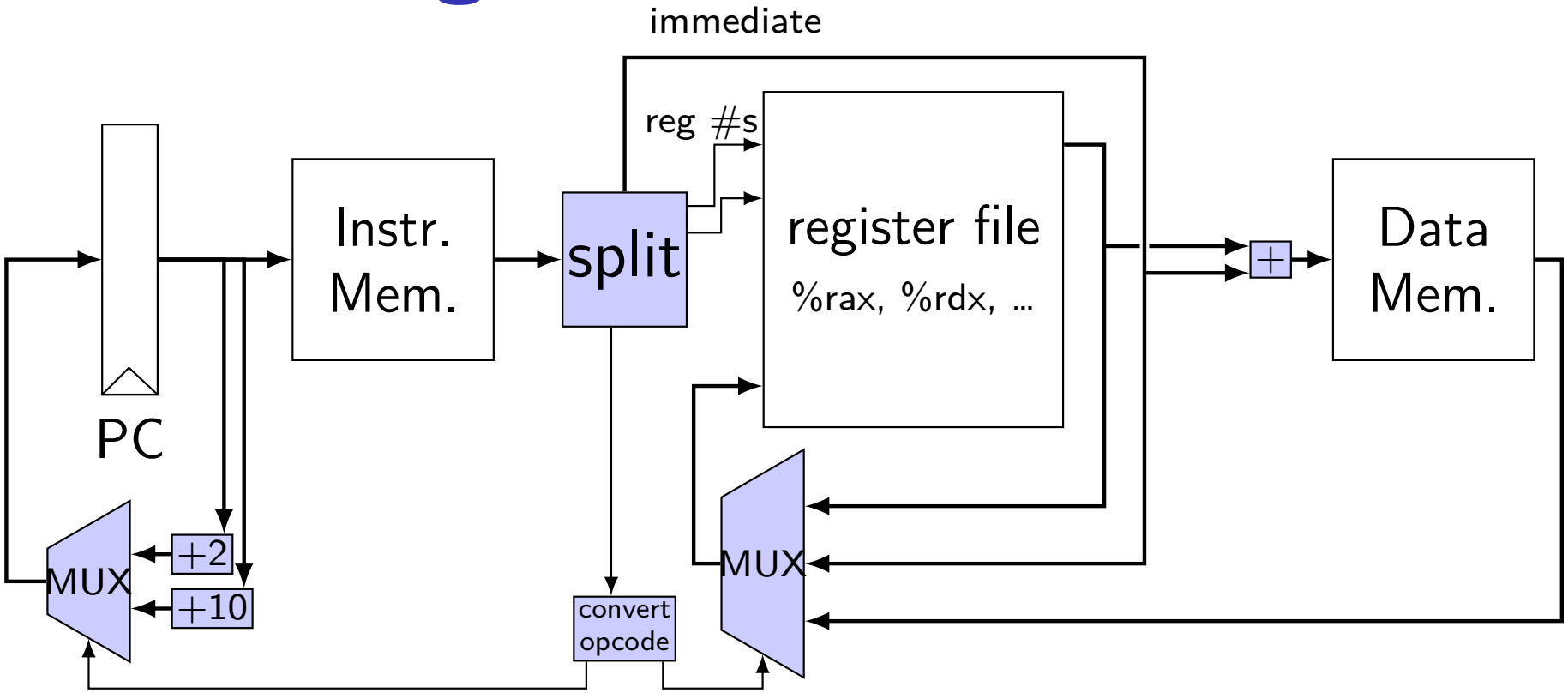
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU



rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D(rB)*, *rA*	5	0	*rA*	*rB*	*D*

mov-to-register CPU

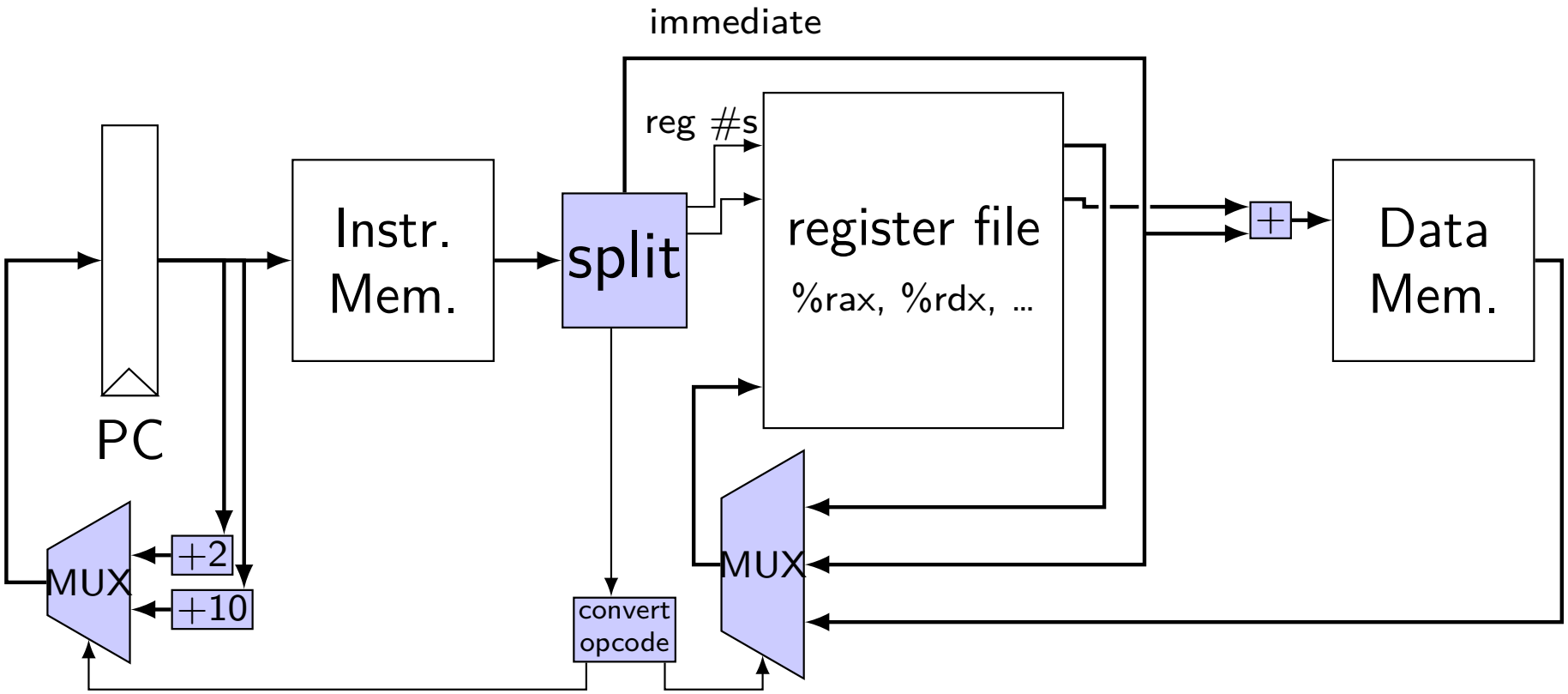


rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

Simple ISA 4B: Mov

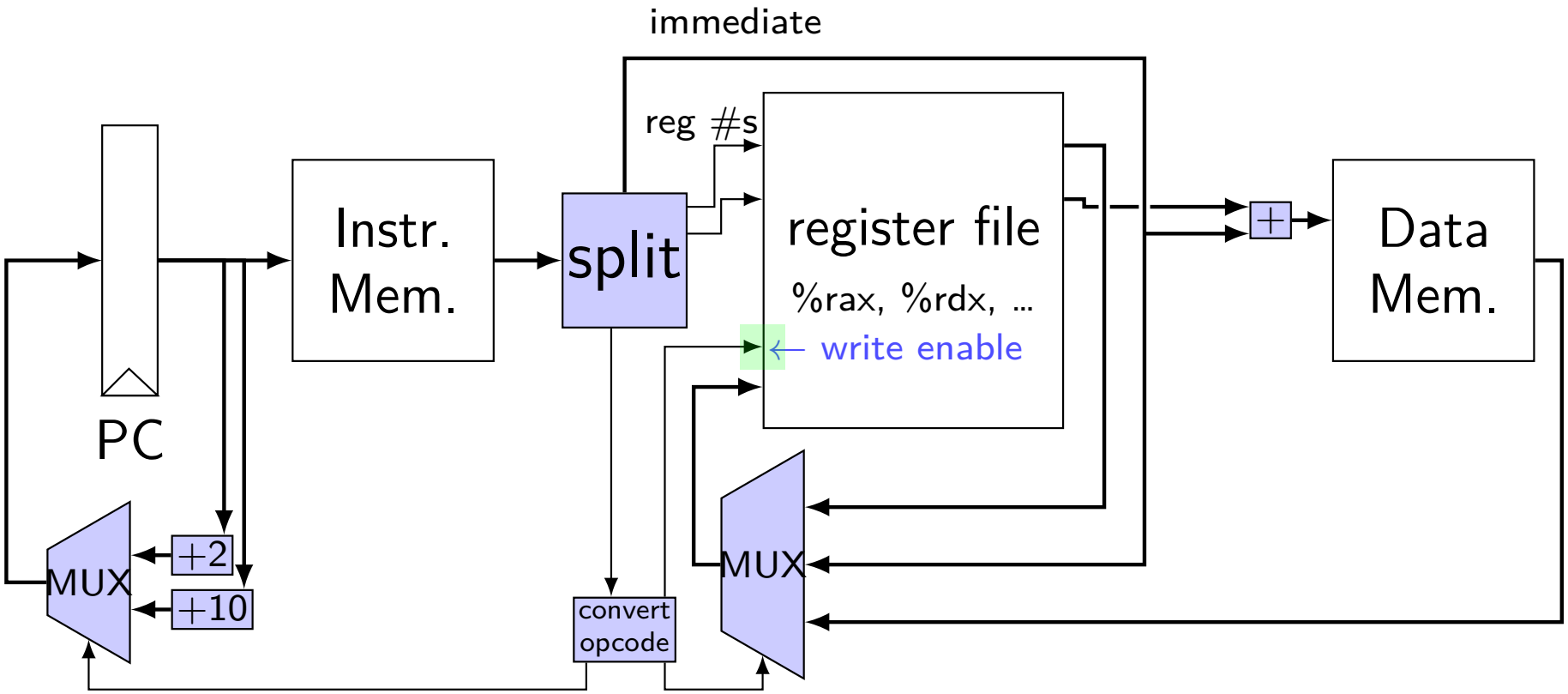
```
irmovq $constant, %rYY
rrmovq %rXX, %rYY
mrmovq 10(%rXX), %rYY
rmmovq %rXX, 10(%rYY)
```

