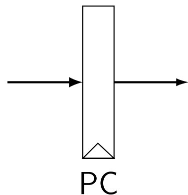
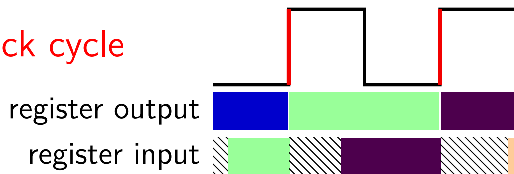
SEQUENTIAL Part 1

Samira Khan
The slides are prepared by Charles Reiss

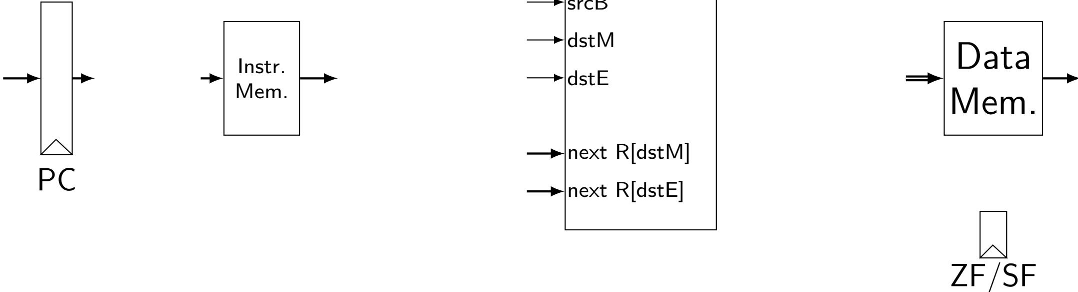
registers



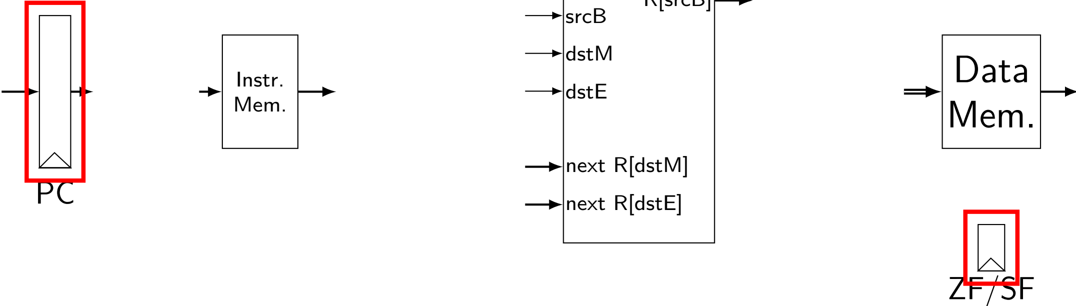
updates every clock cycle



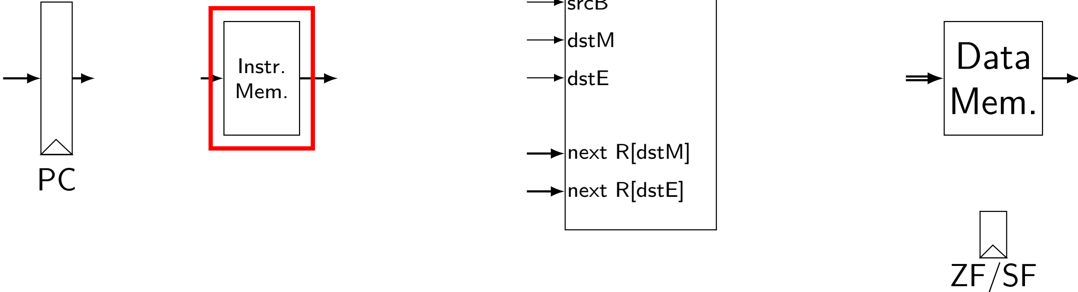
state in Y86-64



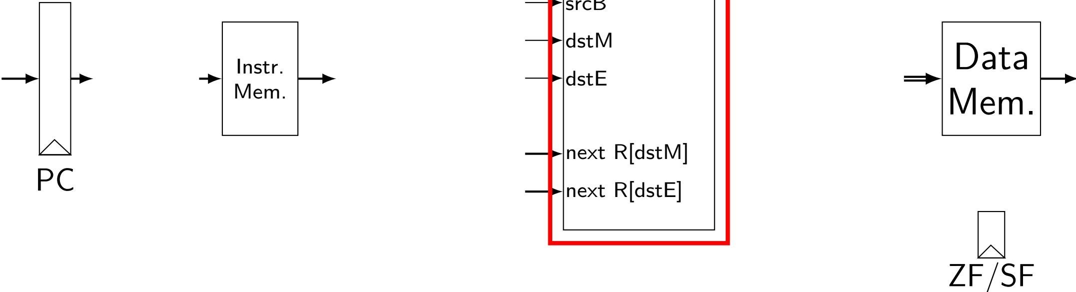
state in Y86-64



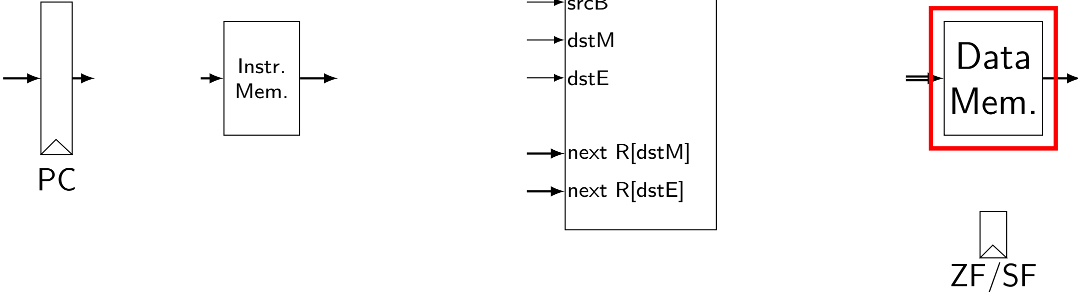
state in Y86-64



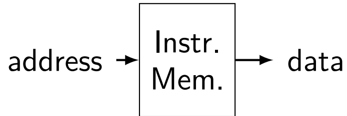
state in Y86-64

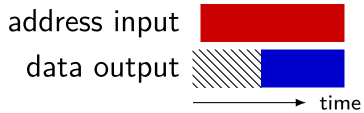


state in Y86-64

