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

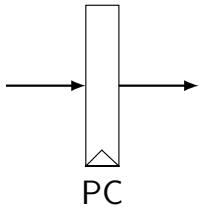
program counter (register)

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

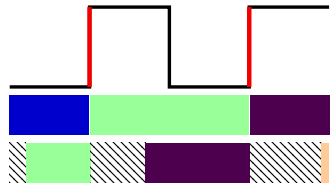
condition codes (ZF, SF)

status register (is the processor still running?)

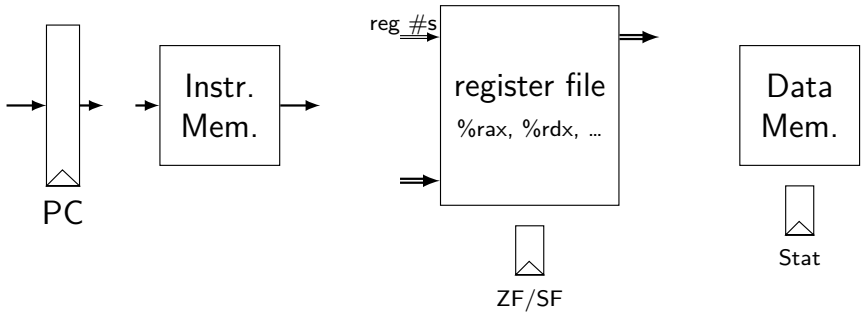
Registers



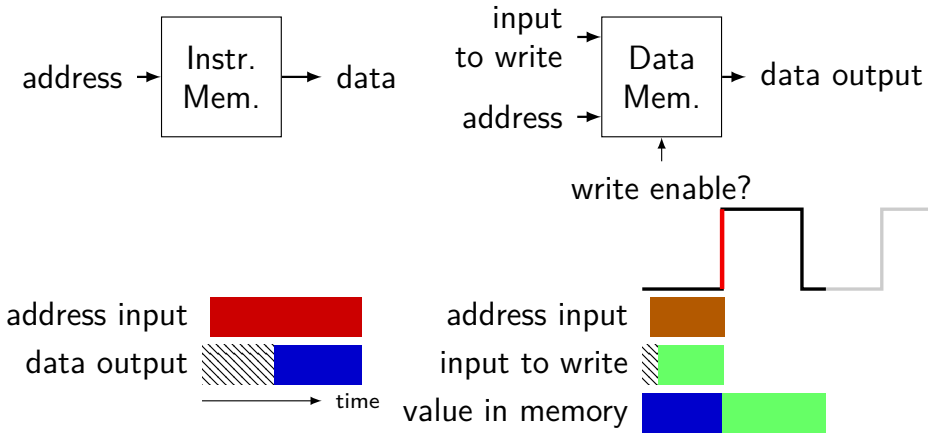
updates every clock cycle



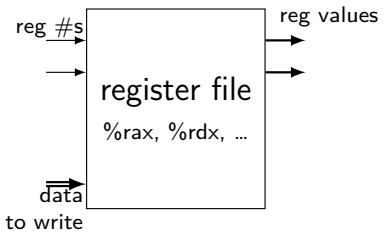
State in Y86-64



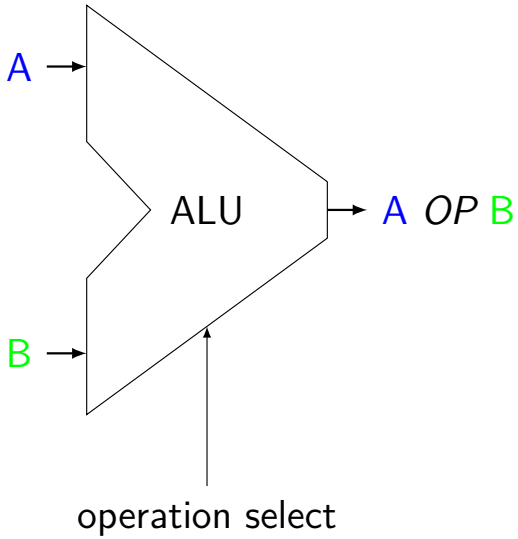
Memories



Register file



ALUs



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

Simple ISA 1: addq

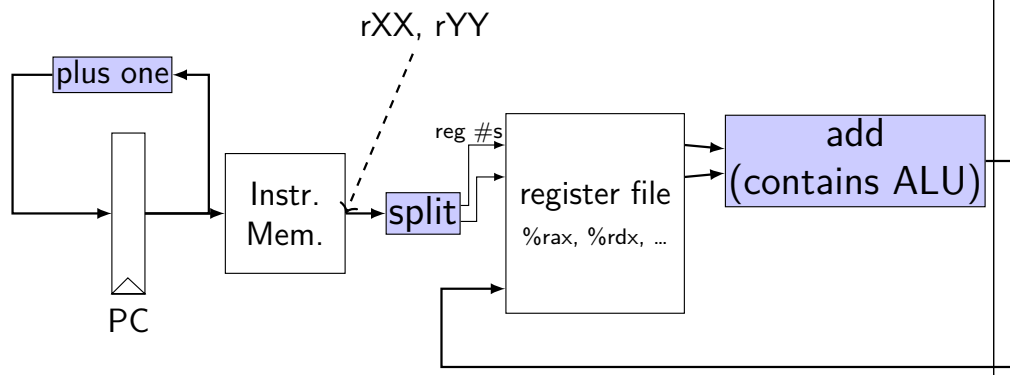
```
addq %rXX, %rYY
```

