last time

timing on our processor

- changing stored values: triggered by rising clock edge
- other things: computed as inputs change
- current design: one instruction per cycle

data memory / write enable

implementing movs, etc.

"stages"

- fetch / decode [reg read] / execute [ALU] / memory / write back [reg write]

quiz Q1

adding incq: what encoding?

want: distinct, so we won't confuse it with others

- `0x60, 0x1[rX]` — already used for `addq %rcx, rX`
- `[rX]0xF` — if rX = %rcx, shares icode with nop (technically not alreay used, but ...)

want: fields in same place as other instructions to simplify logic

- `[rX]0xF` and `0xE[rX]` — rX not in second byte like other instructions

quiz Q2

adding incq: new MUX to control?

address input to instruction memory — no, still from PC register out

register value input — no, still comes from ALU

- add MUX controlling ALU inputs instead

quiz Q3

rjmq %r?? — machine code format

- one field for register index
- icode, but different than others

most similar to popq — just change icode

others required changing length/removing fields/etc.

quiz Q4

rjmq %r?? — PC register input

needs to be address of next instruction

= address stored in %r??

$\neq$ index of %r?? in register file

address stored in %r?? will be output from register file

quiz Q5

rjmq %r??

need to read register %r??

so, need to pass register index from instruction to register file

SEQ: instruction fetch

read instruction memory at PC

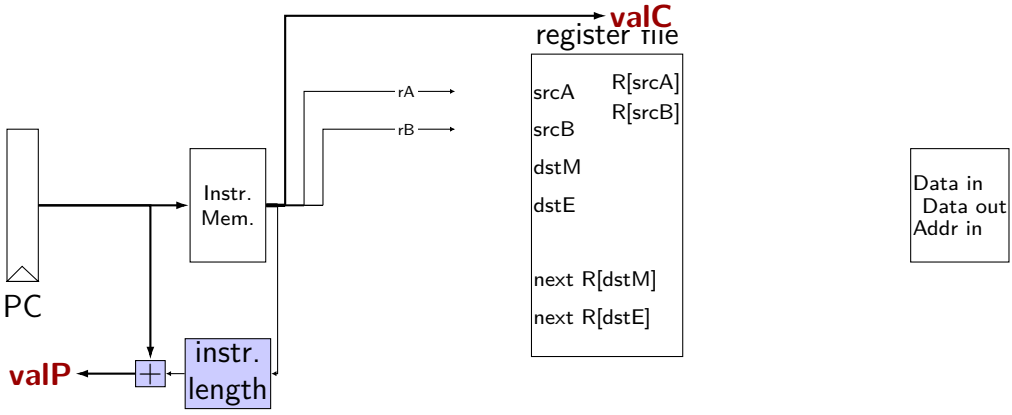
split into seperate wires:

- icode:ifun — opcode
- rA, rB — register numbers
- valC — call target or mov displacement

compute next instruction address:

- valP — PC + (instr length)

instruction fetch

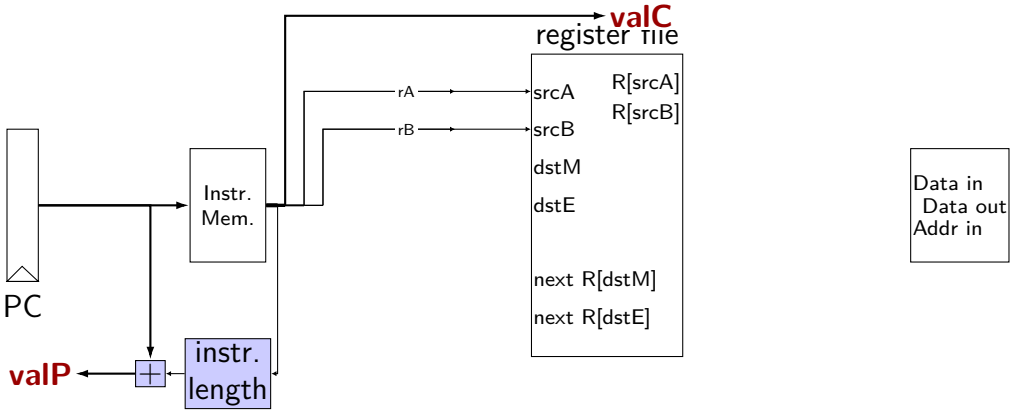


SEQ: instruction “decode”

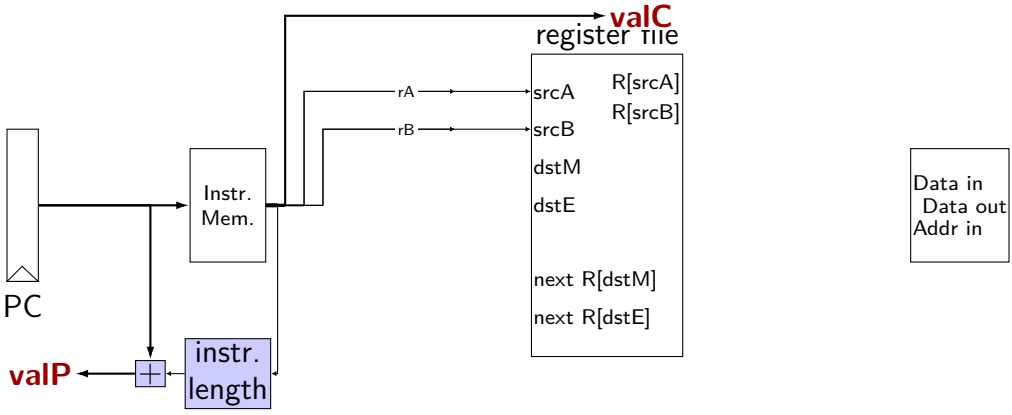
read registers

valA, valB — register values

instruction decode (1)

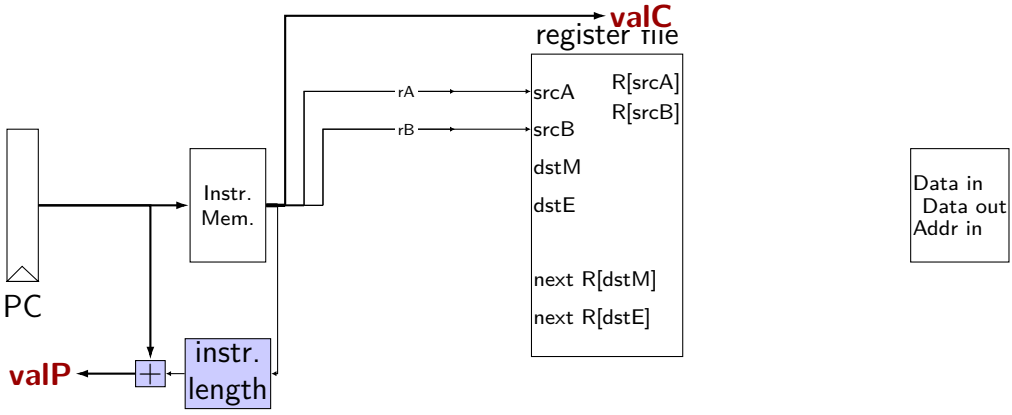


instruction decode (1)



exercise: for which instructions would there be a problem ?
nop, addq, `mrmovq`, `rmmovq`, `jmp`, pushq

instruction decode (1)



SEQ: srcA, srcB

always read rA, rB?

Problems:

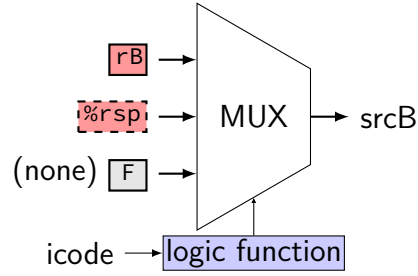
- push rA
- pop
- call
- ret

book: extra signals: srcA, srcB — computed input register

MUX controlled by icode

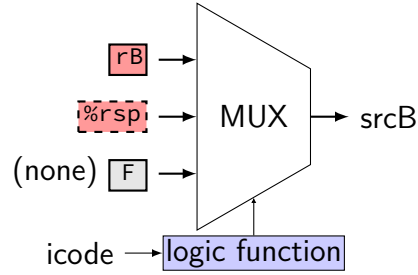
SEQ: possible registers to read

instruction	srcA	srcB
`halt`, `nop`, `j`*CC*, `irmovq`	none	none
`cmovCC`, `rrmovq`	rA	none
`mrmovq`	none	rB
`rmmovq`, *OP*`q`	rA	rB
`call`, `ret`	none?	`%rsp`
`pushq`, `popq`	rA	`%rsp`



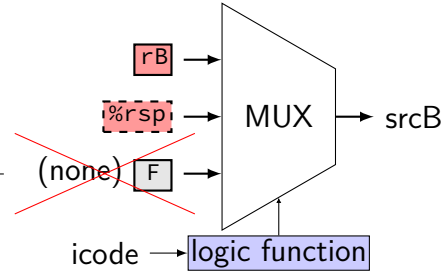
SEQ: possible registers to read

instruction	srcA	srcB
`halt`, `nop`, `j`*CC*, `irmovq`	none	none
`cmovCC`, `rrmovq`	rA	none
`mrmovq`	none	rB
`rmmovq`, *OP*`q`	rA	rB
`call`, `ret`	none?	`%rsp`
`pushq`, `popq`	rA	`%rsp`