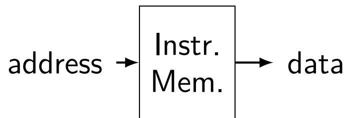


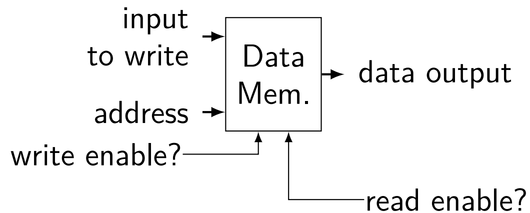
memories

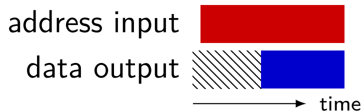




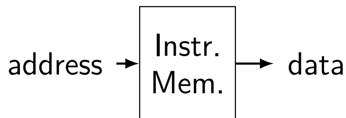
memories

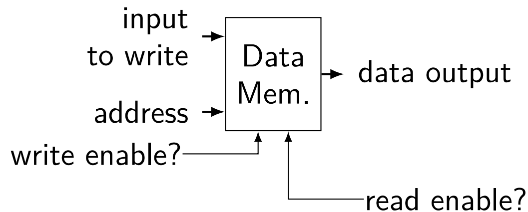


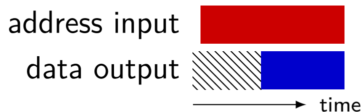


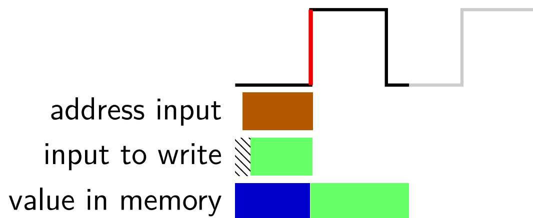


memories

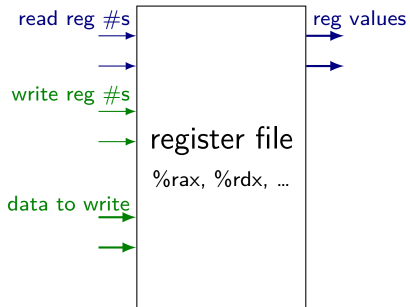




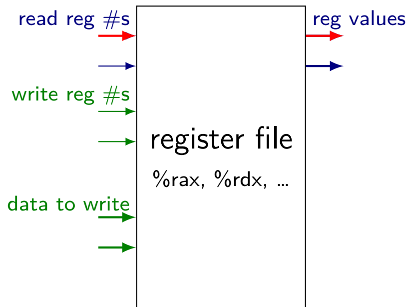


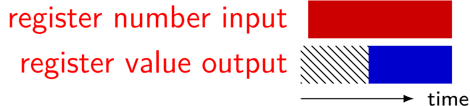


register file

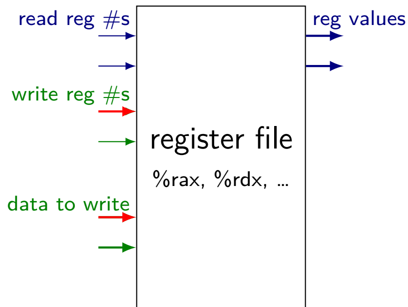


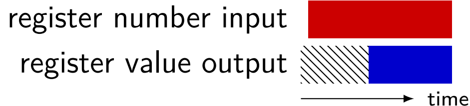
