SEQ part 2 / HCL

last time

Y86 en/decoding

wires and wires bundle

combinatorial logic

registers and clocked logic

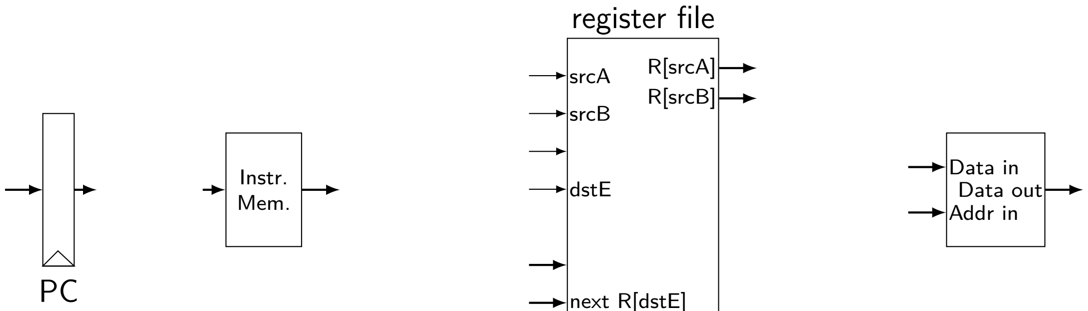
memories and register files

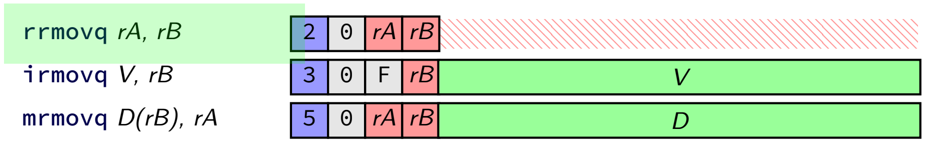
MUXes and simple processors

simple ISA 4: mov-to-register

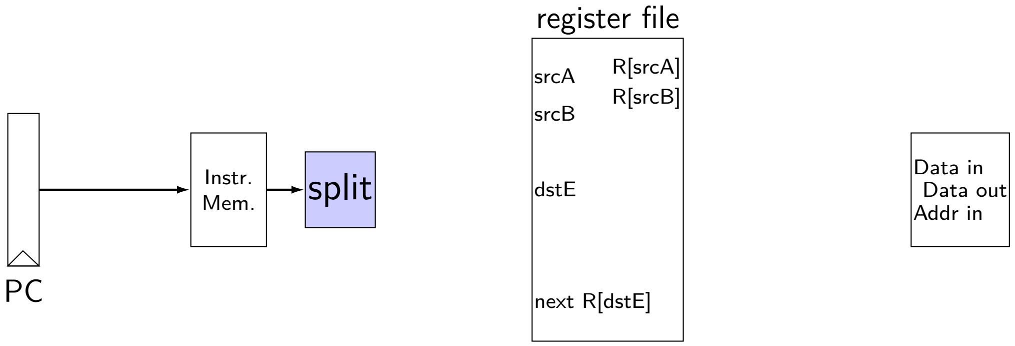
```
irmovq $constant, %rYY

rrmovq %rXX, %rYY