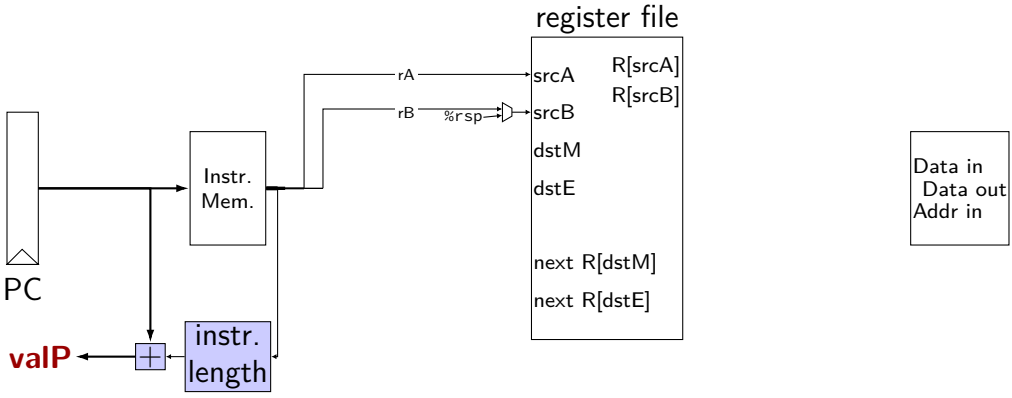


SEQ: possible registers to read

instruction	srcA	srcB
`halt`, `nop`, `j`*CC*, `irmovq`	none	none
`cmovCC`, `rrmovq`	rA	none
`mrmovq`	none	rB
`rmmovq`, *OP*`q`	rA	rB
`call`, `ret`	none?	`%rsp`
`pushq`, `popq`	rA	`%rsp`



instruction decode (2)



SEQ: execute

perform ALU operation (add, sub, xor, and)

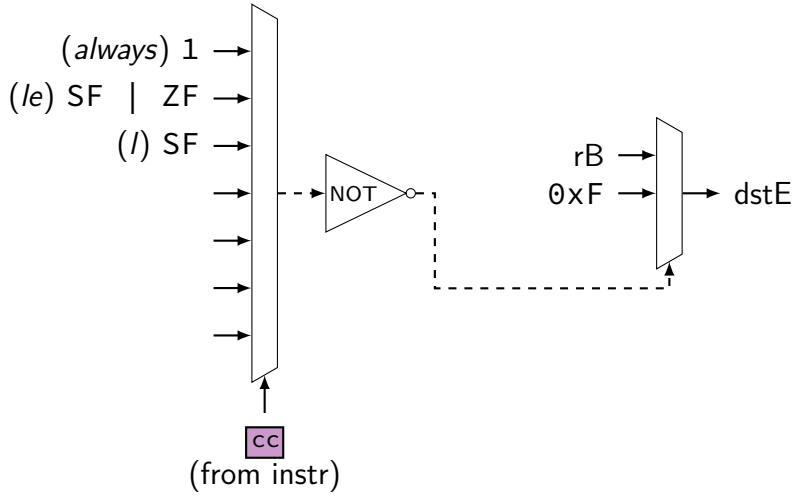
- valE — ALU output

read prior condition codes

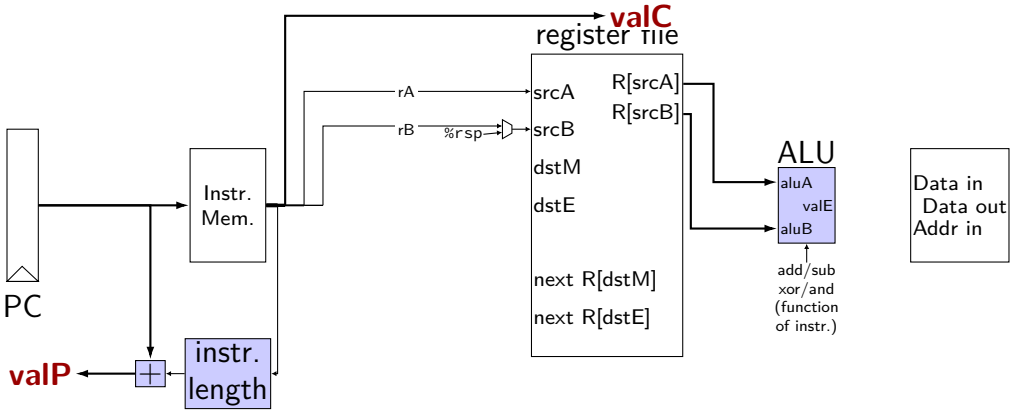
- Cnd — condition codes based on ifun (instruction type for jCC/cmovCC)

write new condition codes

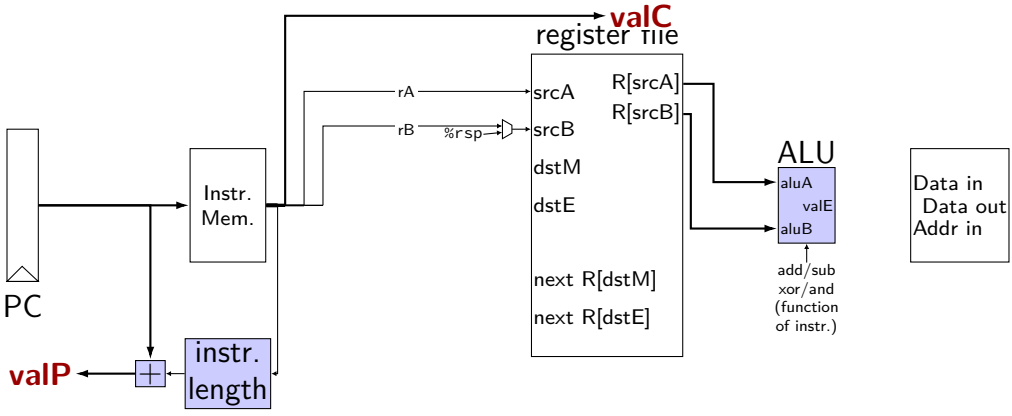
using condition codes: cmov



execute (1)


valC
register file
rA
rB
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
add/sub
xor/and
(function
of instr.)
Data in
Data out
Addr in
Instr.
Mem.
PC
valP
instr.
length

execute (1)



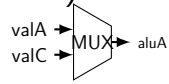
exercise: which of these instructions would there be a problem ?
`nop`, `addq`, `mrmovq`, `popq`, `call`,

SEQ: ALU operations?

ALU inputs always valA, valB (register values)?

no, inputs from instruction: (Displacement + rB)

```
mrmovq
rmmovq
```



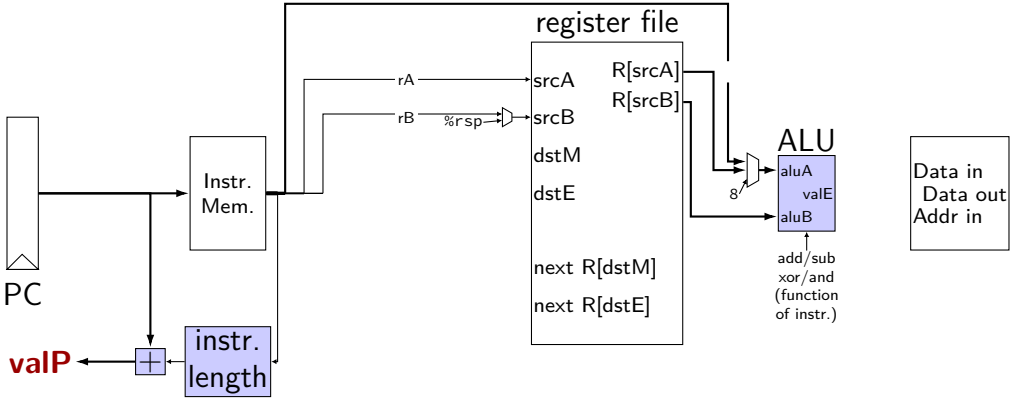
no, constants: (rsp +/- 8)

```
pushq
popq
call
ret
```

extra signals: aluA, aluB

computed ALU input values

execute (2)