register file

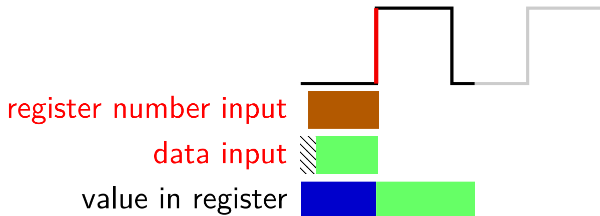




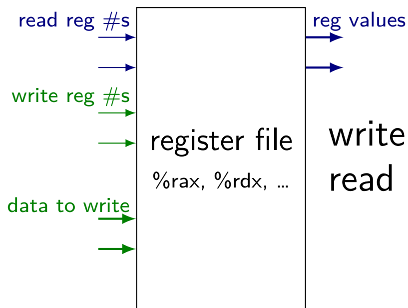
register file



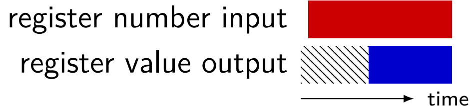


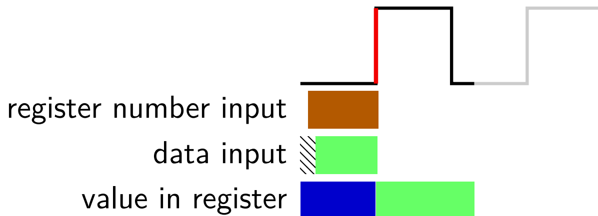


register file

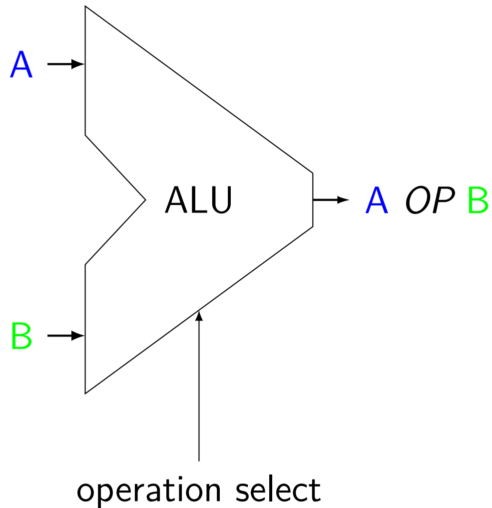


write register #15: write is ignored
read register #15: value is always 0





ALUs



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

simple ISA 1: addq

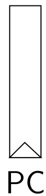
`addq %rXX, %rYY`

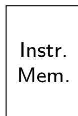
encoding: `%rXX` `%rYY` (two 4-bit register #s)