mov CPU



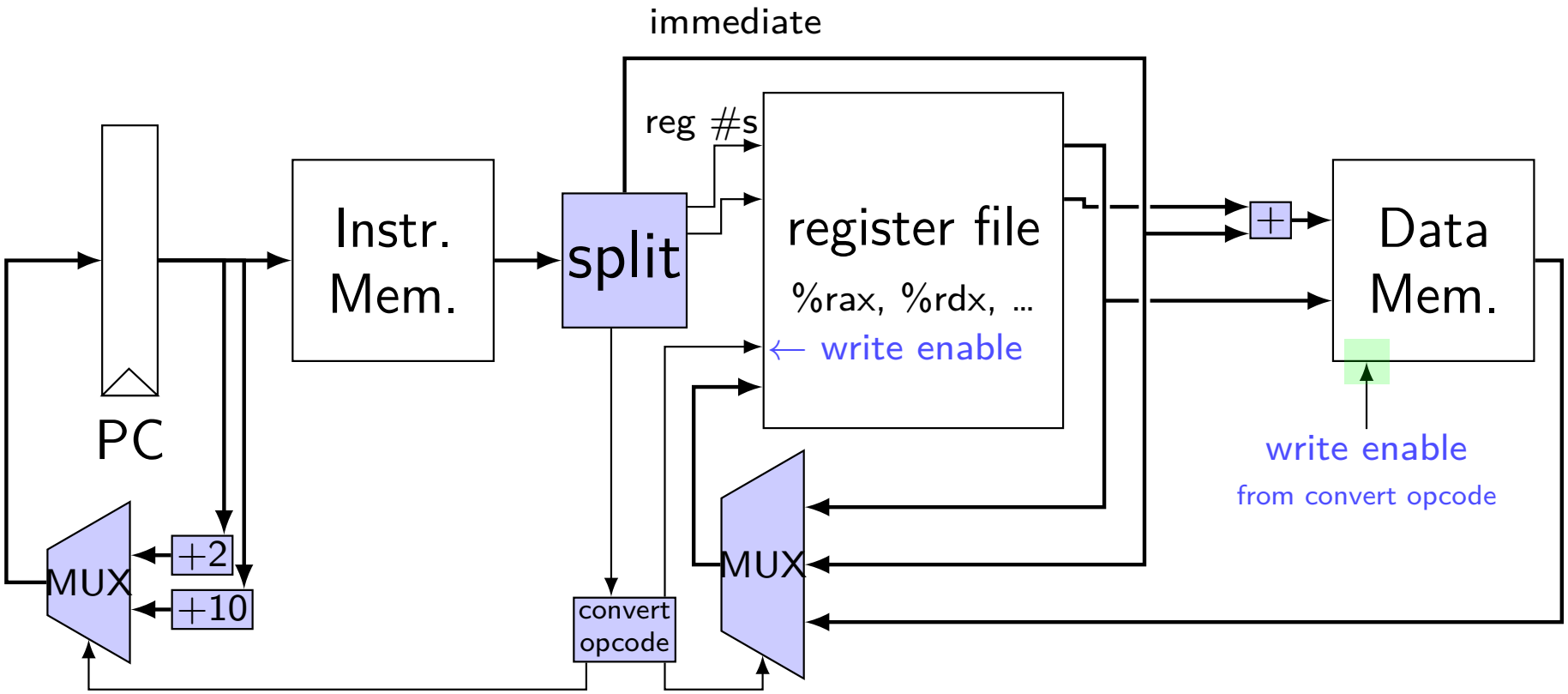
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D(rB)*, *rA*	5	0	*rA*	*rB*	*D*
rmmovq *rA*, *D(rB)*	4	0	*rA*	*rB*	*D*

mov CPU



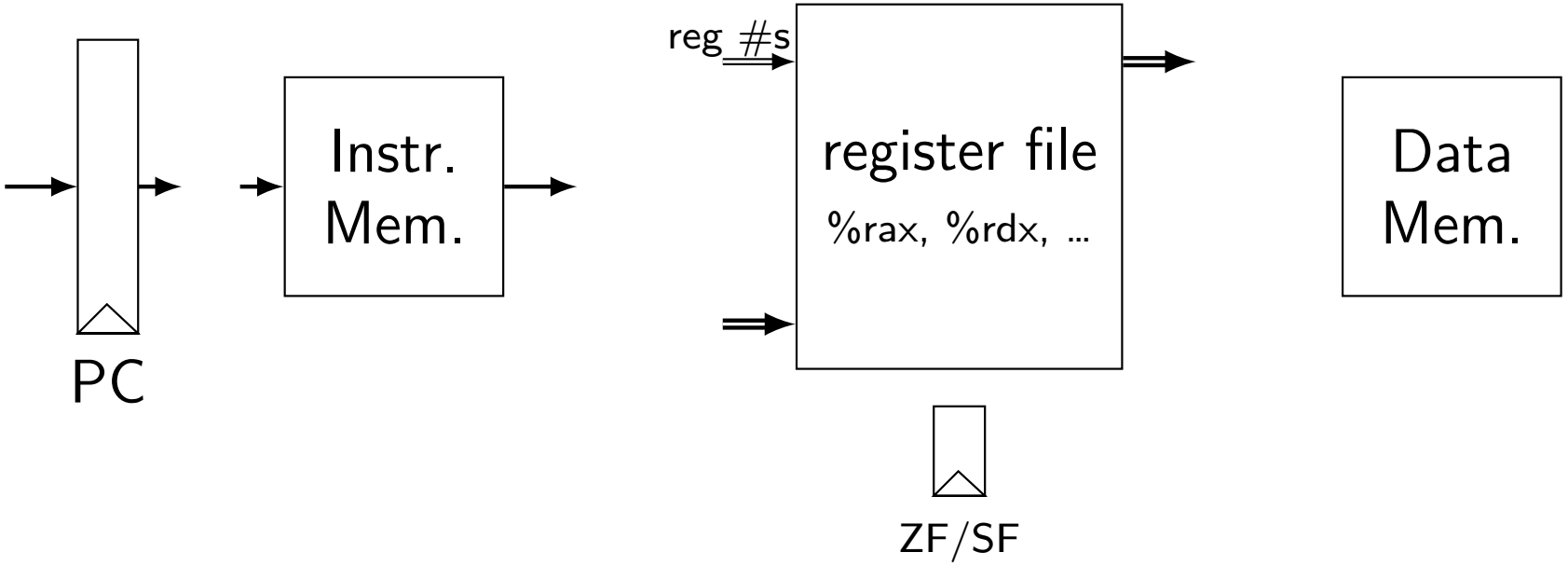
rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D(rB)*, *rA*	5	0	*rA*	*rB*	*D*
rmmovq *rA*, *D(rB)*	4	0	*rA*	*rB*	*D*

mov CPU

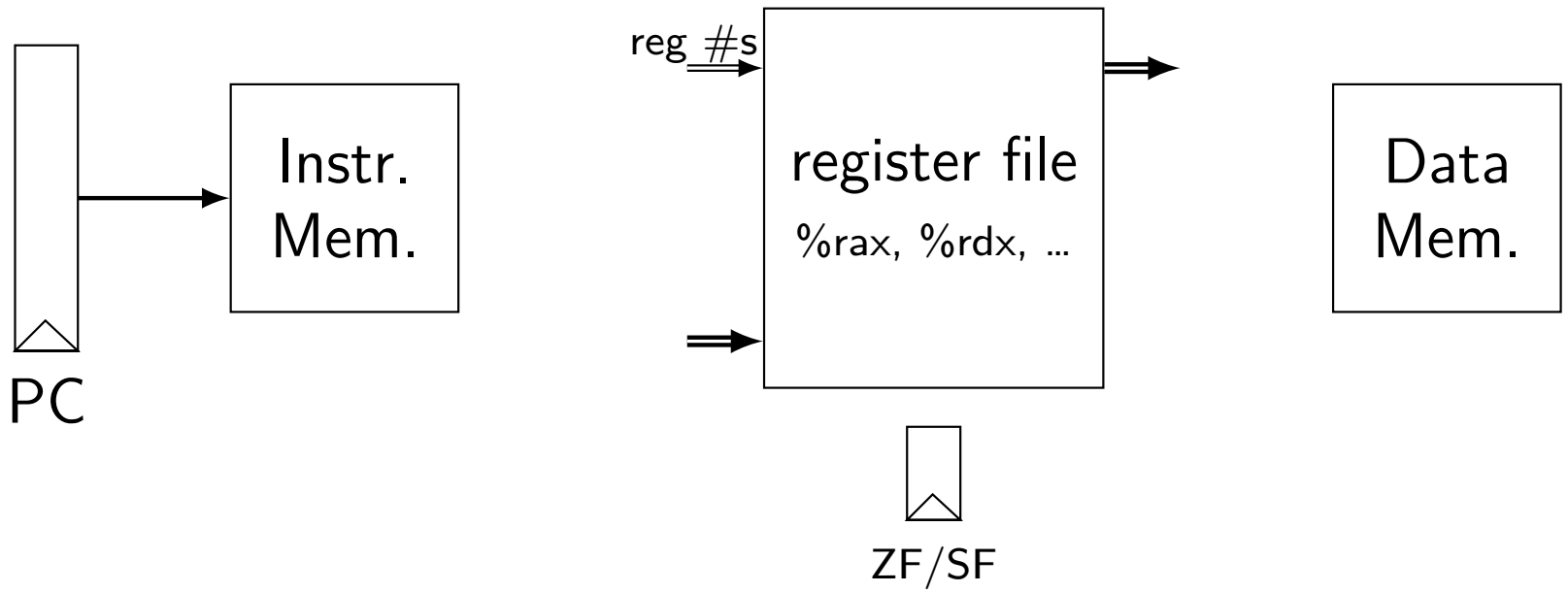


rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*
rmmovq *rA*, *D*(*rB*)	4	0	*rA*	*rB*	*D*