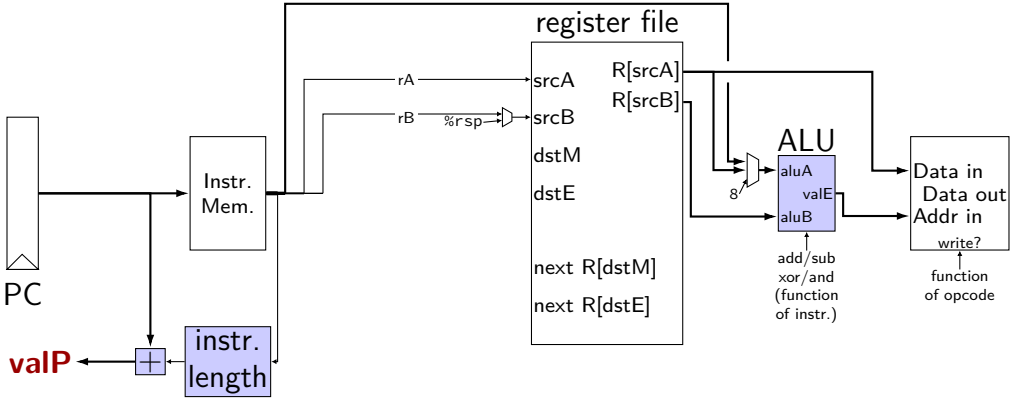

register file
rA
rB
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
8
add/sub
xor/and
(function
of instr.)
Data in
Data out
Addr in
Instr.
Mem.
PC
valP
instr.
length

SEQ: Memory

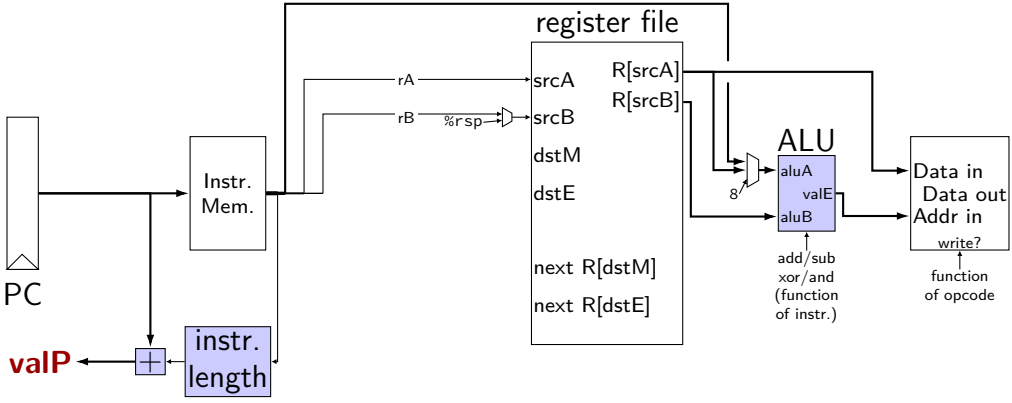
read or write data memory

valM — value read from memory (if any)

memory (1)


register file
rA
rB
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
8
add/sub
xor/and
(function
of instr.)
Data in
Data out
Addr in
write?
function
of opcode
Instr.
Mem.
PC
valP
instr.
length

memory (1)



exercise: which of these instructions would there be a problem ?
`nop`, `rmmovq`, `mrmovq`, `popq`, `call`,

SEQ: control signals for memory

read/write — read enable? write enable?

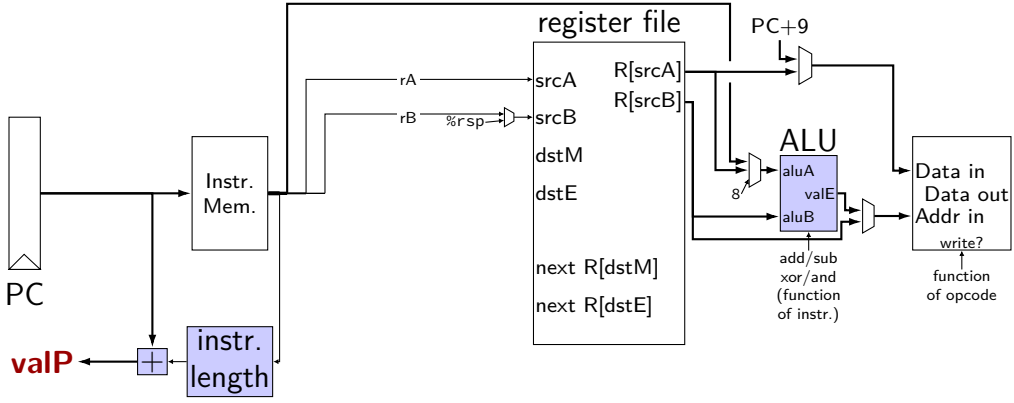
Addr — address

- mostly ALU output
- special cases (need extra MUX): `popq`, `ret`

Data — value to write

- mostly valA
- special cases (need extra MUX): `call`

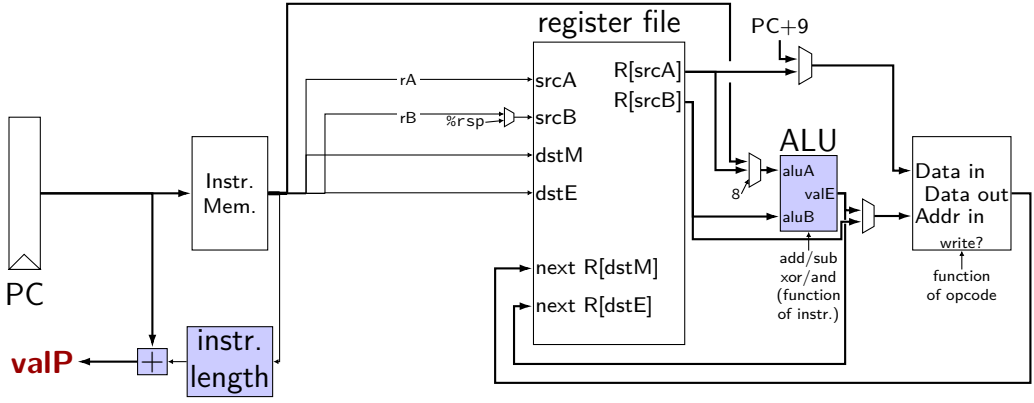
memory (2)


register file
PC+9
rA
rB
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
8
add/sub
xor/and
(function
of instr.)
Data in
Data out
Addr in
write?
function
of opcode
Instr.
Mem.
PC
valP
instr.
length

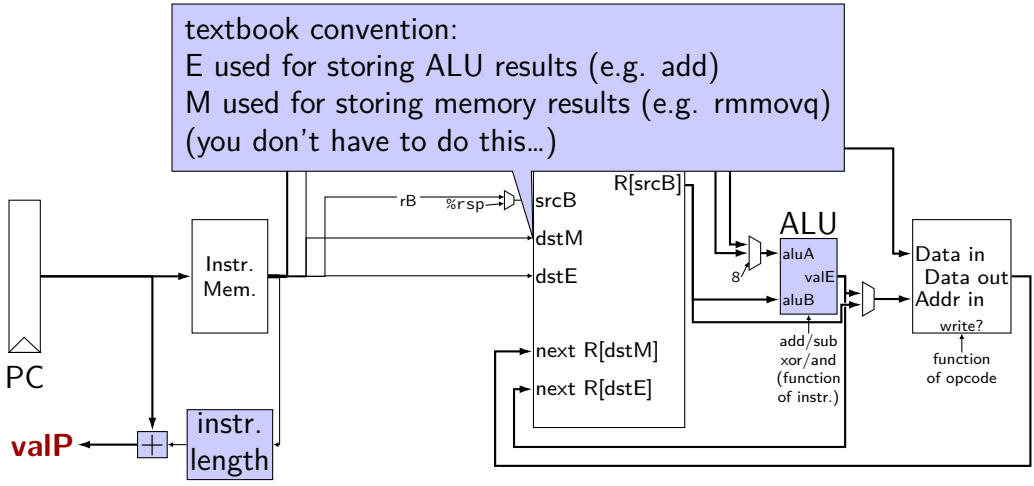
SEQ: write back

write registers

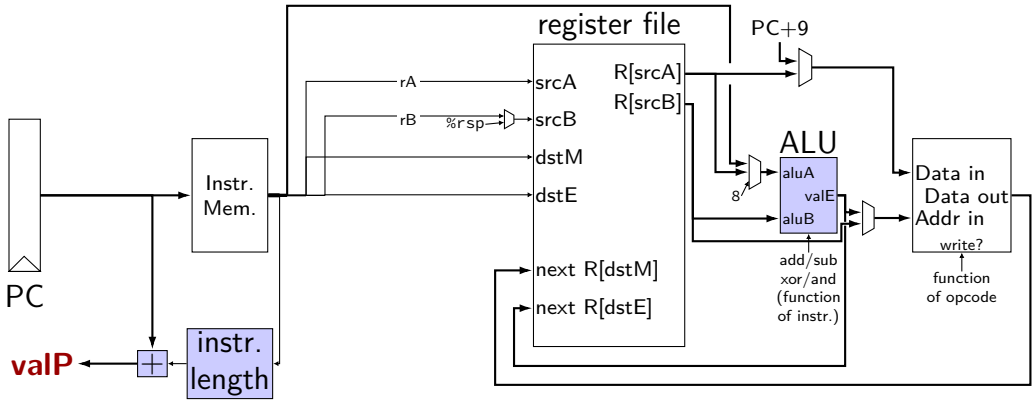
write back (1)


register file
PC+9
rA
rB
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
ALU
aluA
valE
aluB
8
Data in
Data out
Addr in
write?
function
of opcode
add/sub
xor/and
(function
of instr.)
next R[dstM]
next R[dstE]
Instr.
Mem.
PC
valP
instr.
length

write back (1)


textbook convention:
E used for storing ALU results (e.g. add)
M used for storing memory results (e.g. rmmovq)
(you don't have to do this...)
PC
Instr.
Mem.
rB
%rsp
srcB
dstM
dstE
R[srcB]
ALU
aluA
valE
aluB
8
Data in
Data out
Addr in
write?
function
of opcode
add/sub
xor/and
(function
of instr.)
next R[dstM]
next R[dstE]
valP
instr.
length

write back (1)



exercise: which of these instructions would there be a problem ?
nop, `irmovq`, `mrmovq`, `rmmovq`, addq, popq