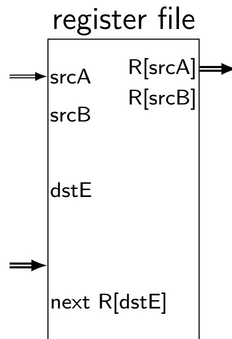
1 byte instructions, no opcode

no other instructions

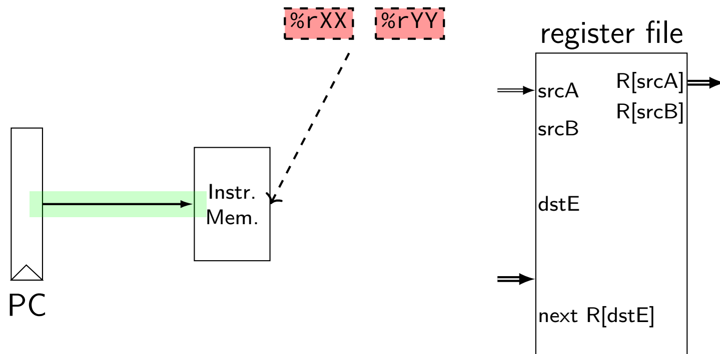
addq CPU



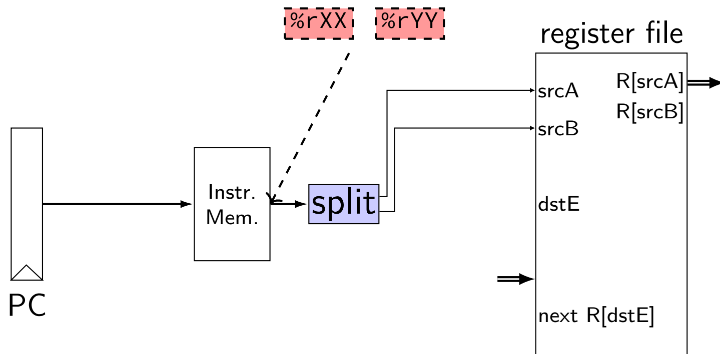




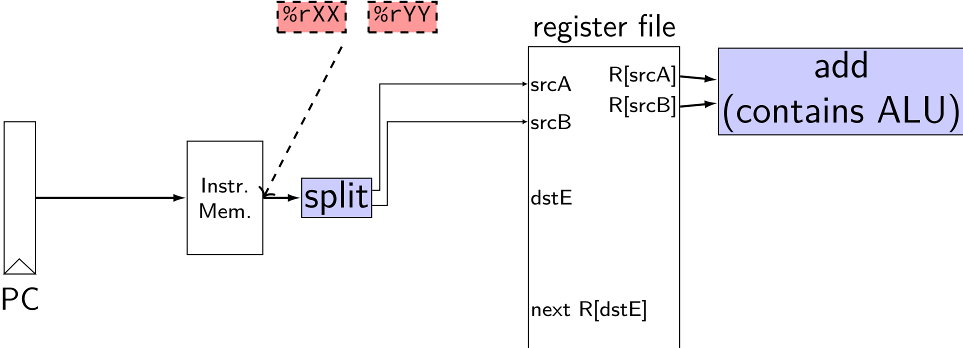
addq CPU



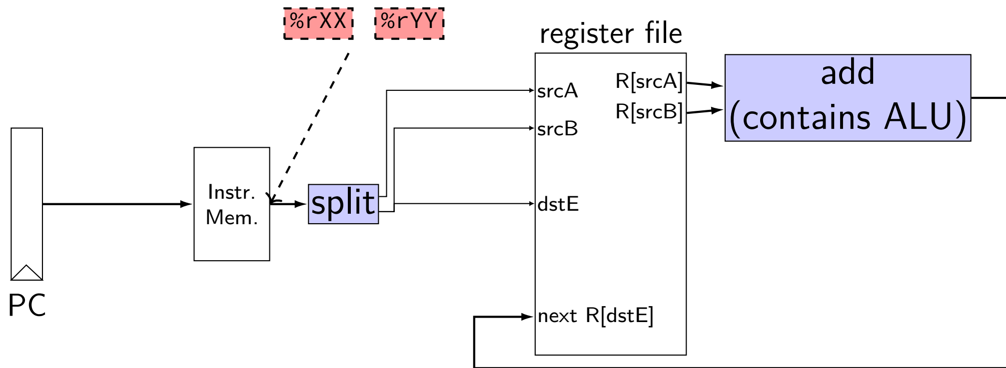
addq CPU



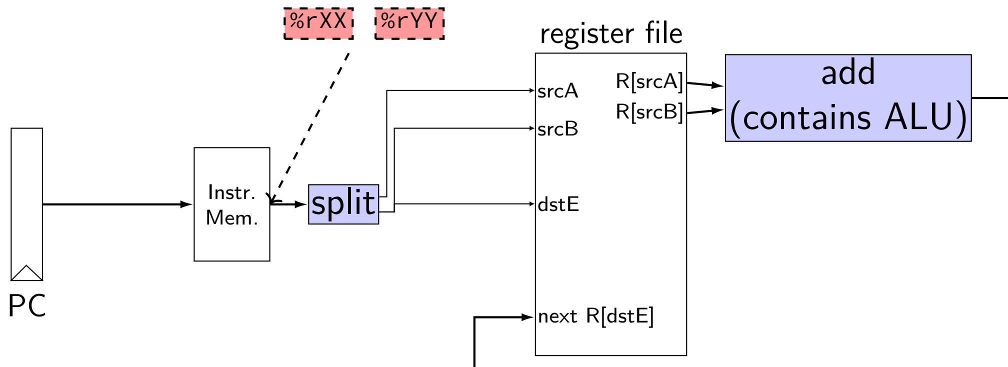
addq CPU



addq CPU



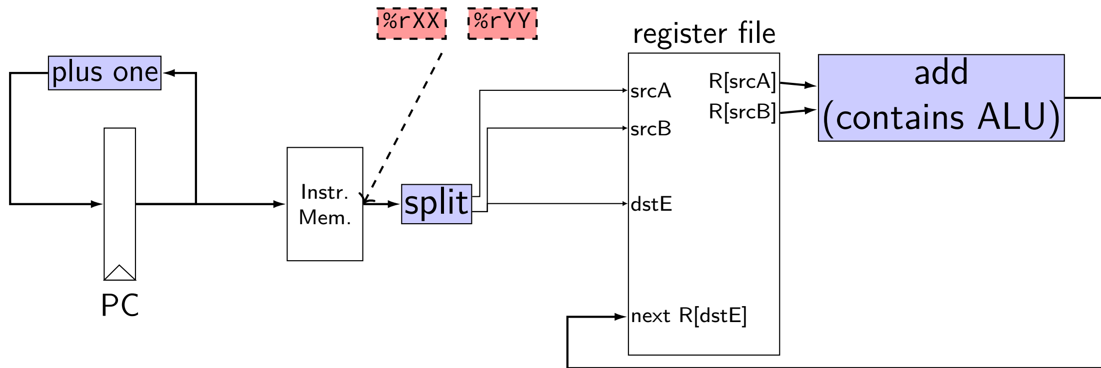