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

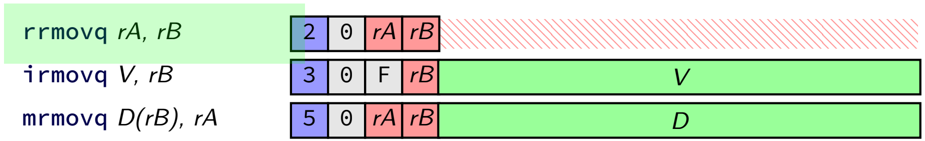
mov-to-register CPU



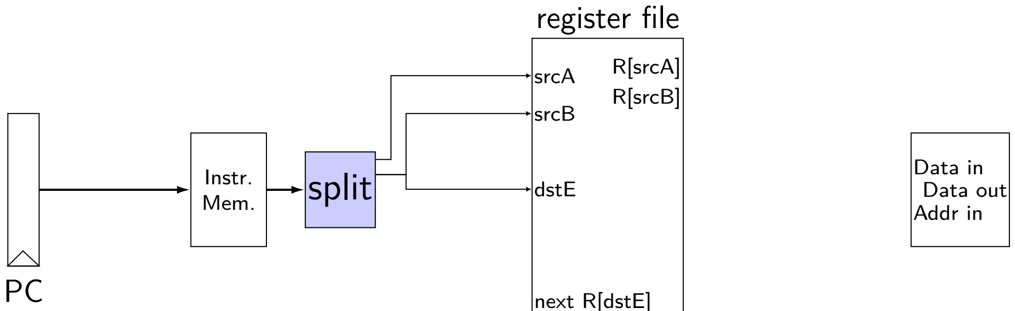


mov-to-register CPU



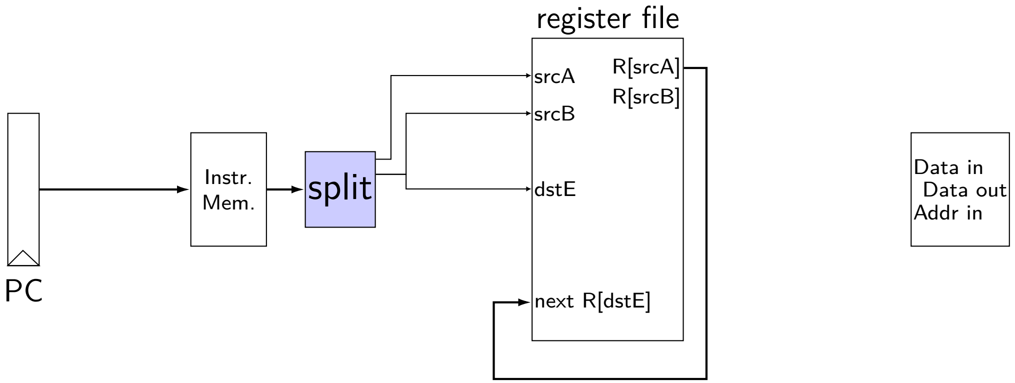


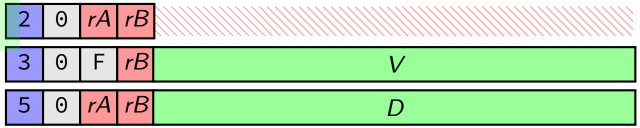
mov-to-register CPU



`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



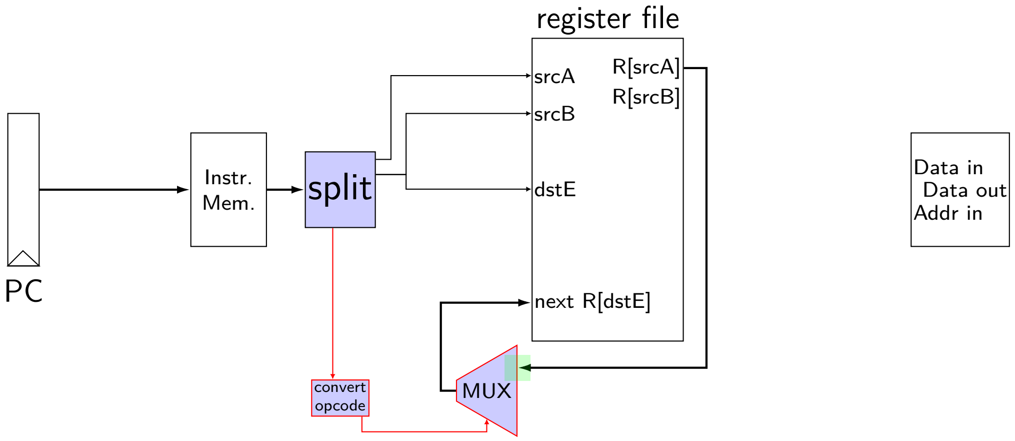