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

1 byte instructions, no opcode

no other instructions

addq CPU




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

initially: PC = 0x00, rax = 1, rbx = 2, rdx = ~~3~~ 3

after cycle 1: PC = 0x01, rax = 1, rbx = 2, rdx = ~~4~~ 4

after cycle 2: PC = 0x02, rax = ??, rbx = ??, rdx = ~~6~~ ??

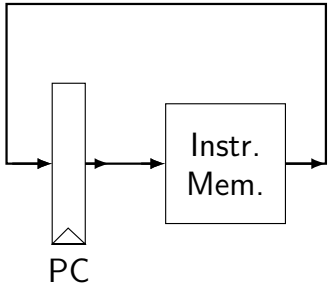
Simple ISA 2: jmp

```
jmp label
```

encoding: *8-byte little-endian address*

8 byte instructions, no opcode

jmp CPU




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

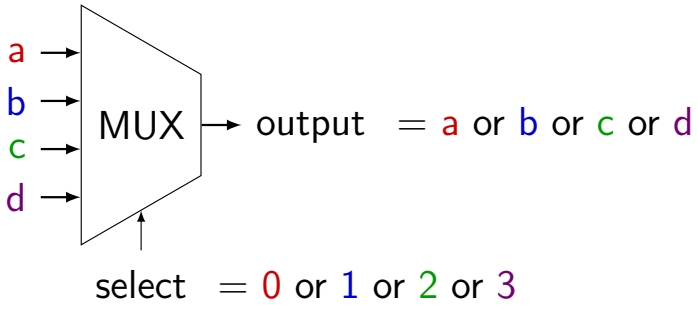
initially: PC = 0x00

after cycle 1: PC = 0x10

after cycle 2: PC = 0x08

after cycle 3: PC = 0x00

Multiplexers



truth table:

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

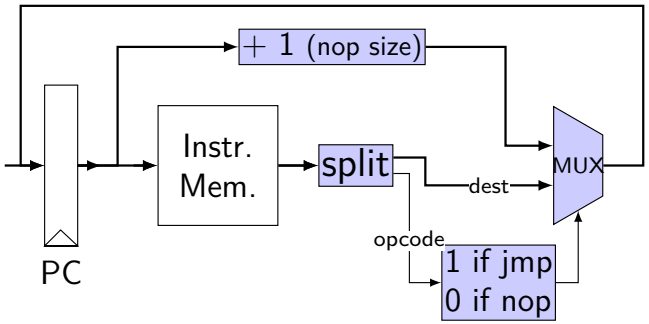
Simple ISA 3: Jmp or No-Op

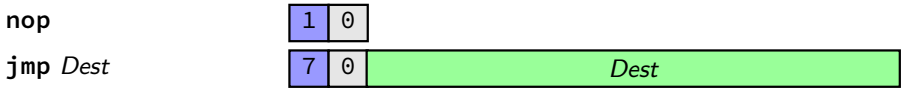
actual subset of Y86-64

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

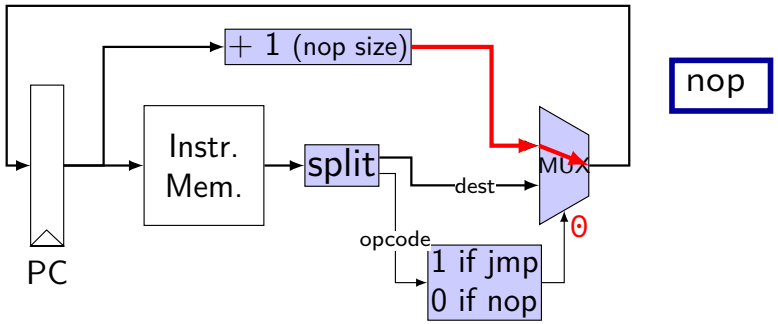
nop — encoded as `0x10`

jmp+nop CPU

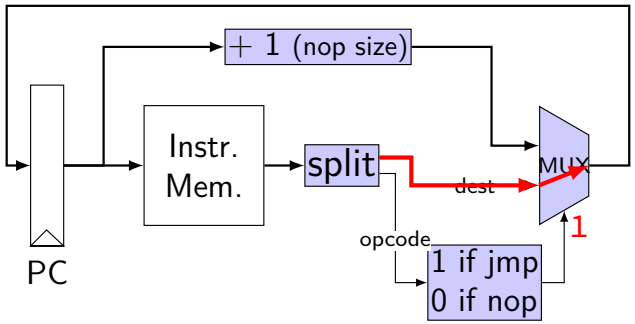




jmp+nop CPU



jmp+nop CPU

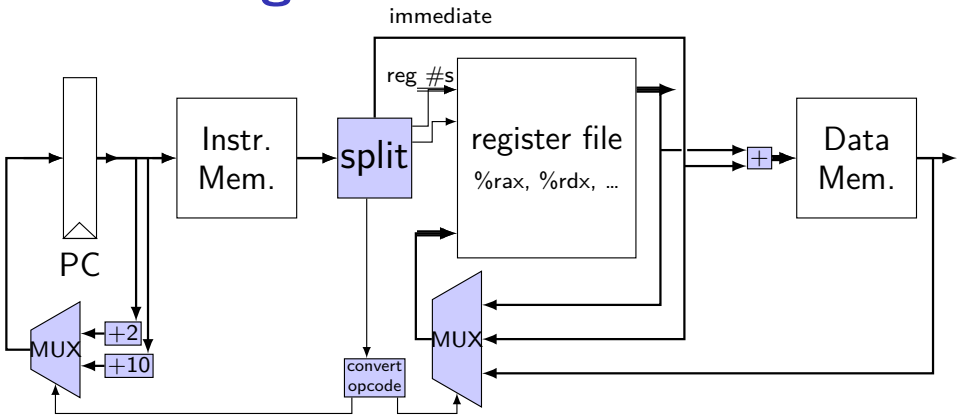



```
jmp dest
```


Simple ISA 4: Mov-to-register

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


mov-to-register CPU

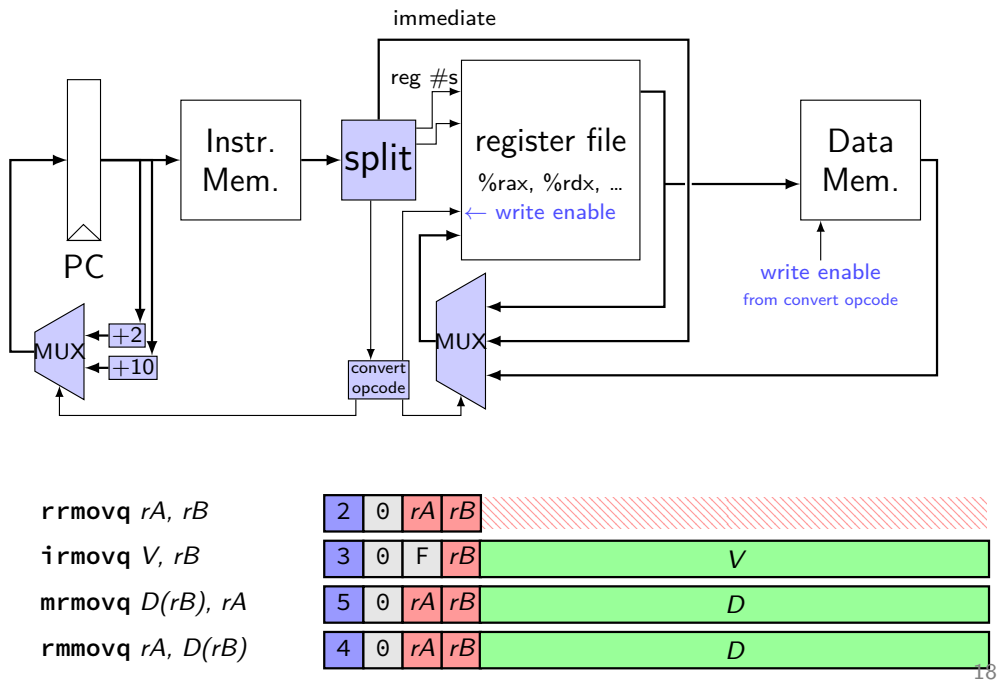


rrmovq *rA, rB*	2	0	*rA*	*rB*	
irmovq *V, rB*	3	0	F	*rB*	*V*
mrmovq *D(rB), rA*	5	0	*rA*	*rB*	*D*