addq CPU



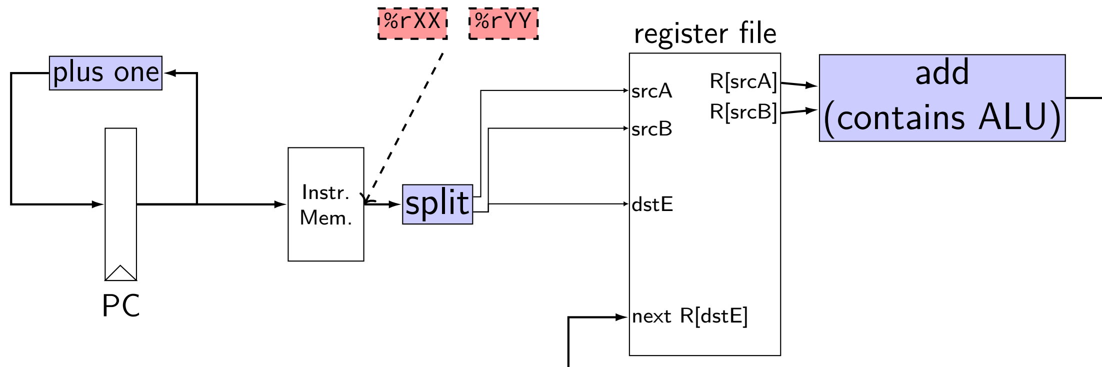

```
/* 0x00: */ addq %rax, %rdx
/* 0x01: */ addq %rbx, %rdx
```

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



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

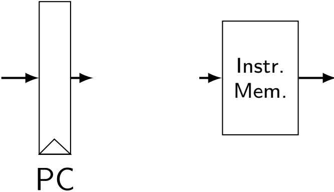
Simple ISA 2: jmp

```
jmp label
```

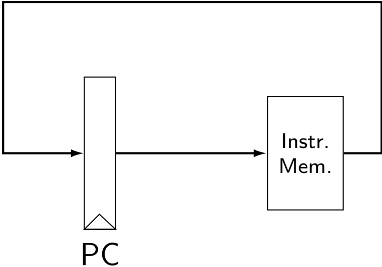
encoding: *8-byte little-endian address*