SEQ: control signals for WB

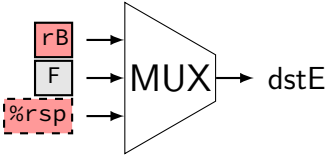
two write inputs — two needed by `popq`

valM (memory output), valE (ALU output)

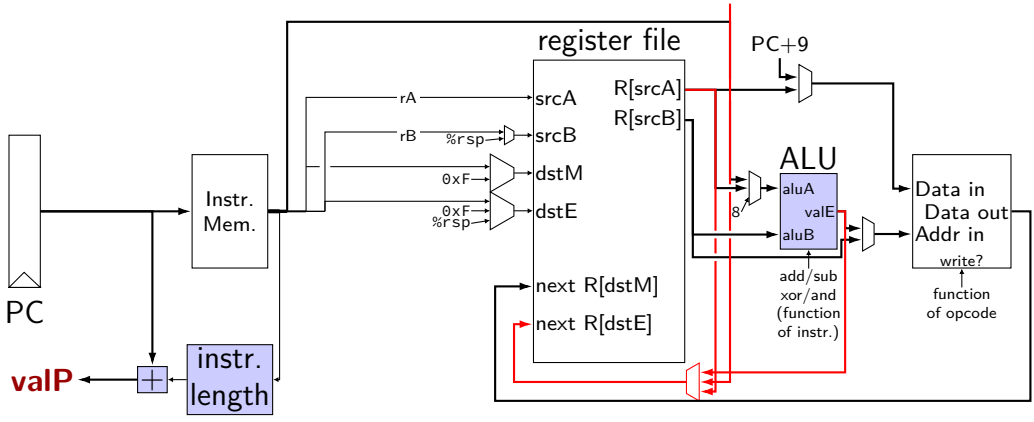
two register numbers

dstM, dstE

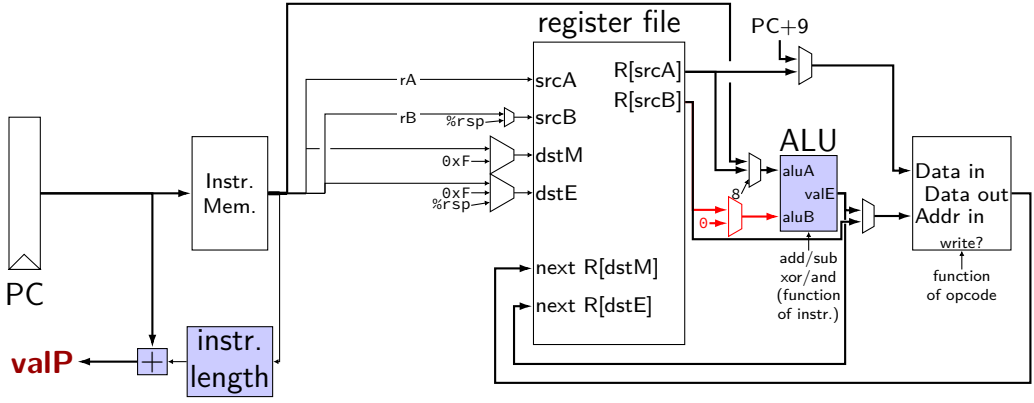
write disable — use dummy register number `0xF`



write back (2a)



write back (2b)

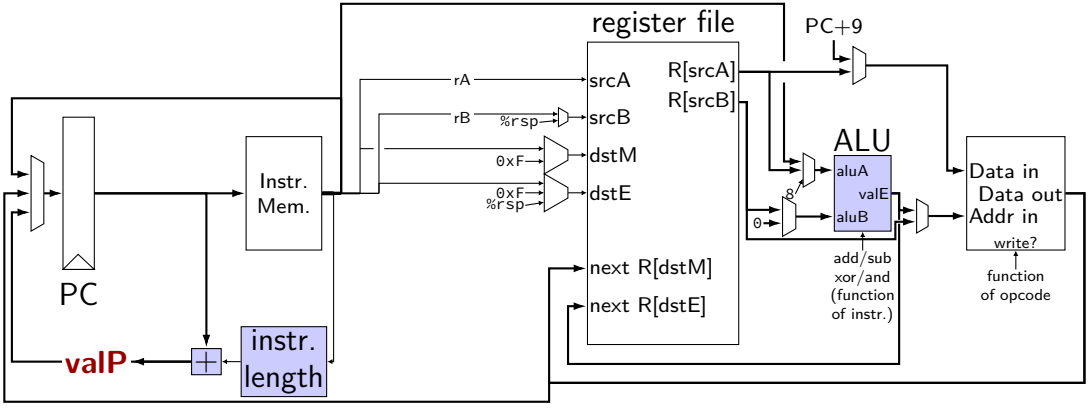


SEQ: Update PC

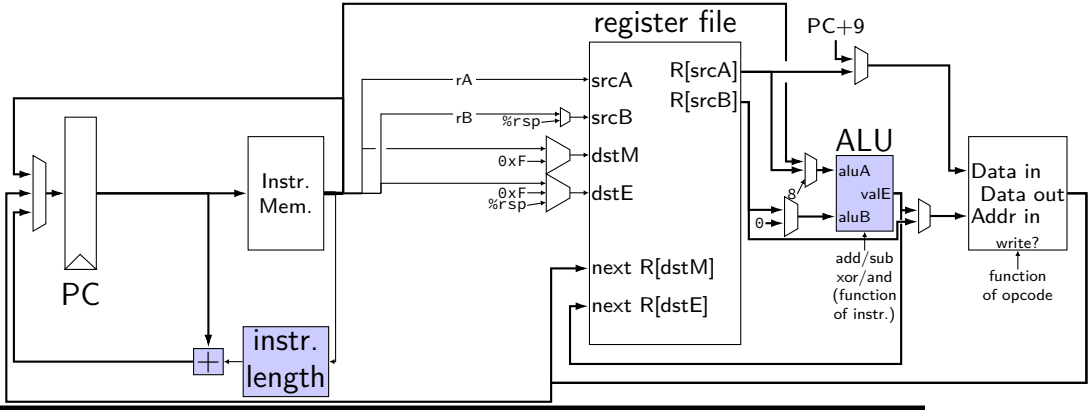
choose value for PC next cycle (input to PC register)

- usually valP (following instruction)
- exceptions: `call`, `j`*CC*, `ret`

PC update


register file
PC+9
rA
rB
%rsp
0xF
0xF
%rsp
srcA
srcB
dstM
dstE
R[srcA]
R[srcB]
next R[dstM]
next R[dstE]
ALU
aluA
valE
aluB
8
0
add/sub
xor/and
(function
of instr.)
Data in
Data out
Addr in
write?
function
of opcode
Instr.
Mem.
PC
valP
instr.
length

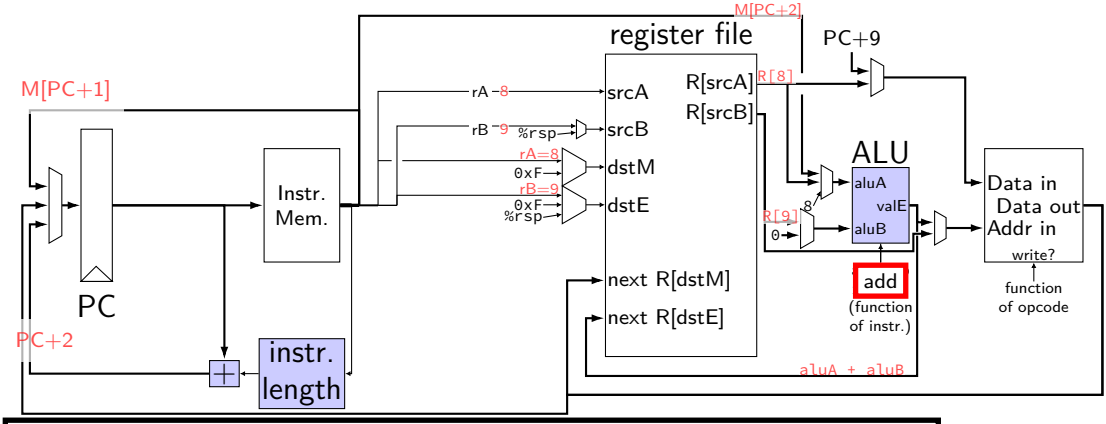
circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...