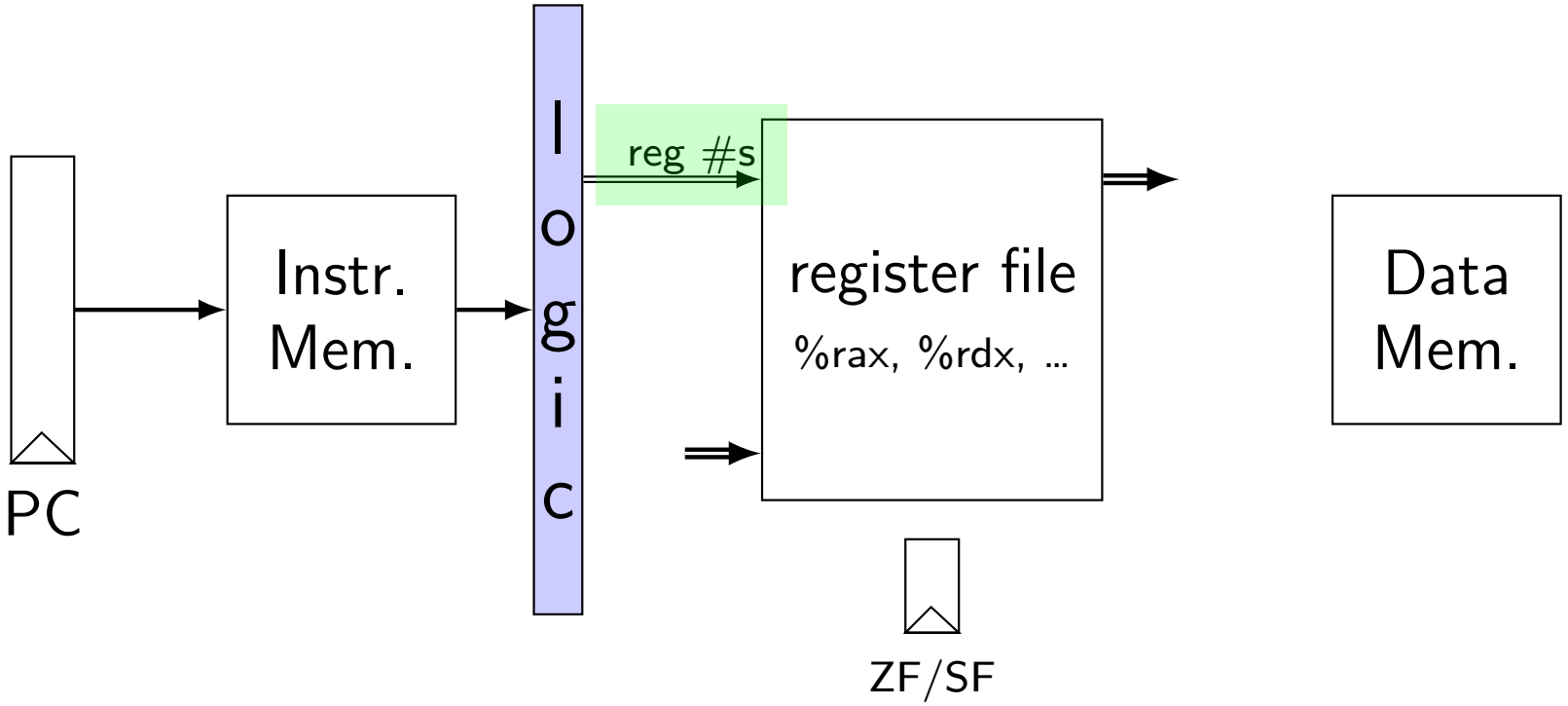
Connections in Y86-64



Connections in Y86-64

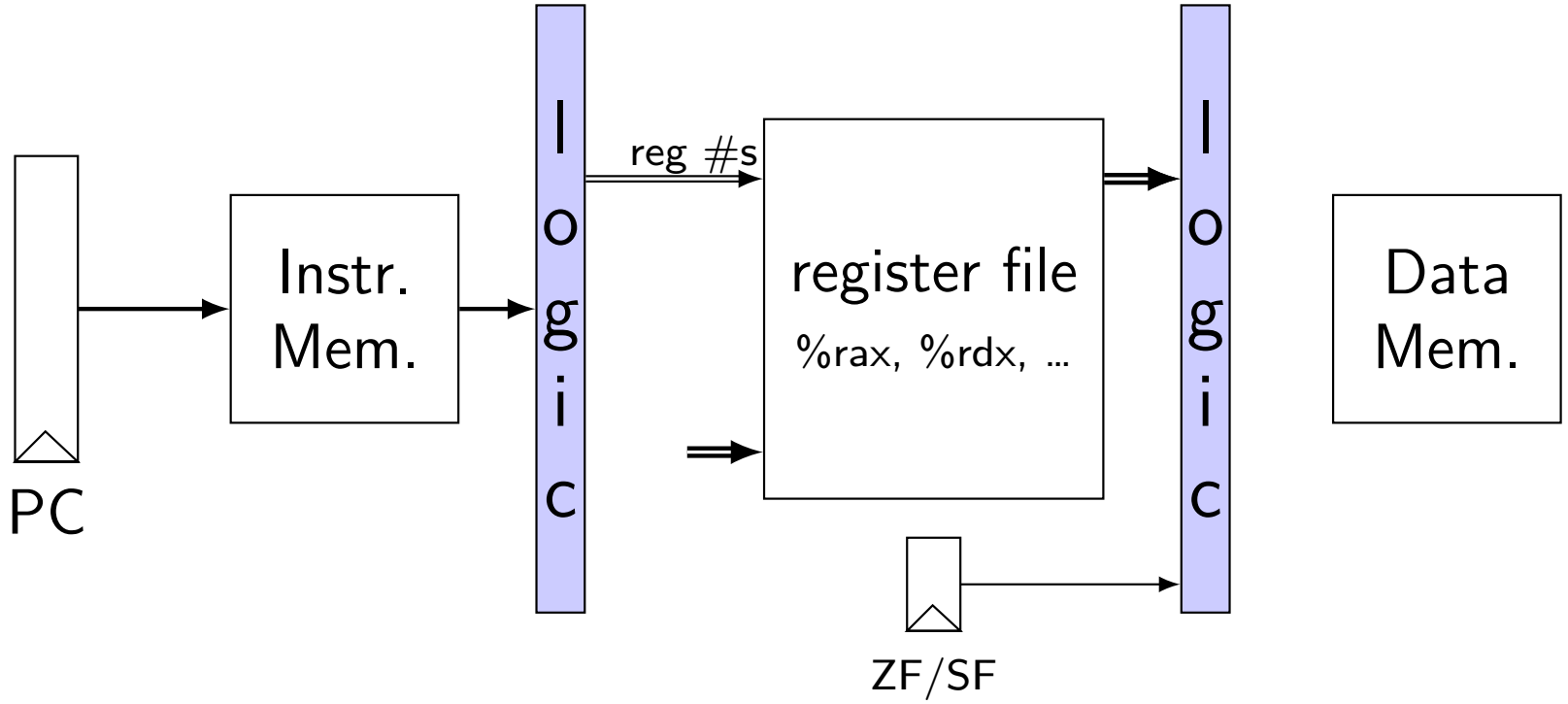


Connections in Y86-64

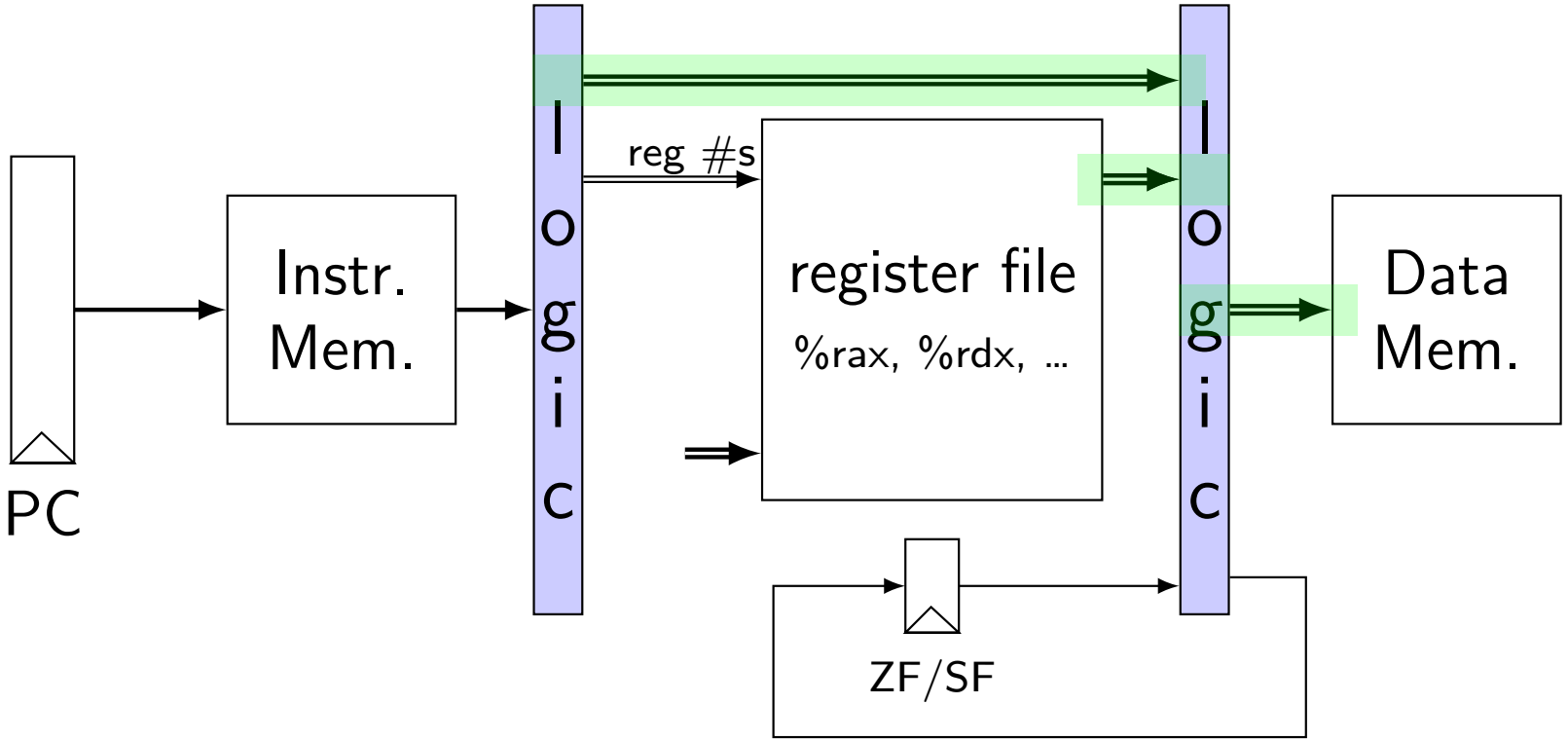



```
addq %r8, %r9