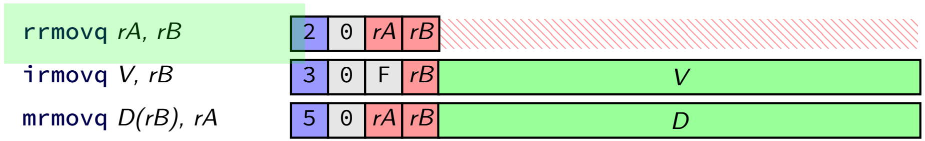
`rrmovq` *rA*, *rB*

`irmovq` *V*, *rB*

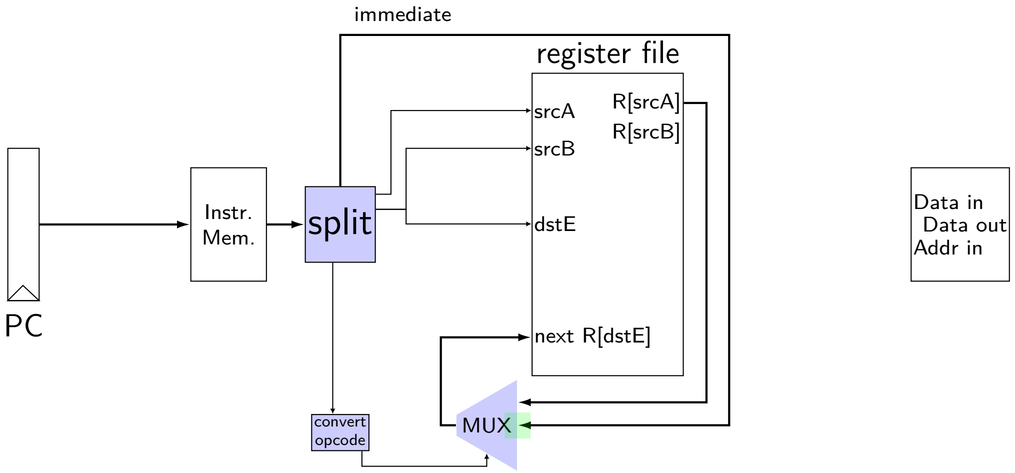
`mrmovq` *D*(*rB*), *rA*

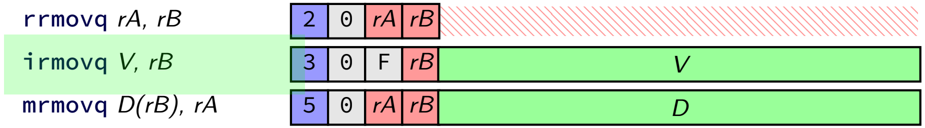
mov-to-register CPU





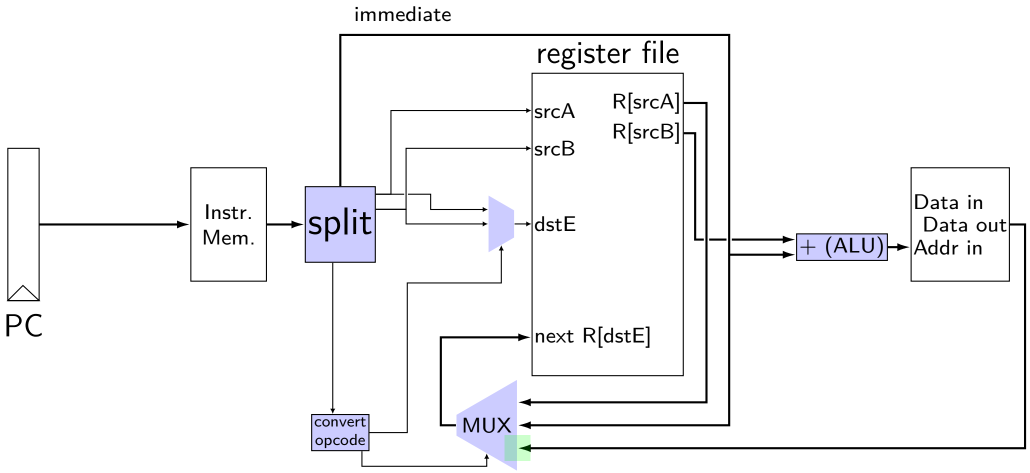
mov-to-register CPU





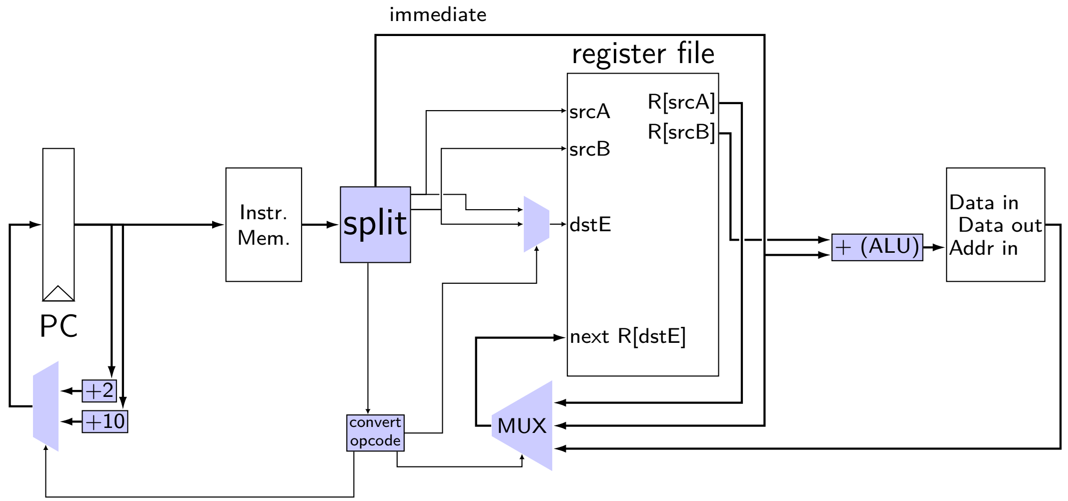
`rrmovq` *rA*, *rB*	2	0	*rA*	*rB*	▨
`irmovq` *V*, *rB*	3	0	F	*rB*	*V*