Exercise: what do they select when running `addq %r8, %r9`?

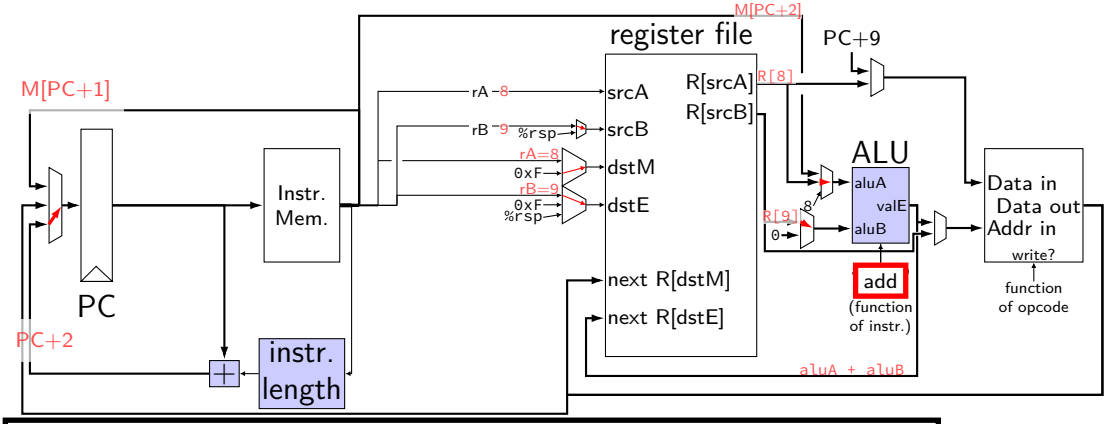
circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...

Exercise: what do they select when running `addq %r8, %r9`?

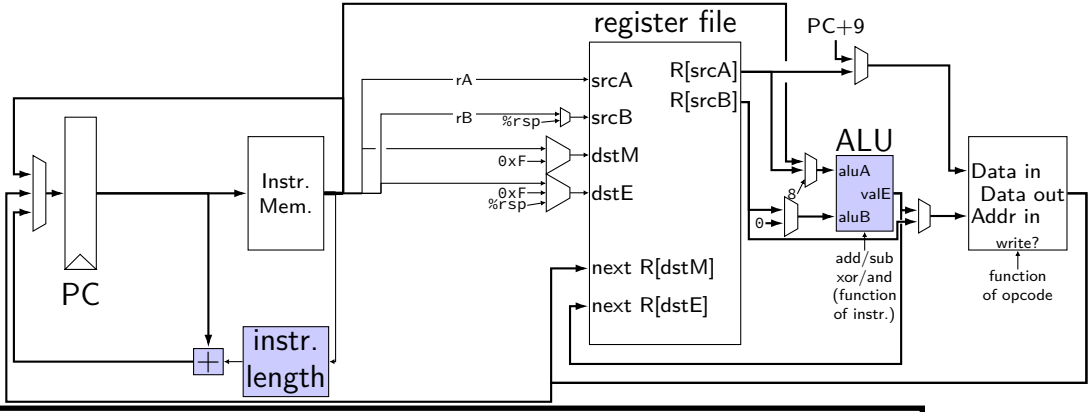
circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...

Exercise: what do they select when running `addq %r8, %r9`?

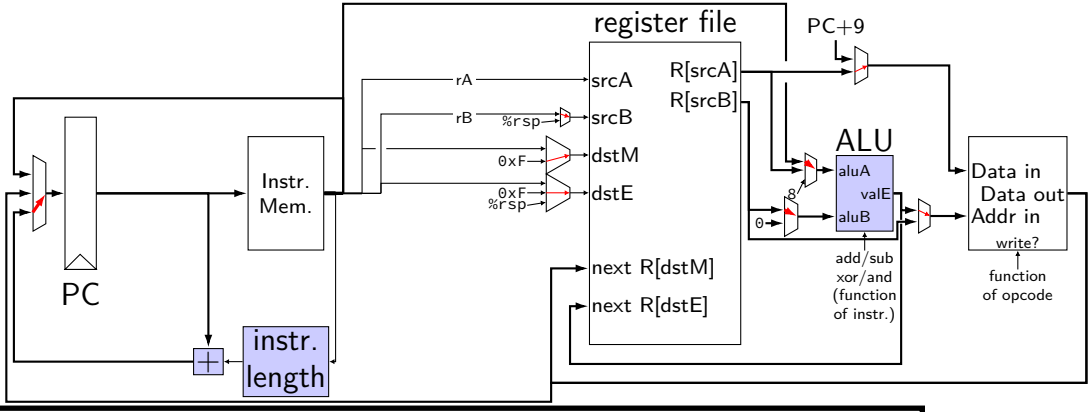
circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...

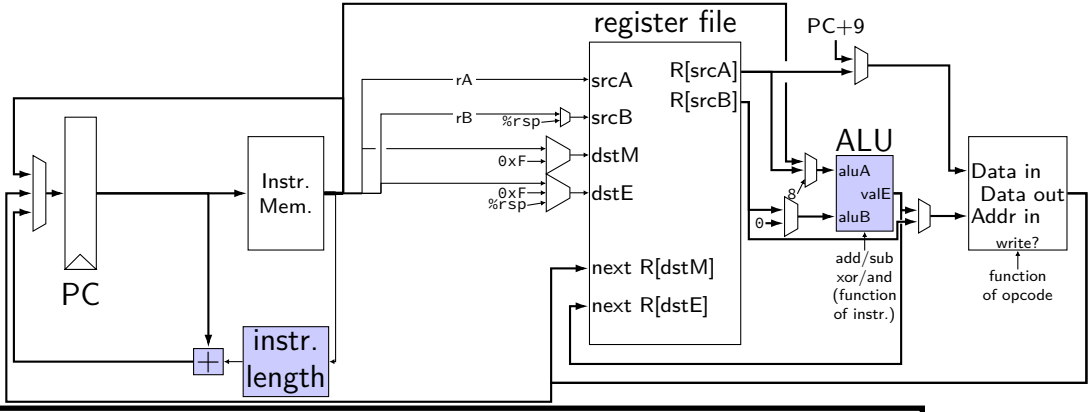
Exercise: what do they select for `rmmovq`?

circuit: setting MUXes



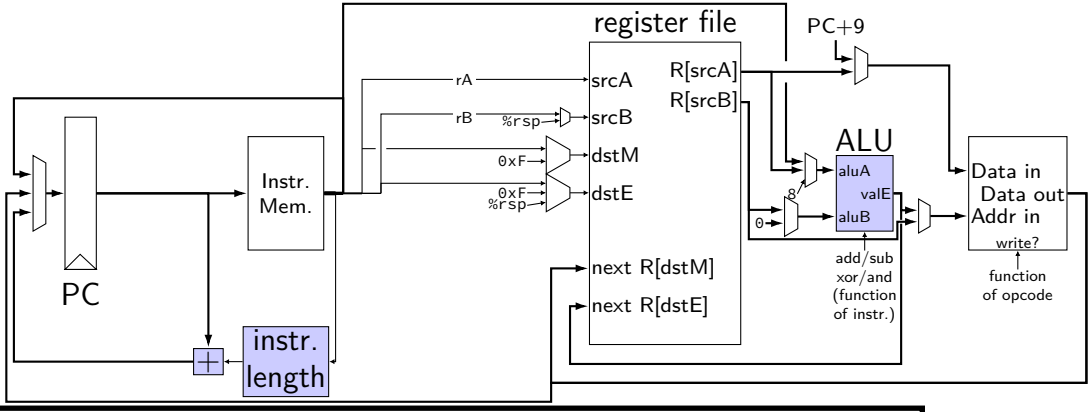
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `rmmovq`?

circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `irmovq`?

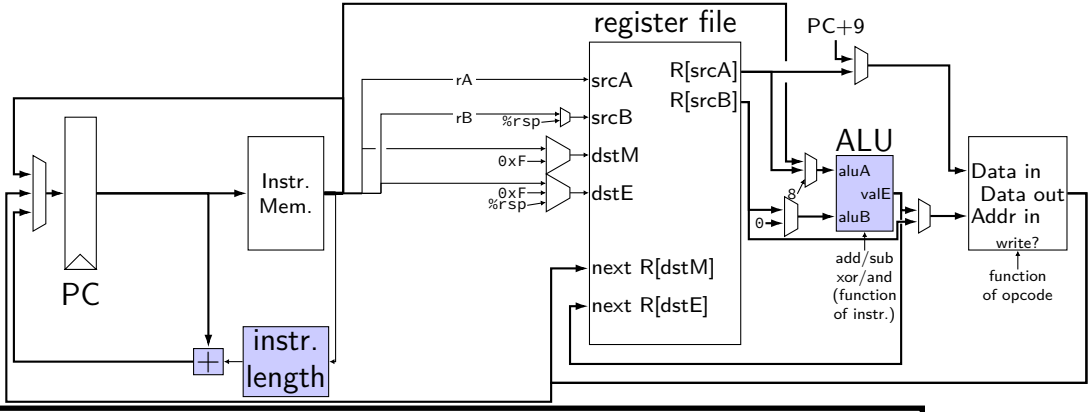
circuit: setting MUXes



MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...