pushq %r8 (and %rsp)
```

Connections in Y86-64

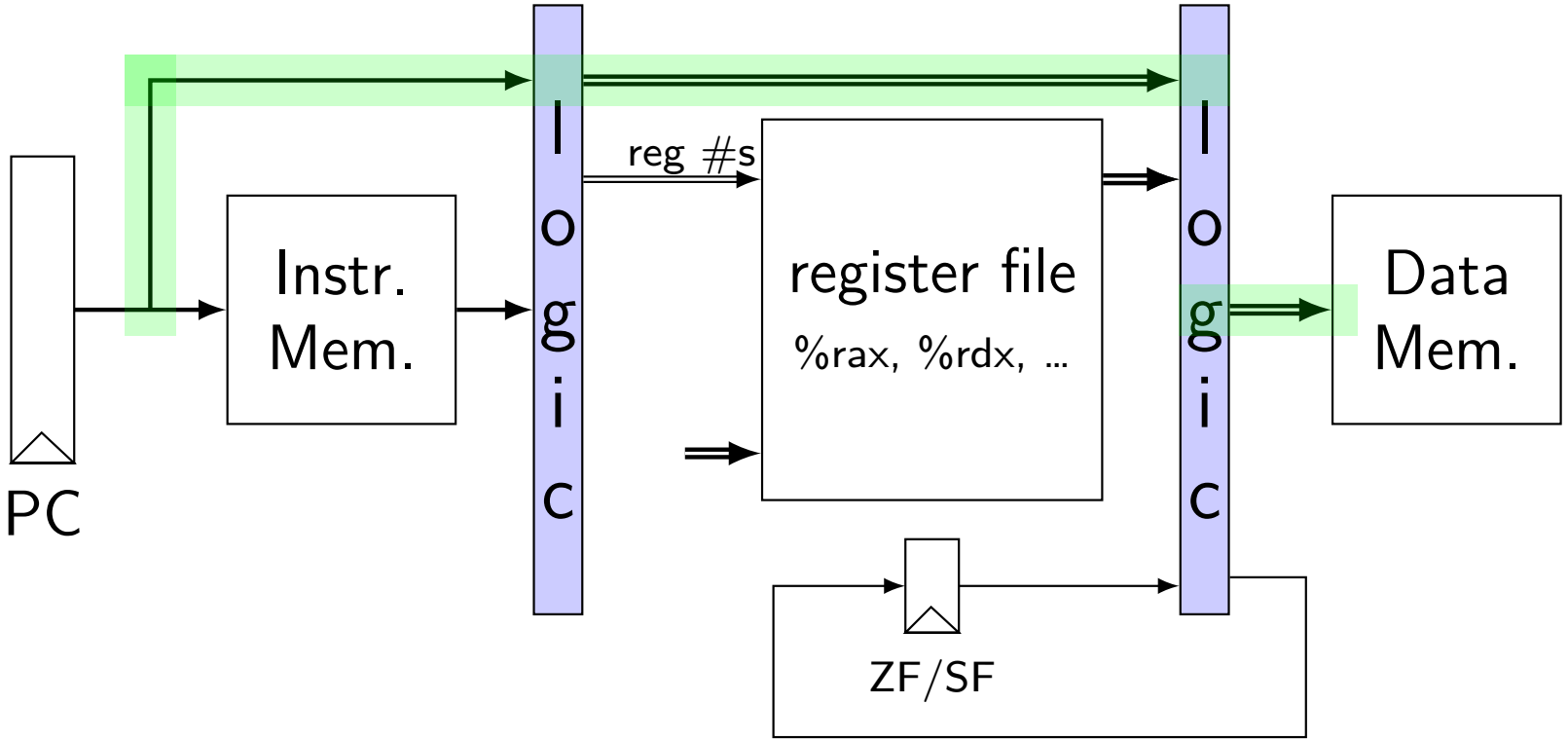


Connections in Y86-64



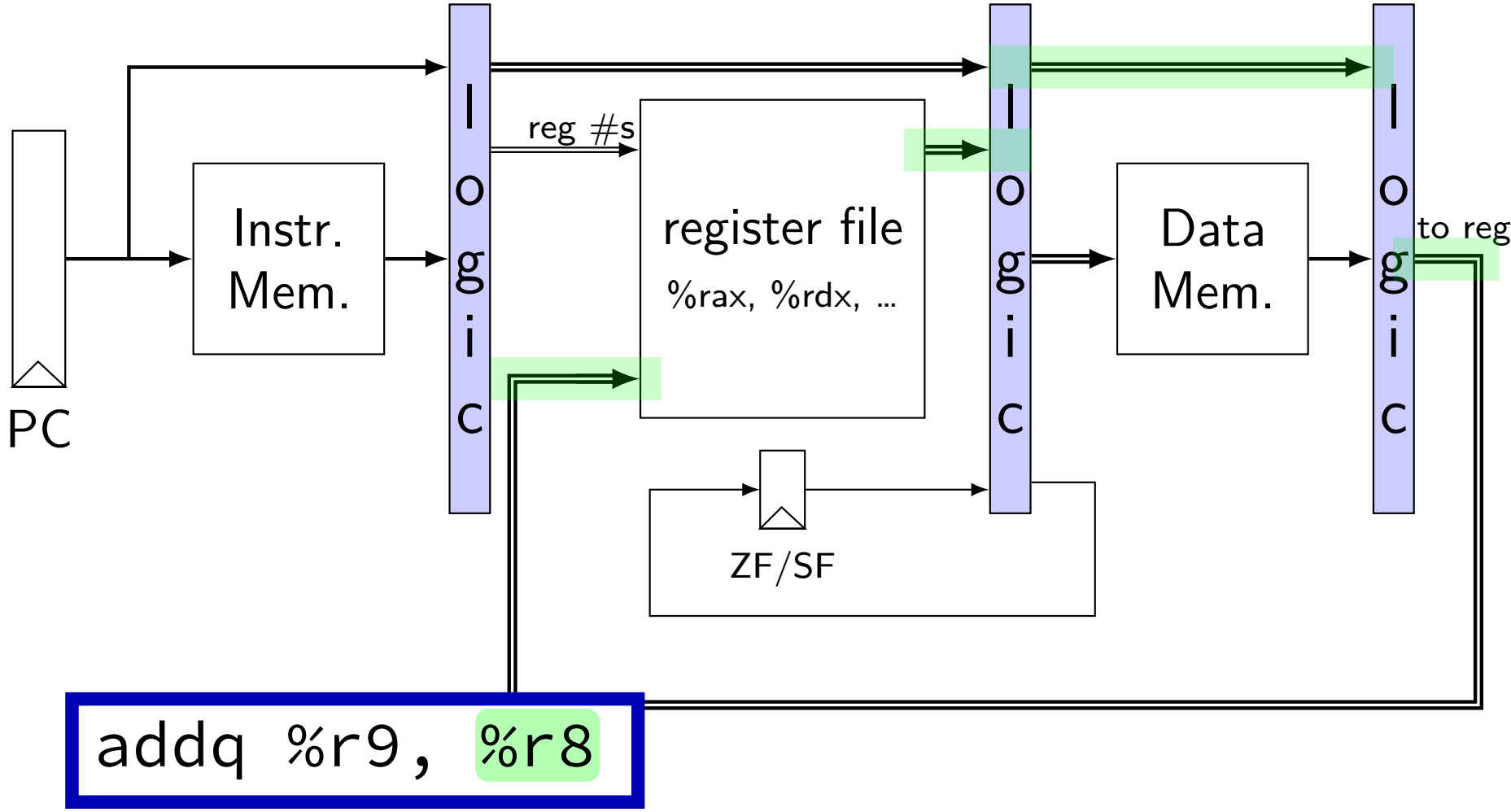

```
mrmovq 1000(%r9), %r8
rmmovq %r8, 1000(%r9)
```