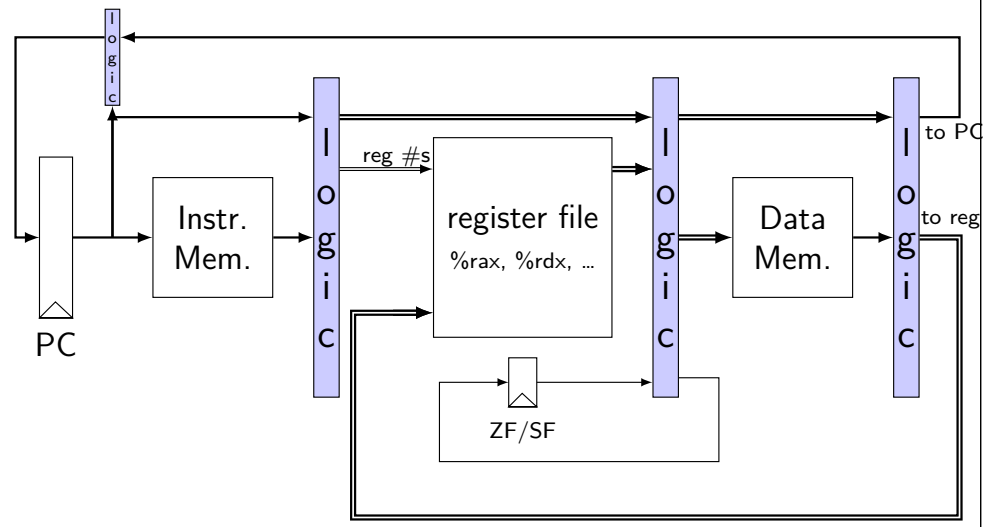
Simple ISA 4B: Mov

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

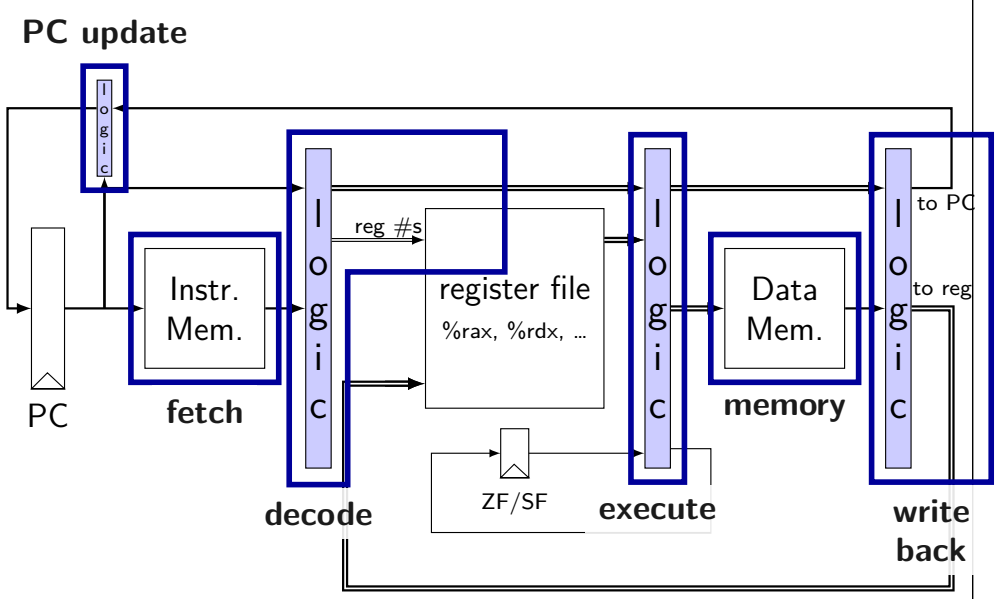
mov CPU



Connections in Y86-64



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

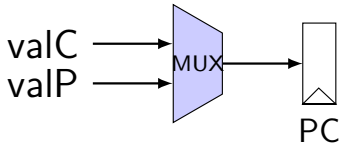
a. 1; then 2, 3, and 4 in any order
b. 1; then 2, 3, and 4 at almost the same time
c. 1; then 2; then 3; then 4
d. 1; then 3; then 2; then 4
e. 1; then 2; then 3 and 4 at almost the same time
f. something else

Stages Example: nop

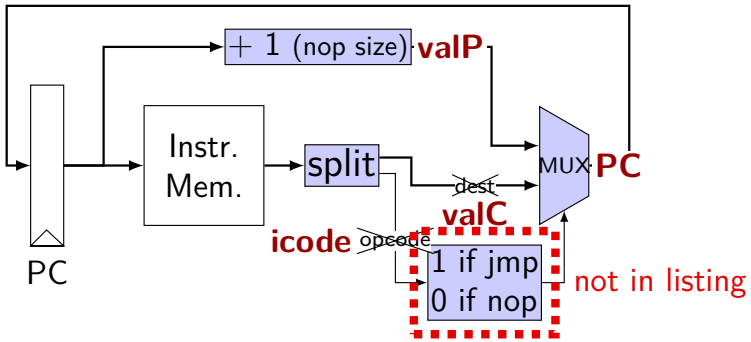
stage	nop
fetch	icode : ifun $\leftarrow M_1$[PC] valP $\leftarrow$ PC $+ 1$
decode	
memory	
write back	
PC update	PC $\leftarrow$ valP

Stages Example: nop/jmp

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