`mrmovq` *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

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

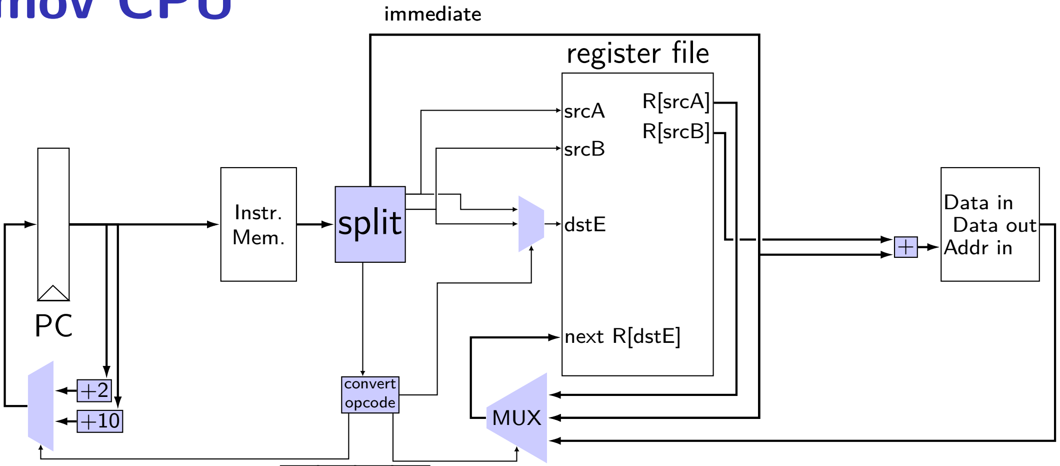


`rrmovq` *rA*, *rB*	2	0	*rA*	*rB*	▨
`irmovq` *V*, *rB*	3	0	F	*rB*	*V*
`mrmovq` *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*

simple ISA 4B: mov

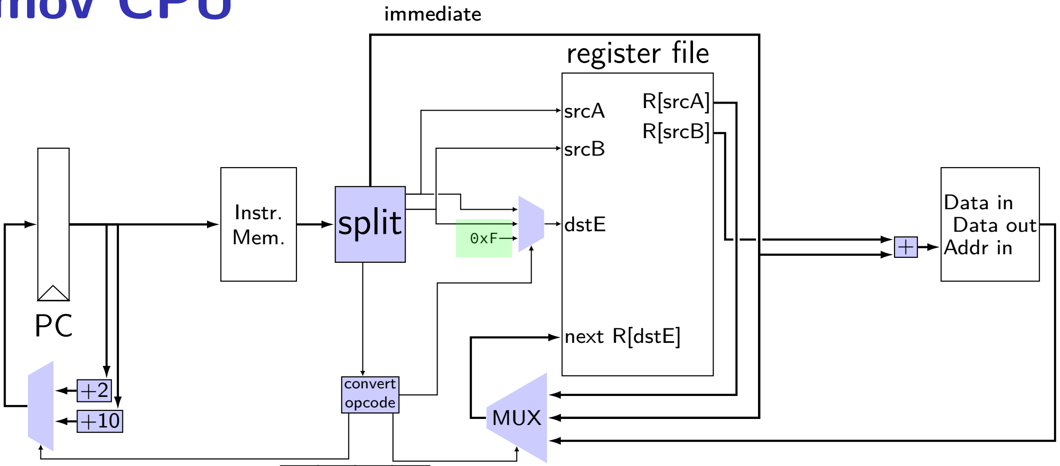
```
irmovq $constant, %rYY