Connections in Y86-64

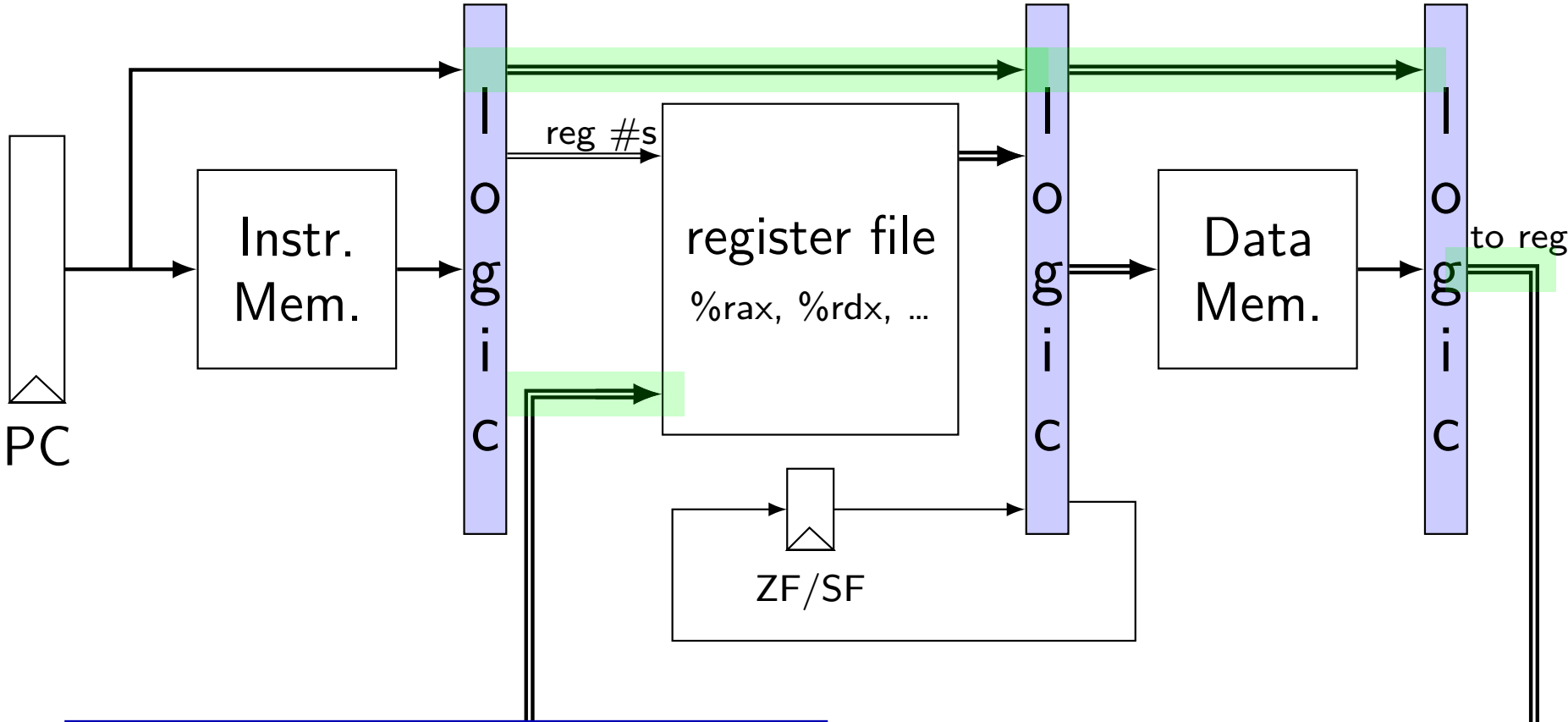


`call function` (saves next PC)