  8 byte instructions, no opcode

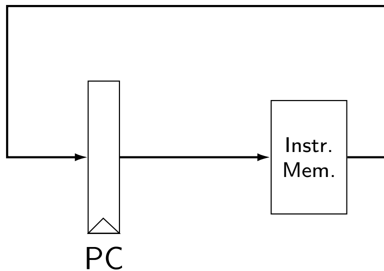
jmp CPU



jmp CPU



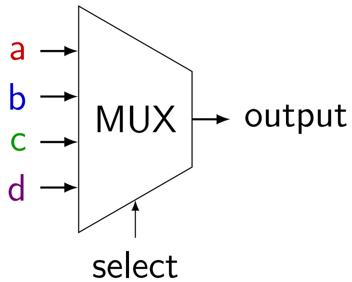
jmp CPU



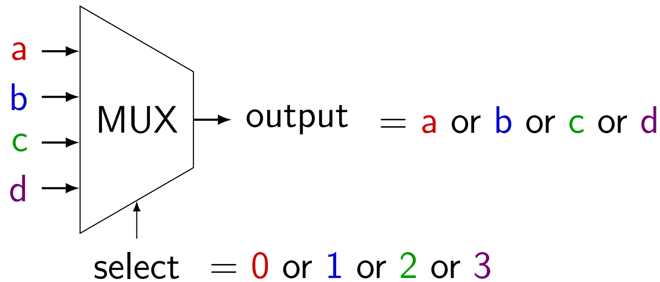

```
/* 0x00: */ jmp 0x10
/* 0x08: */ jmp 0x00
/* 0x10: */ jmp 0x08
```

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

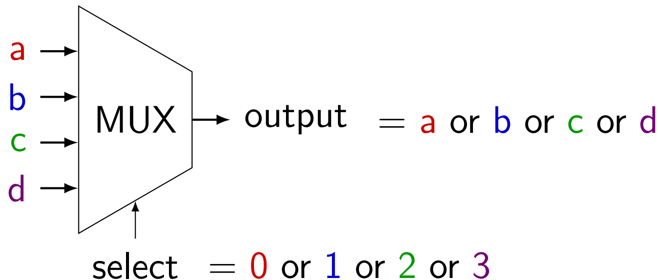
multiplexers



multiplexers



multiplexers



truth table:

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

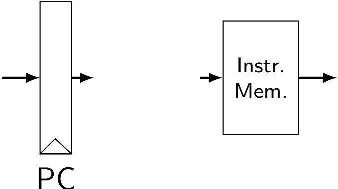
Simple ISA 3: Jmp or No-Op

actual subset of Y86-64

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

`nop` — encoded as `0x10`

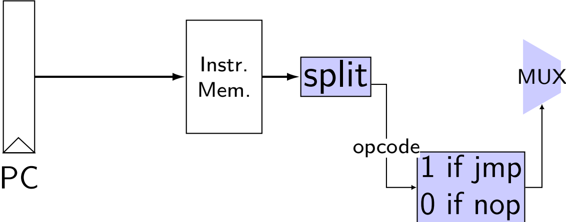
jmp+nop CPU



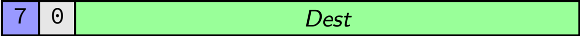
`nop`

`jmp` *Dest*

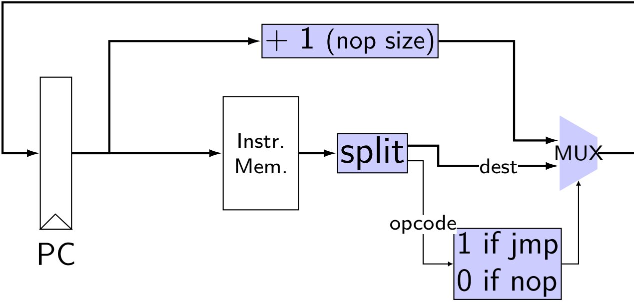
jmp+nop CPU

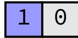


nop

jmp *Dest* 

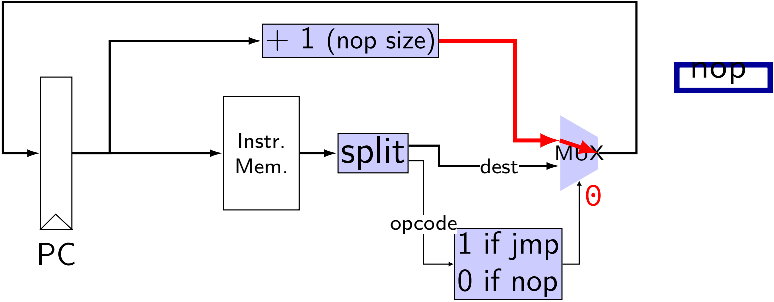
jmp+nop CPU



nop 

jmp *Dest*

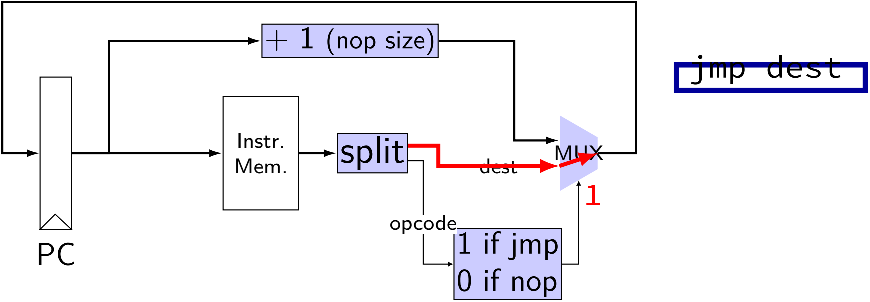
jmp+nop CPU



nop

jmp *Dest*

jmp+nop CPU



nop

jmp *Dest*

exercise: nop/add CPU

Let's say we wanted to make nop+add CPU. Where would need MUXes?