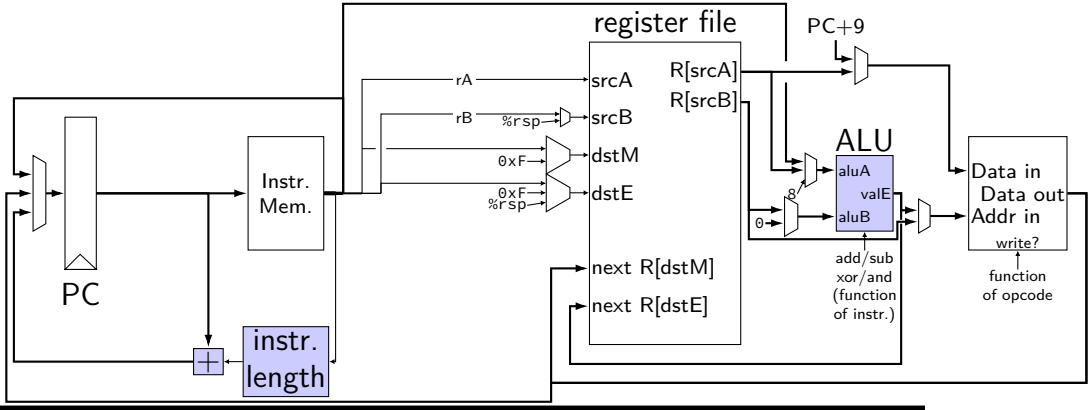
Exercise: what do they select for `mrmovq`?

circuit: setting MUXes



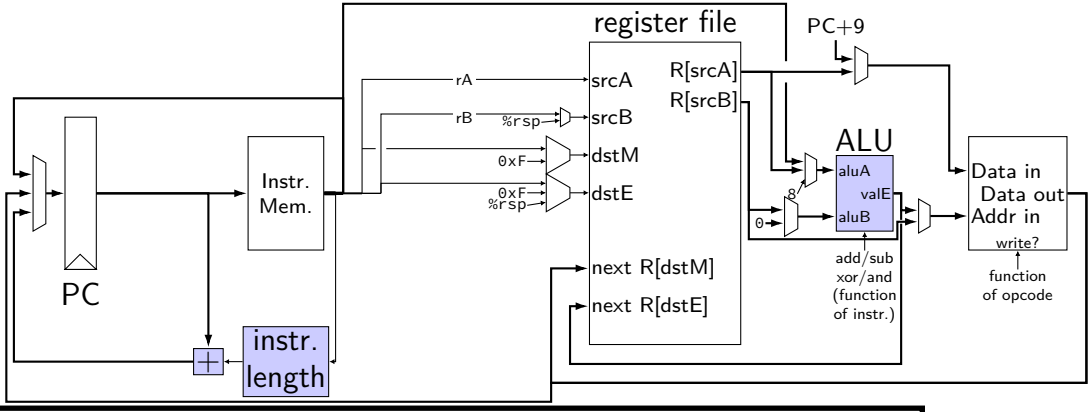
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `jle`?

circuit: setting MUXes



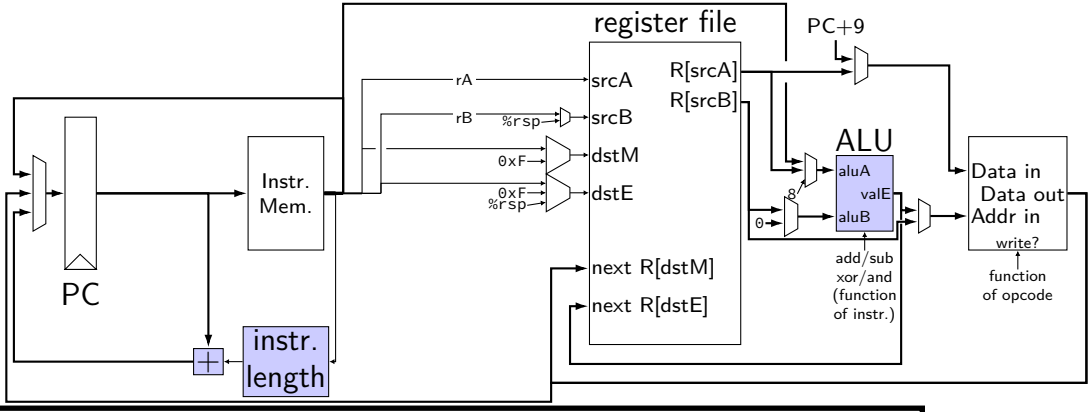
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `cmovle`?

circuit: setting MUXes



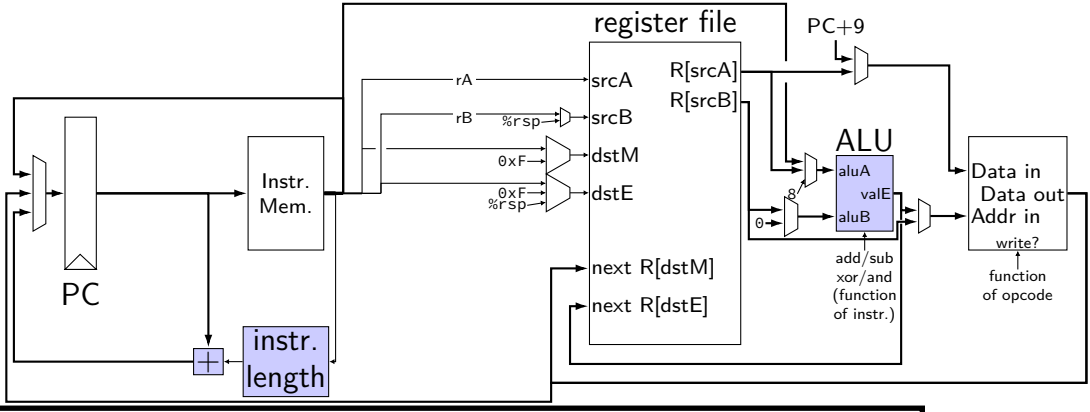
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `ret`?

circuit: setting MUXes



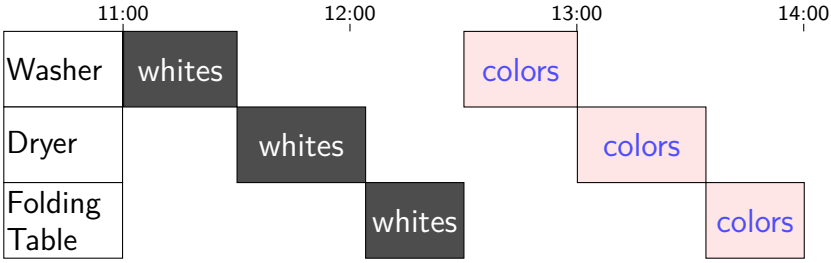
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for **popq**?

circuit: setting MUXes

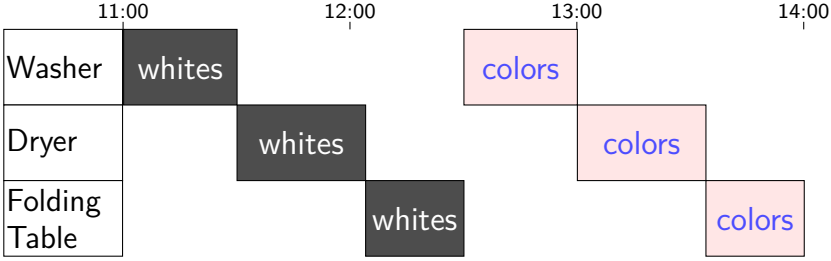


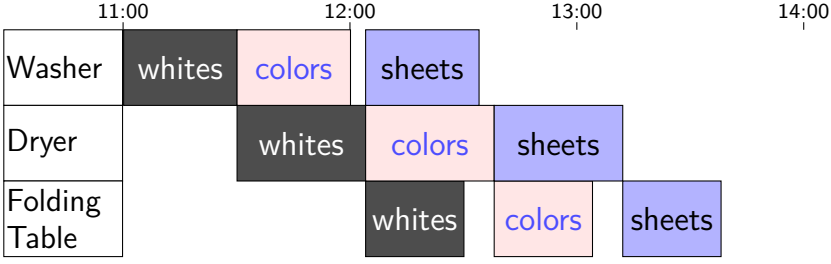
MUXes — PC, dstM, dstE, aluA, aluB, dmemIn, dmemAddr, ...
Exercise: what do they select for `call`?