jmp+nop CPU



Stages Example: rmmovq/mrmovq

stage	`rmmovq rA, D(rB)`	`mrmovq D(rB), rA`
fetch	icode : ifun $\leftarrow M_1$[PC] valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8$[PC $+ 2$]	icode : ifun $\leftarrow M_1$[PC] valP $\leftarrow$ PC $+ 10$ valC $\leftarrow M_8$[PC $+ 2$]
decode	val	
ex		
m		rA] $\leftarrow$ valM
		$\leftarrow$ valP

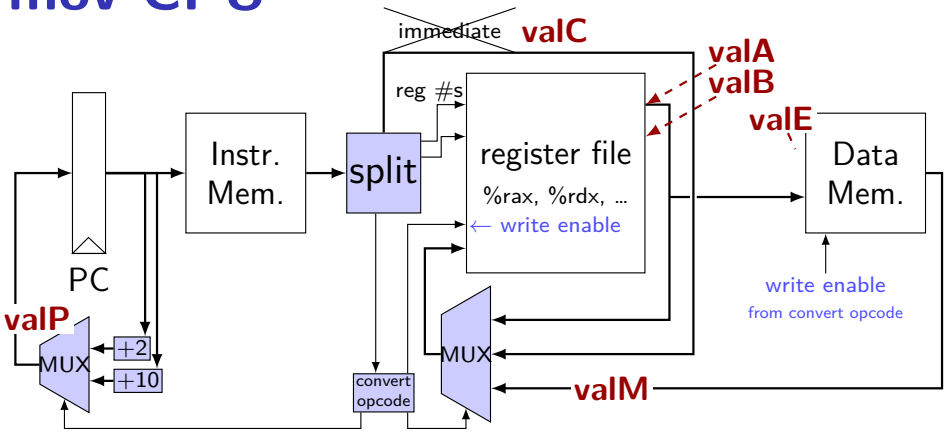
assignment means:
setting address wires to valE *and*
setting memory input wires to valA *and*
setting memory write enable to 1

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

assignment me
setting register
naming output

assignment means:
setting address wires to valE *and*

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*