A. before one or both of the register file 'register number to read' inputs

B. before the PC register's input

C. before one of the register file 'register number to write' inputs

D. before one of the register file 'register value to write' inputs

E. before the instruction memory's address input

Summary

each instruction takes one cycle

divided into stages for design convenience

read values from previous cycle

send new values to state components

control what is sent with MUXes

simple ISA 4: mov-to-register

```
irmovq $constant, %rYY

rrmovq %rXX, %rYY

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

mov-to-register CPU

register file

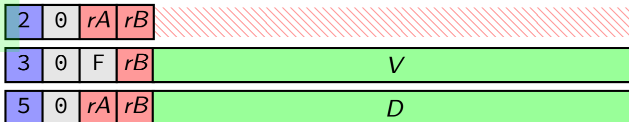
srcA
srcB
dstE
next R[dstE]
R[srcA]
R[srcB]

Instr.
Mem.

PC

Data in
Data out
Addr in

rrmovq *rA*, *rB*
irmovq *V*, *rB*
mrmovq *D*(*rB*), *rA*



mov-to-register CPU

register file

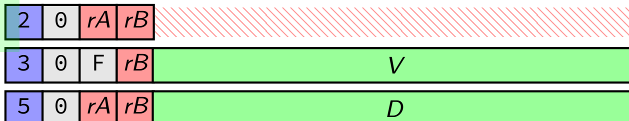
srcA R[srcA]
srcB R[srcB]
dstE
next R[dstE]

PC

Instr. Mem.

split

Data in
Data out
Addr in



mov-to-register CPU

register file

srcA
srcB
dstE
R[srcA]
R[srcB]
next R[dstE]

PC

Instr.
Mem.

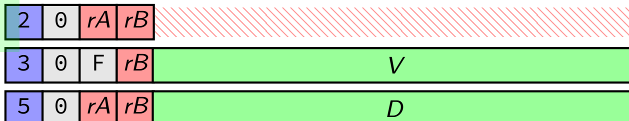
split

Data in
Data out
Addr in

rrmovq *rA, rB*

irmovq *V, rB*

mrmovq *D(rB), rA*



mov-to-register CPU

register file

srcA

srcB

dstE

next R[dstE]

R[srcA]

R[srcB]

PC

Instr. Mem.

split

Data in

Data out

Addr in

`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

register file

srcA
srcB
dstE
next R[dstE]
R[srcA]
R[srcB]

PC

Instr.
Mem.

split

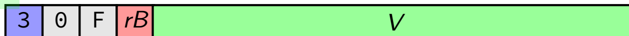
convert
opcode

MUX

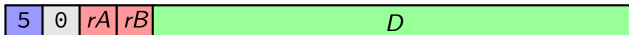
Data in
Data out
Addr in

rrmovq *rA*, *rB*

irmovq *V*, *rB*



mrmovq *D*(*rB*), *rA*



mov-to-register CPU

immediate

register file

srcA
srcB
dstE
next R[dstE]
R[srcA]
R[srcB]

PC

Instr.
Mem.

split

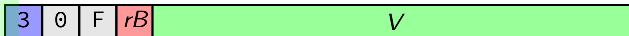
convert
opcode

MUX

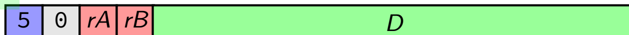
Data in
Data out
Addr in

`rrmovq` *rA*, *rB*

`irmovq` *V*, *rB*



`mrmovq` *D*(*rB*), *rA*



mov-to-register CPU

immediate

register file

srcA
srcB
dstE
next R[dstE]
R[srcA]
R[srcB]

PC

Instr.
Mem.

split

convert
opcode

MUX

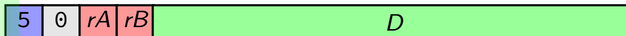
+ (ALU)

Data in
Data out
Addr in

rrmovq *rA*, *rB*

irmovq *V*, *rB*

mrmovq *D(rB)*, *rA*



mov-to-register CPU

immediate

register file

srcA

srcB

dstE

next R[dstE]

R[srcA]

R[srcB]

PC

Instr.
Mem.

split

convert
opcode

+2

+10

MUX

+ (ALU)

Data in
Data out
Addr in

`rrmovq` *rA*, *rB*

`irmovq` *V*, *rB*

`mrmovq` *D*(*rB*), *rA*