Human pipeline: laundry

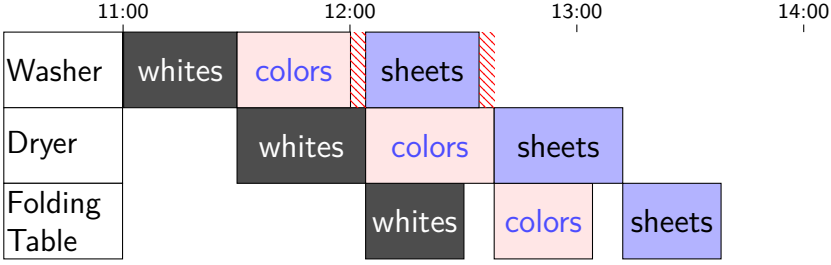


Human pipeline: laundry

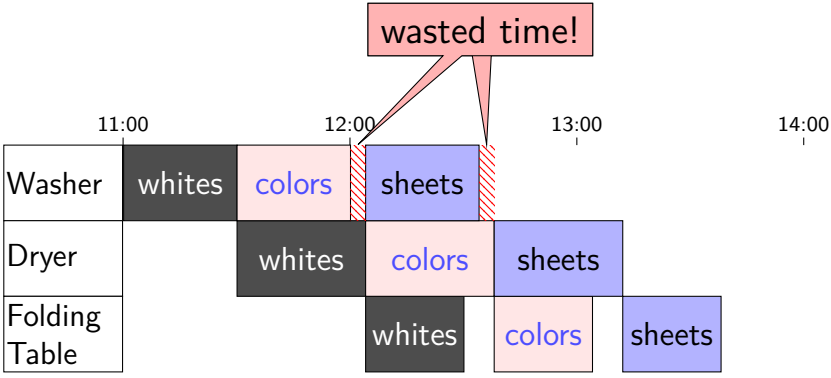




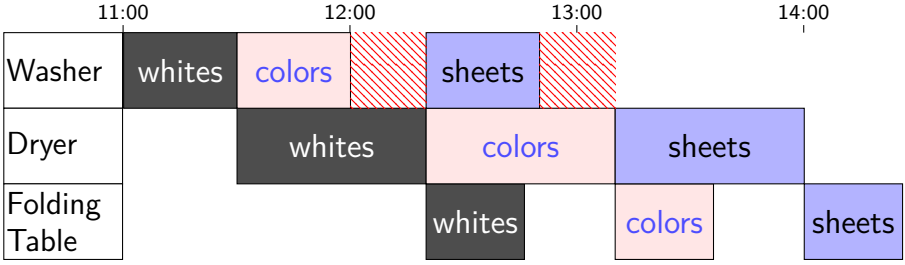
Waste (1)


11:00
12:00
13:00
14:00
Washer
Dryer
Folding Table
whites
colors
sheets
whites
colors
sheets
whites
colors
sheets

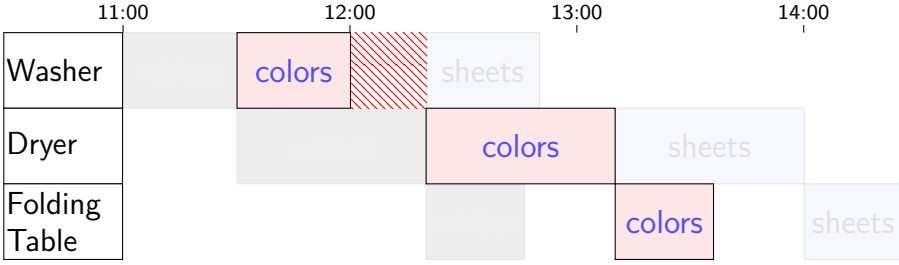
Waste (1)


wasted time!
11:00
12:00
13:00
14:00
Washer
whites
colors
sheets
Dryer
whites
colors
sheets
Folding Table
whites
colors
sheets

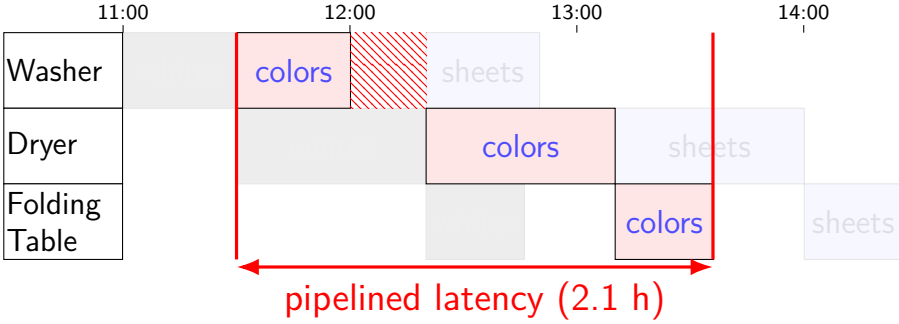
Waste (2)


11:00
12:00
13:00
14:00
Washer
Dryer
Folding Table
whites
colors
sheets
whites
colors
sheets
whites
colors
sheets