Connections in Y86-64

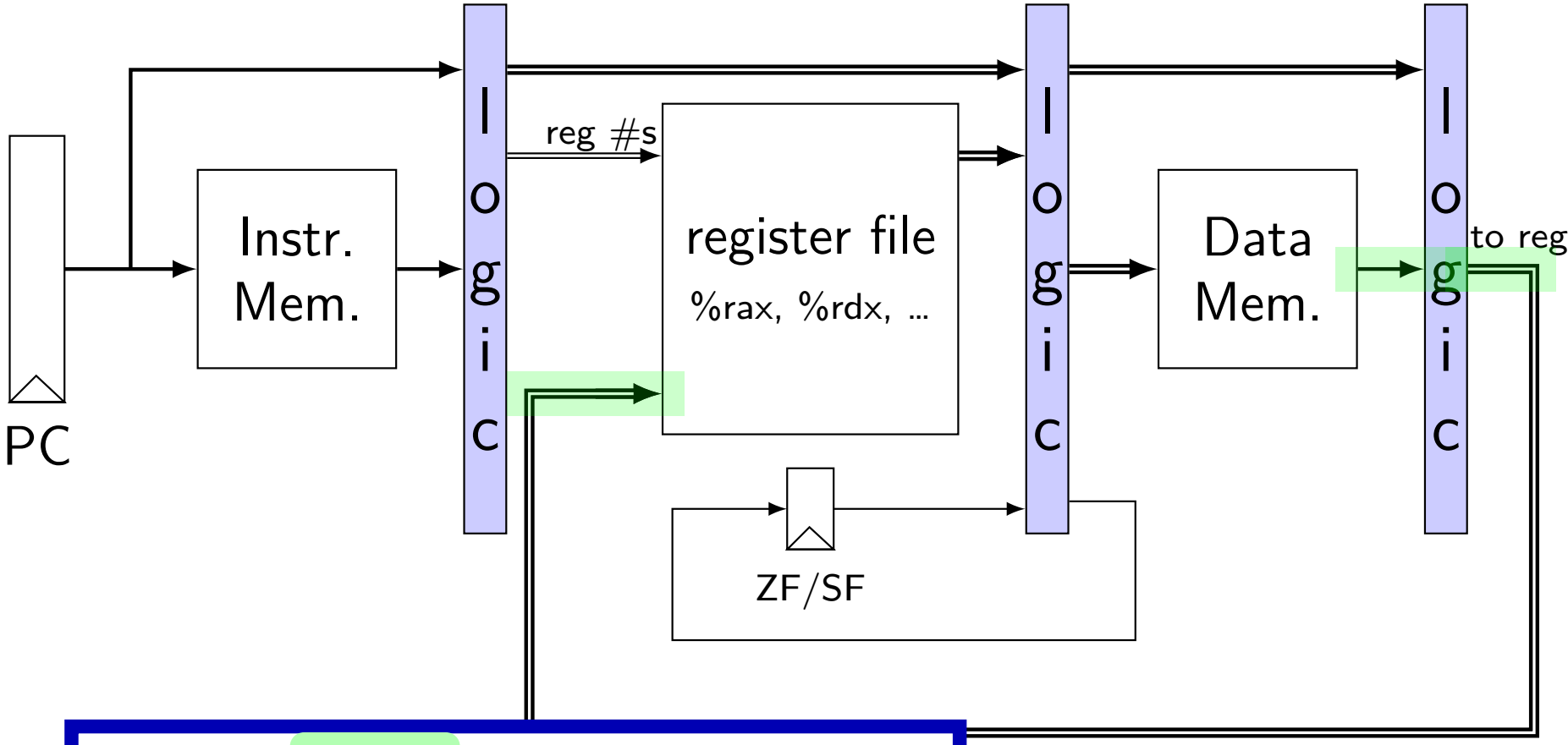


Connections in Y86-64



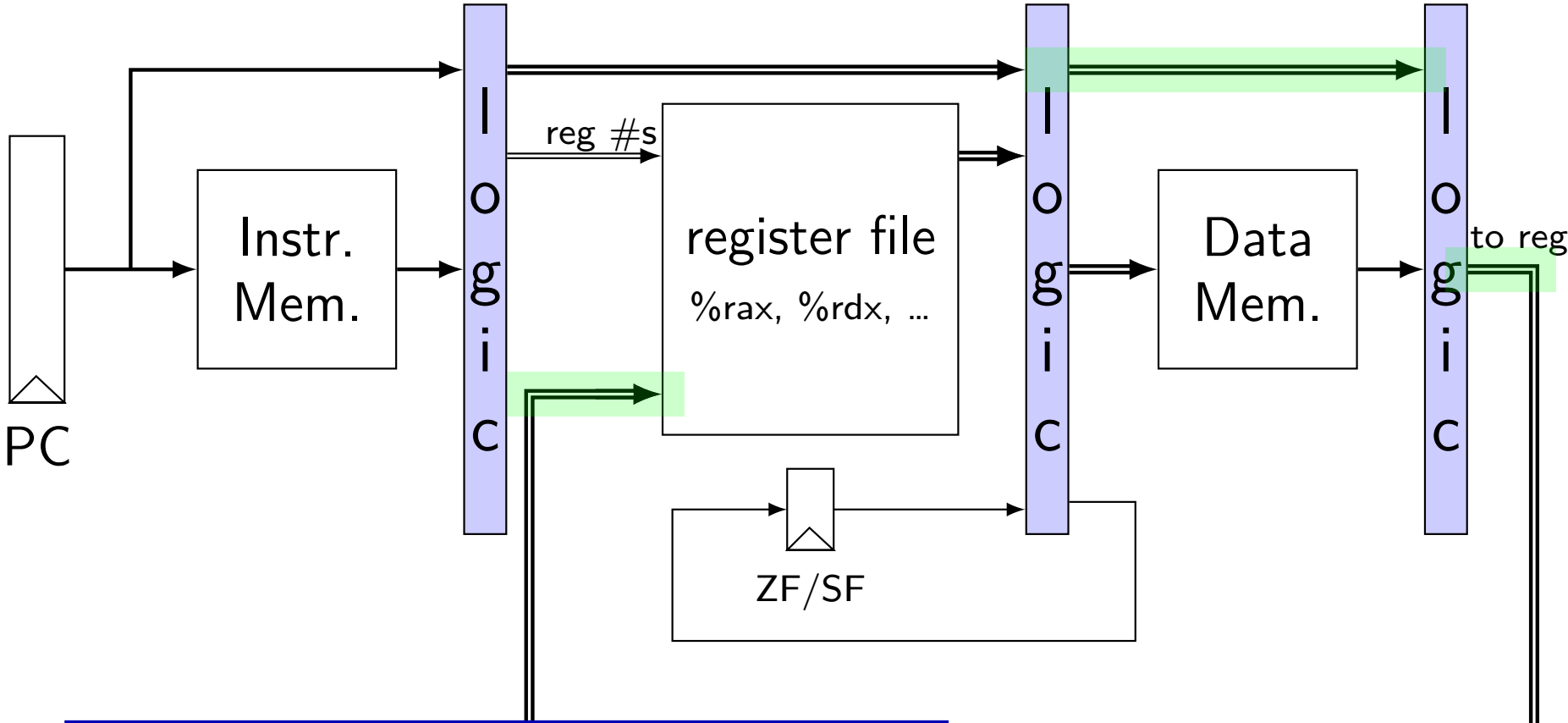

```
irmovq $1000, %r8
```

Connections in Y86-64



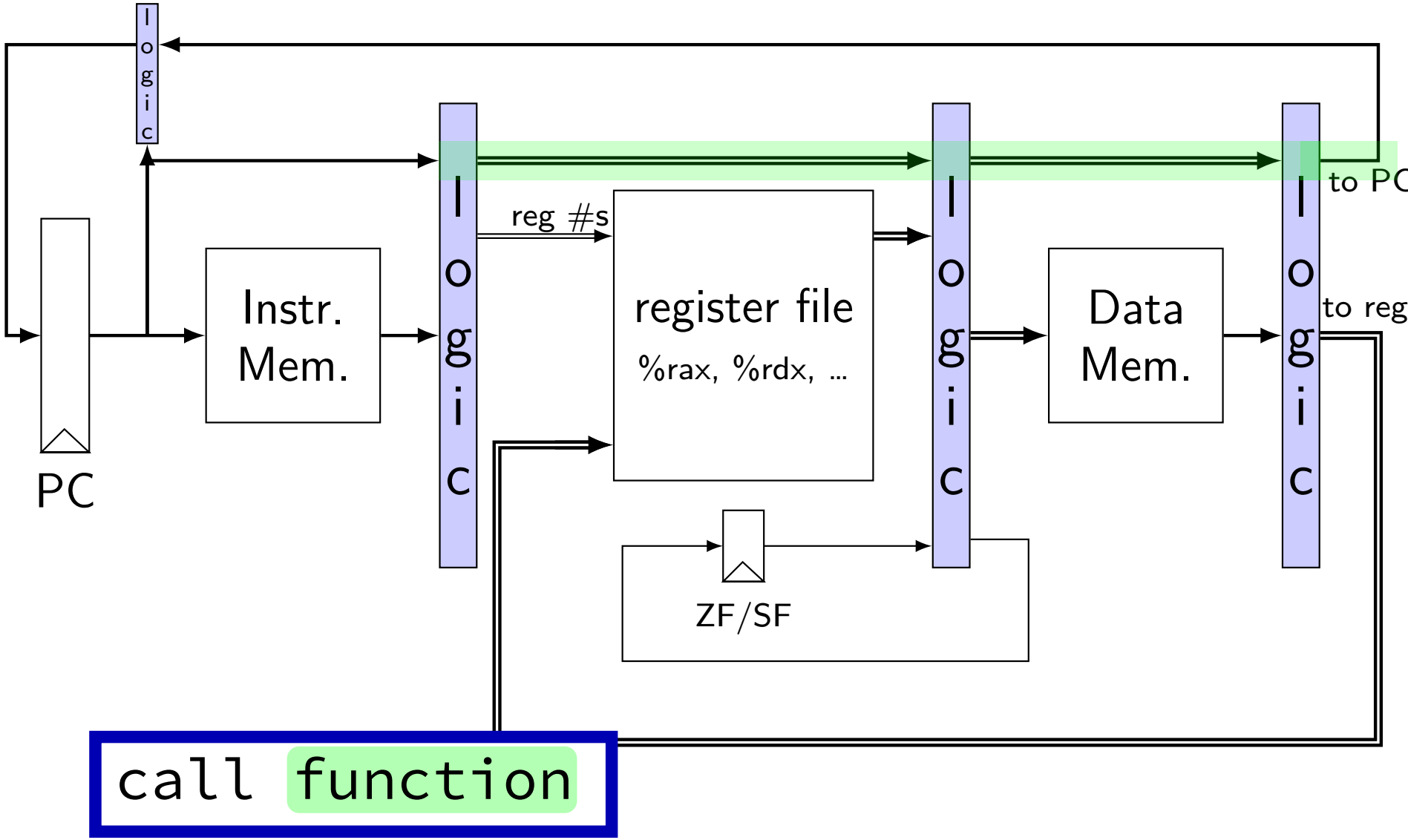

```
popq %rax
mrmovq 1000(%r9), %r8
```

Connections in Y86-64

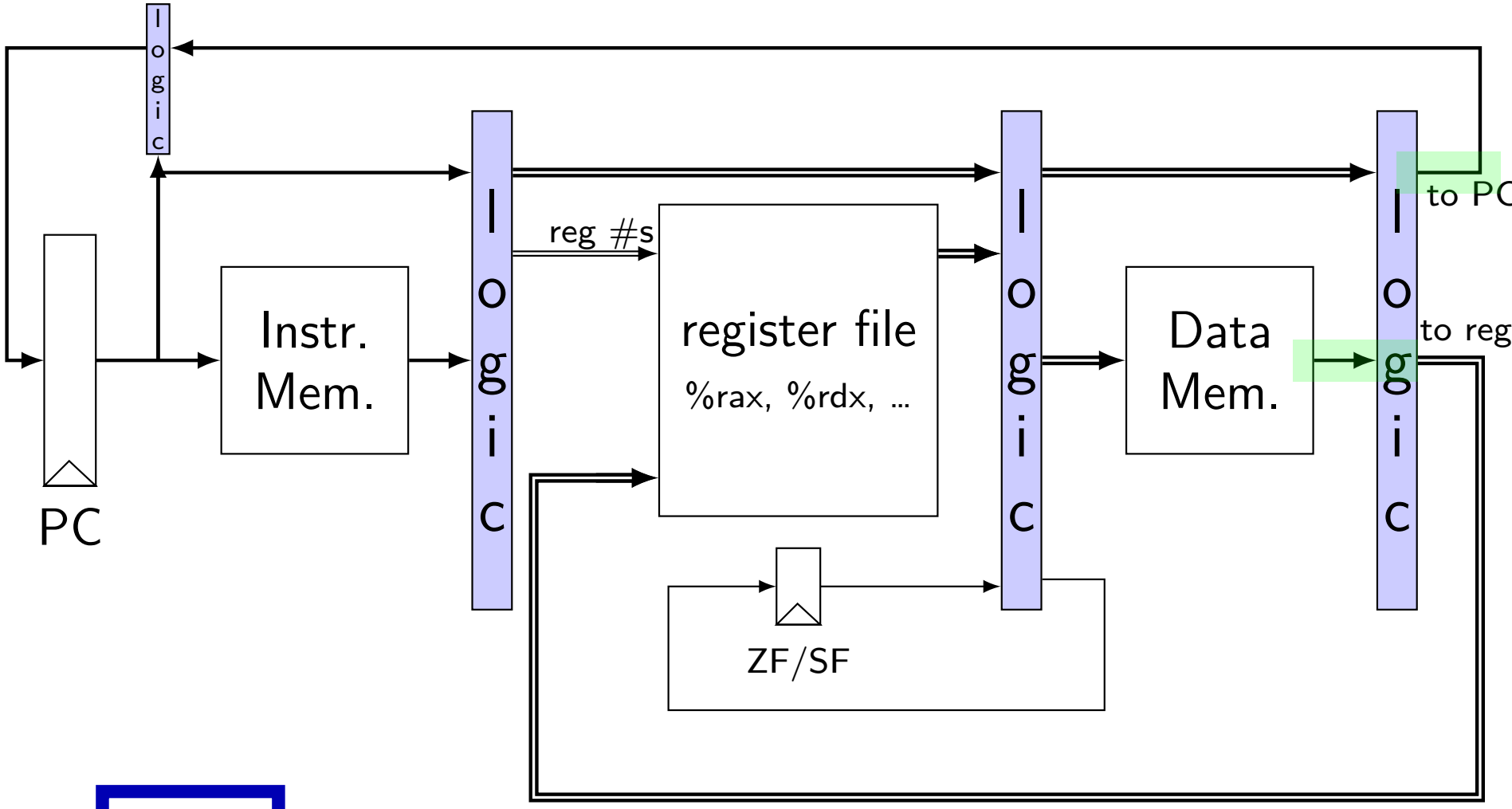


`popq %rax` (update `%rsp`)

Connections in Y86-64

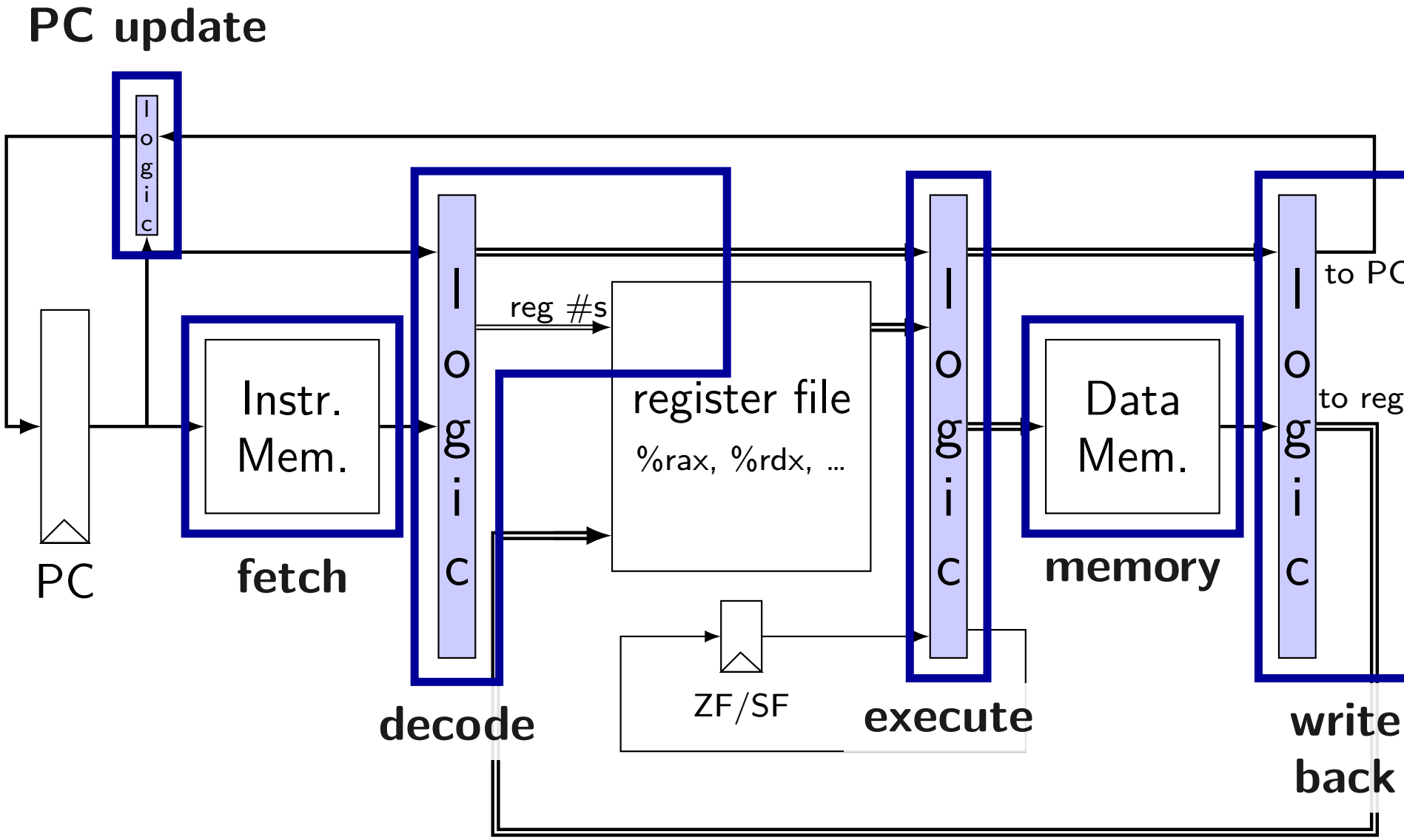


Connections in Y86-64



ret

Stages in Y86-64



Stages

fetch — read instruction memory, split instruction

decode — read register file

execute — arithmetic (including of addresses)

memory — read or write data memory

write back — write to register file

PC update — compute next value of PC

Stages and Time

fetch / decode / execute / memory / write back / PC update

For the design shown, order when these events happen `pushq %rax` instruction:

1. instruction read
2. memory changes
3. `%rsp` changes
4. PC changes

a. 1; then 2, 3, and 4 in any order
b. 1; then 2, 3, and 4 at almost the same time
c. 1; then 2; then 3; then 4
d. 1; then 3; then 2; then 4
e. 1; then 2; then 3 and 4 at almost the same time

Stages Example: nop

stage	nop
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 1$
decode	
memory	
write back	
PC update	PC $\leftarrow$ valP

Stages Example: nop

stage	nop
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$
	valP $\leftarrow$ PC $+ 1$
decode	
memory	
write back	
PC update	PC $\leftarrow$ valP

part of output wires
from instruction memory

Stages Example: nop

stage	nop
fetch	icode : ifun $\leftarrow$ M_1[PC]
	valP $\leftarrow$ PC + 1
decode	
memory	
write back	
PC update	PC $\leftarrow$ valP

name of a wire

$\leftarrow$ means putting a value on a wire

Stages Example: nop

stage	nop
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 1$
decode	
memory	
write back	
PC update	PC $\leftarrow$ valP

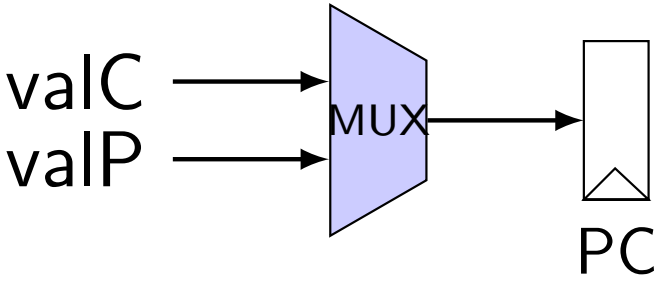
$\leftarrow$ means putting value on input wire to PC register

Stages Example: nop/jmp

stage	`nop`	`jmp dest`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow \text{PC} + 1$	icode : ifun $\leftarrow M_1[\text{PC}]$ valC $\leftarrow M_8[\text{PC} + 1]$
decode		
memory		
write back		
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valC

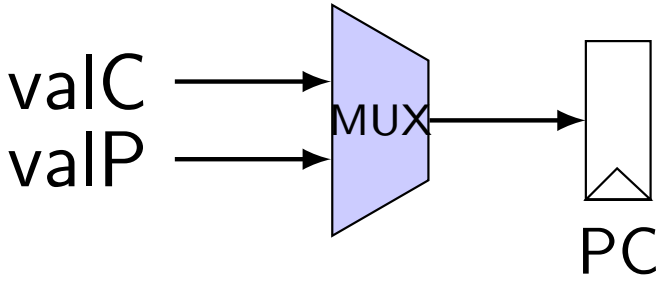
Stages Example: nop/jmp

stage	nop	jmp dest
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 1$	icode : ifun $\leftarrow M_1[\text{PC}]$ valC $\leftarrow M_8[\text{PC} + 1]$
decode		
memory		
write back		
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valC

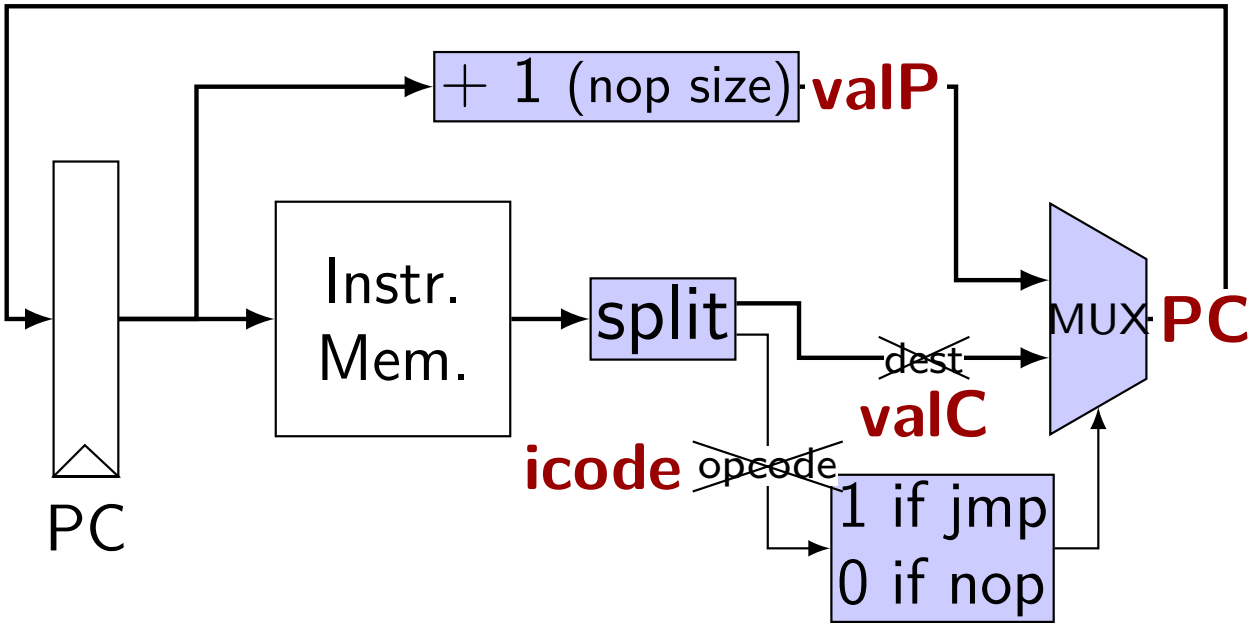


Stages Example: nop/jmp

stage	nop	jmp dest
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 1$	icode : ifun $\leftarrow M_1[\text{PC}]$ valC $\leftarrow M_8[\text{PC} + 1]$
decode		
memory		
write back		
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valC