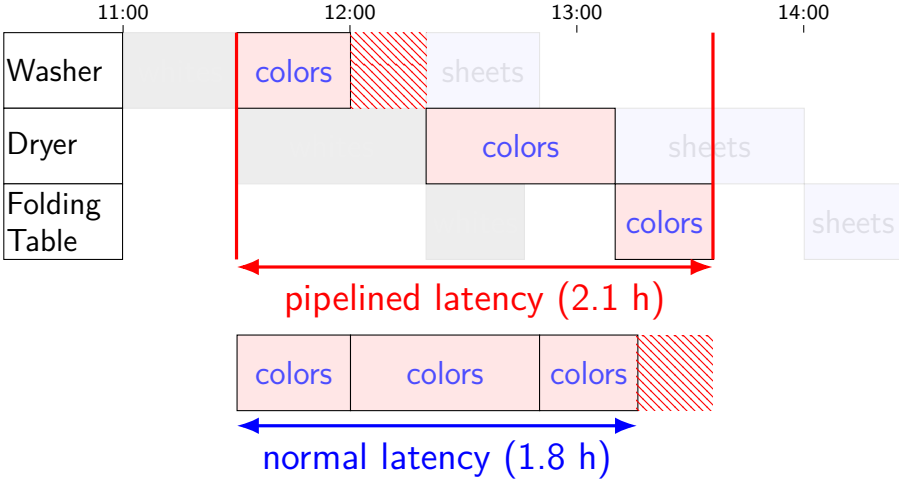
Latency — Time for One



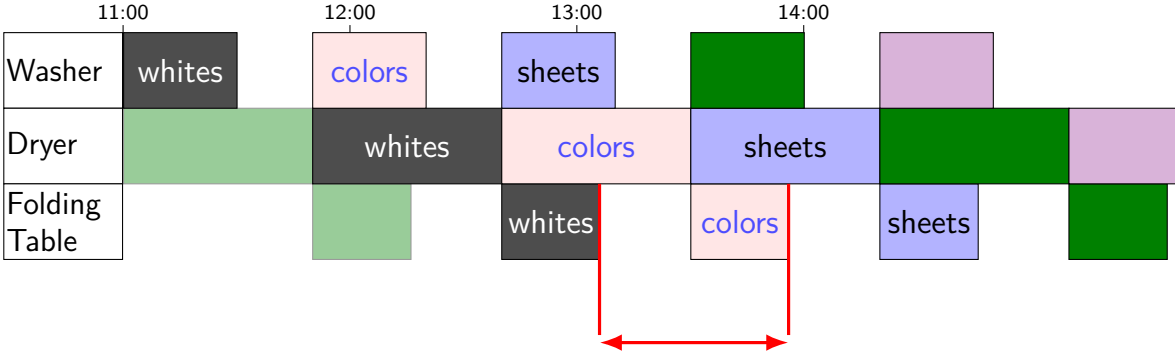
Latency — Time for One



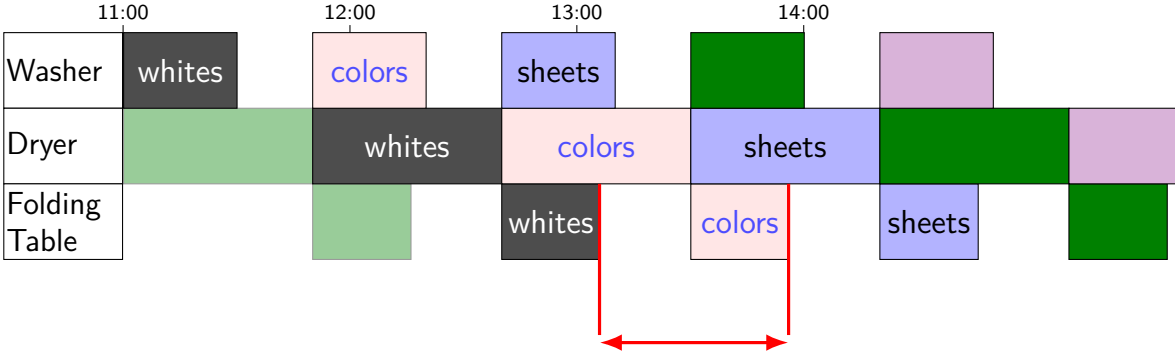
Latency — Time for One



Throughput — Rate of Many

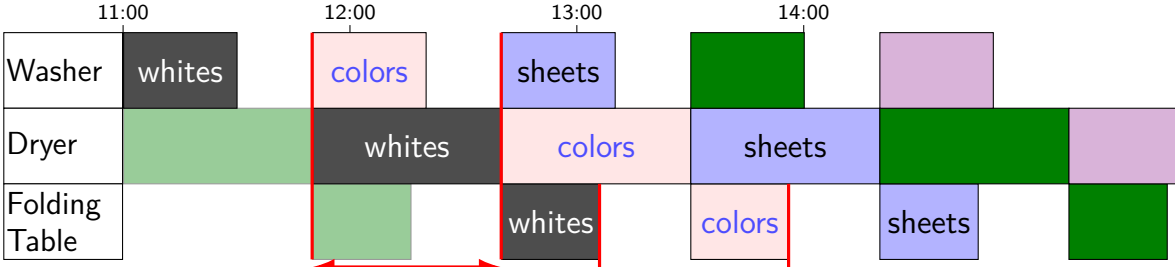


Throughput — Rate of Many



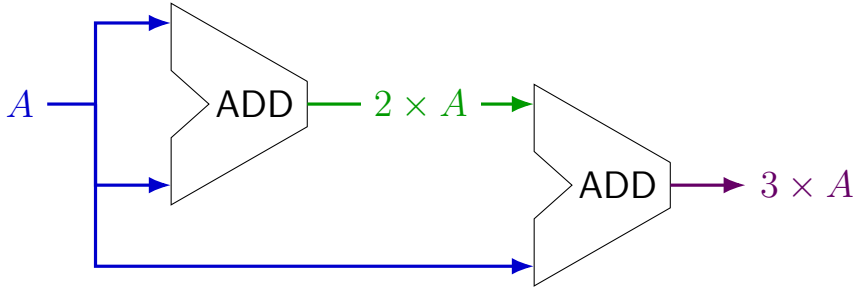
$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

Throughput — Rate of Many

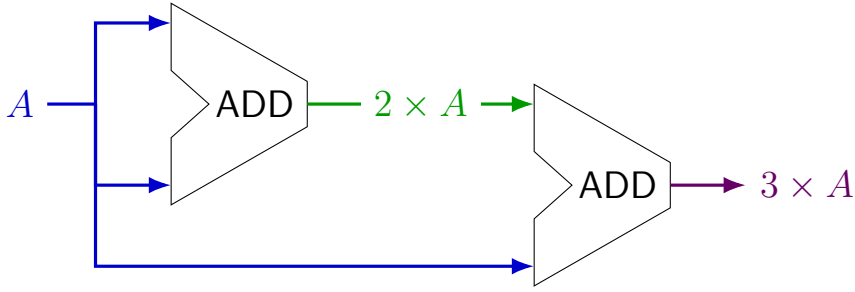


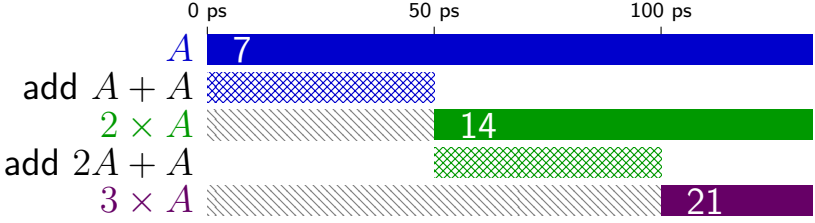
$$\frac{1 \text{ load}}{0.83\text{h}} = 1.2 \text{ loads/h}$$

times three circuit

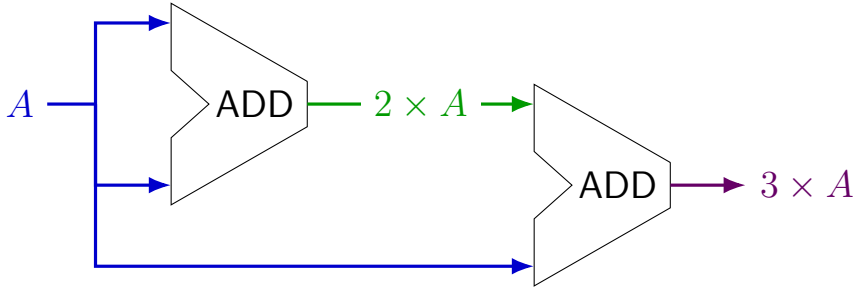


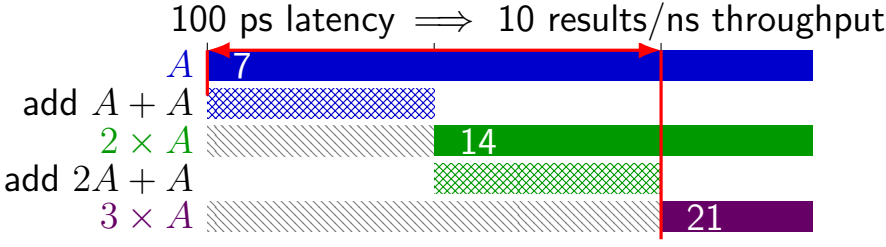
times three circuit



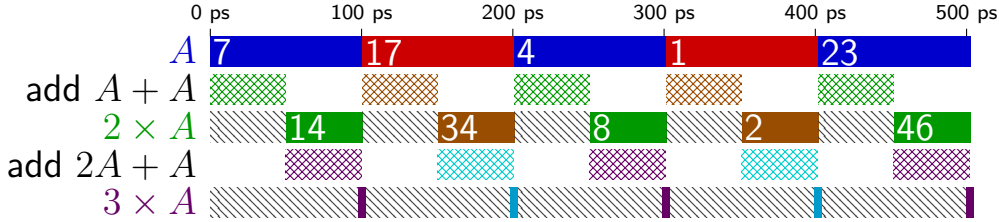


times three circuit

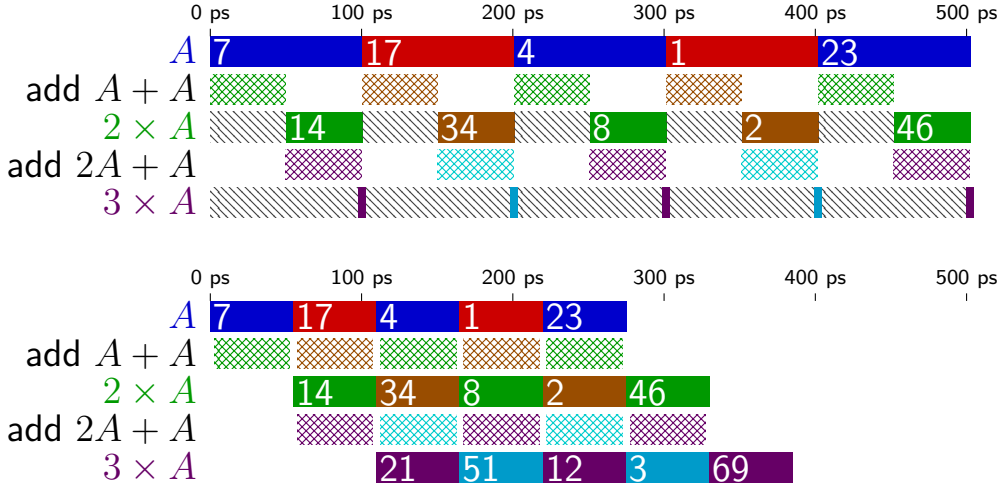




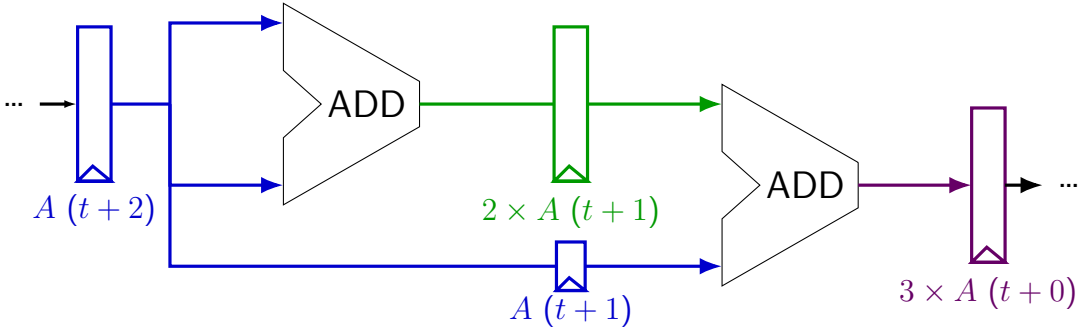
times three and repeat



times three and repeat



pipelined times three



pipelined times three