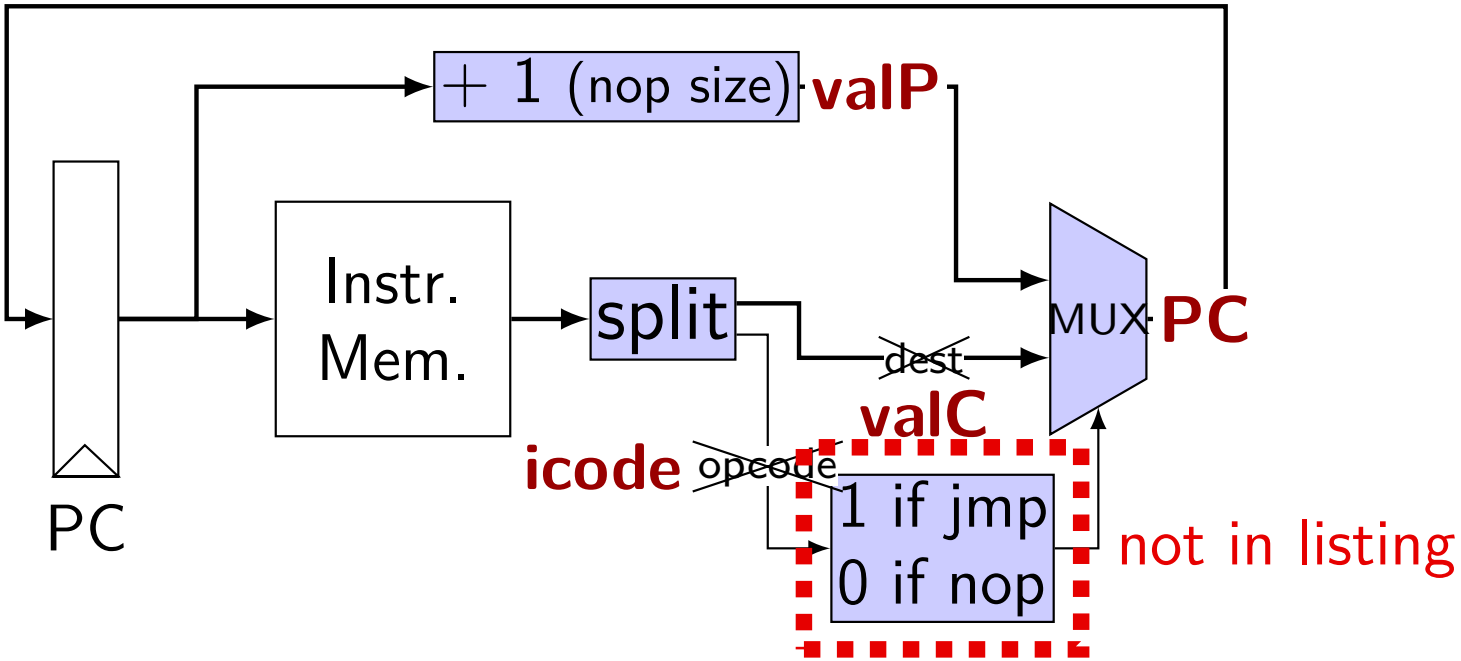


jmp+nop CPU



nop	1	0	
jmp *Dest*	7	0	*Dest*

jmp+nop CPU



nop	1	0	
jmp *Dest*	7	0	*Dest*

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA $\leftarrow R[\text{rA}]$ valB $\leftarrow R[\text{rB}]$	valB $\leftarrow R[\text{rB}]$
execute	valE $\leftarrow$ valB $+$ valC	valE $\leftarrow$ valB $+$ valC
memory	$M_8[\text{valE}] \leftarrow$ valA	valM $\leftarrow M_8[\text{valE}]$
write back		$R[\text{rA}] \leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA $\leftarrow R[\text{rA}]$ valB $\leftarrow R[\text{rB}]$	valB $\leftarrow R[\text{rB}]$
ex		valC
m		lE]
write back		$R[\text{rA}] \leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

assignment means:
setting register number input register file *and* naming output wires of register file

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA $\leftarrow R[\text{rA}]$ valB $\leftarrow R[\text{rB}]$	~~valB~~ $\leftarrow R[\text{rB}]$
execute	valE $\leftarrow$	$+$ valC
memory	$M_8[\text{valE}] \leftarrow$ valA	valM $\leftarrow M_8[\text{valE}]$
write back		$R[\text{rA}] \leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

reading $R[\text{rA}]$ not needed but would be harmless

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA $\leftarrow R[\text{rA}]$ valB $\leftarrow R[\text{rB}]$	valB $\leftarrow R[\text{rB}]$
execute	valE $\leftarrow$ valB $+$ valC	valE $\leftarrow$ valB $+$ valC
memory	$M_8[\text{valE}] \leftarrow$ valA	valM $\leftarrow M_8[\text{valE}]$

assignment means:
setting address wires to valE *and*
setting memory input wires to valA *and*

rA] $\leftarrow$ valM

$\leftarrow$ valP

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA valB	
execute	valE	
memory	$M_8[\text{valE}] \leftarrow$ valA	valM $\leftarrow M_8[\text{valE}]$
write back		$R[\text{rA}] \leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

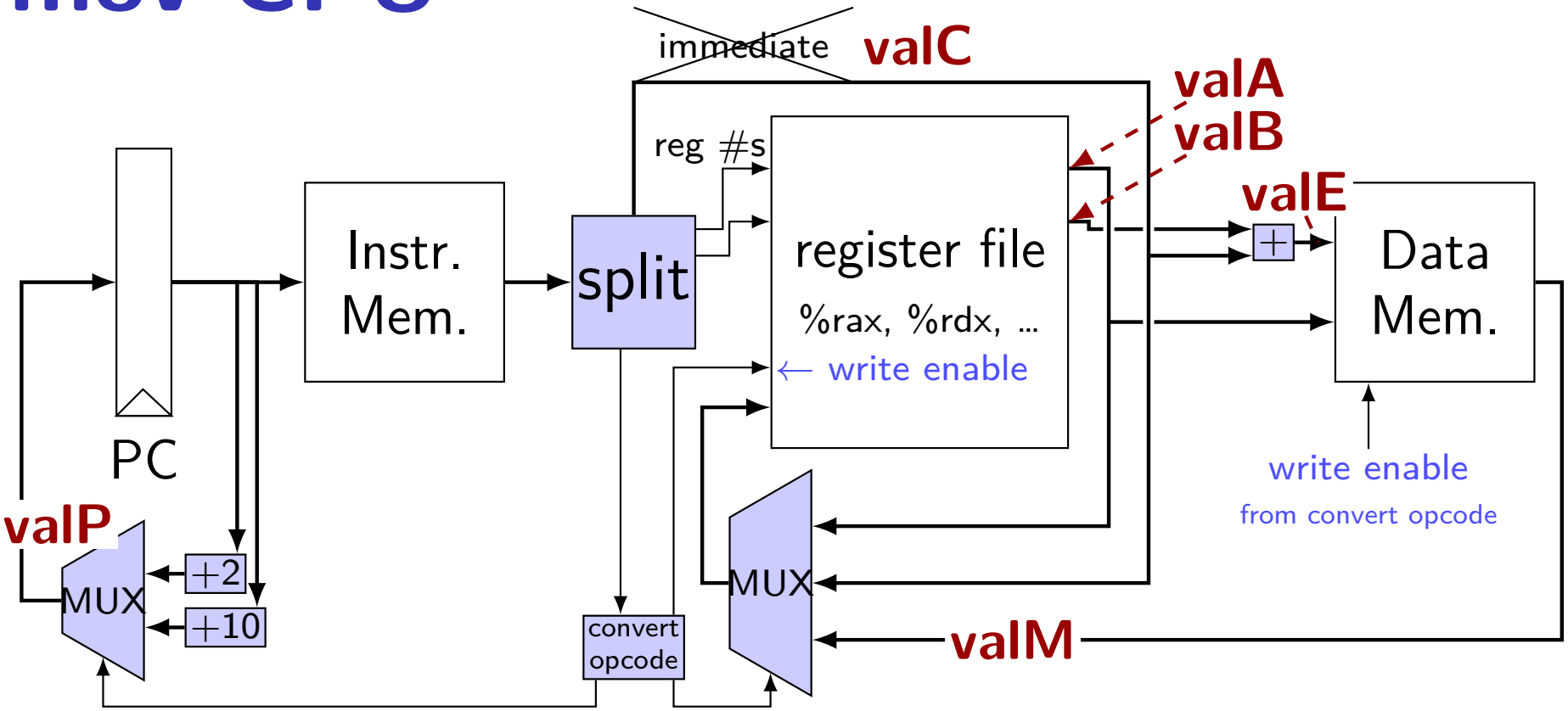
assignment means:
setting address wires to valE *and*
naming the output of the data memory

Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$	icode : ifun $\leftarrow M_1[\text{PC}]$ valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8[\text{PC} + 2]$
decode	valA $\leftarrow R[\text{rA}]$ valB $\leftarrow R[\text{rB}]$	valB $\leftarrow R[\text{rB}]$
execute	valE	
memory	M_8	
write back		$R[\text{rA}] \leftarrow$ valM
PC update	PC $\leftarrow$ valP	PC $\leftarrow$ valP

assignment means:
setting register file input wires to valM
setting register file write enable to true

mov CPU



rrmovq *rA*, *rB*	2	0	*rA*	*rB*	
irmovq *V*, *rB*	3	0	F	*rB*	*V*
mrmovq *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*
rmmovq *rA*, *D*(*rB*)	4	0	*rA*	*rB*	*D*