rrmovq %rXX, %rYY

mrmovq 10(%rXX), %rYY

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

mov CPU



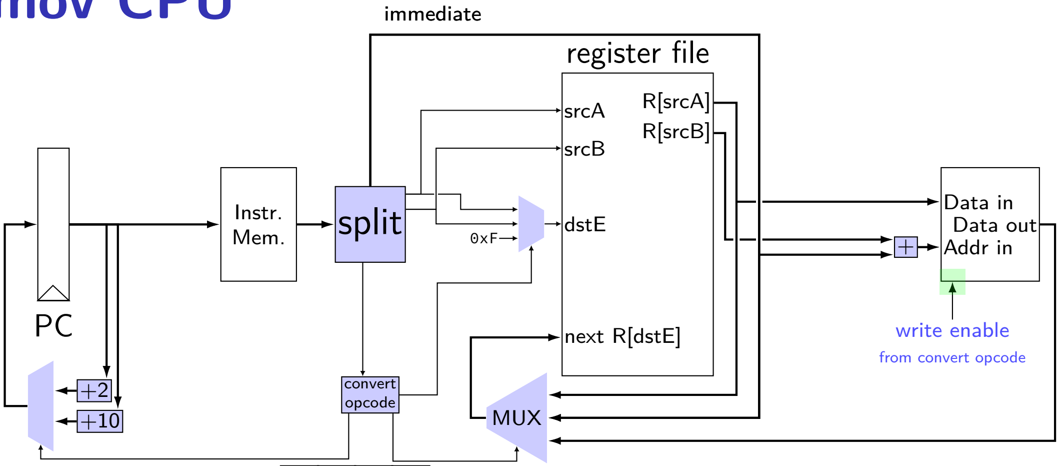
`rrmovq` *rA, rB*	2	0	*rA*	*rB*	▨
`irmovq` *V, rB*	3	0	F	*rB*	*V*
`mrmovq` *D(rB), rA*	5	0	*rA*	*rB*	*D*
`rmmovq` *rA, D(rB)*	4	0	*rA*	*rB*	*D*

mov CPU



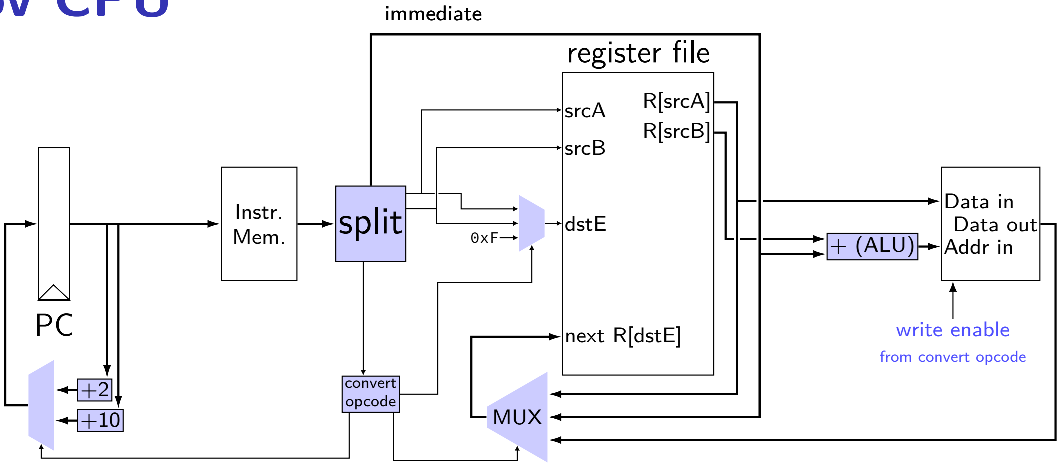
`rrmovq` *rA*, *rB*	2	0	*rA*	*rB*	▨
`irmovq` *V*, *rB*	3	0	F	*rB*	*V*
`mrmovq` *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*
`rmmovq` *rA*, *D*(*rB*)	4	0	*rA*	*rB*	*D*

mov CPU

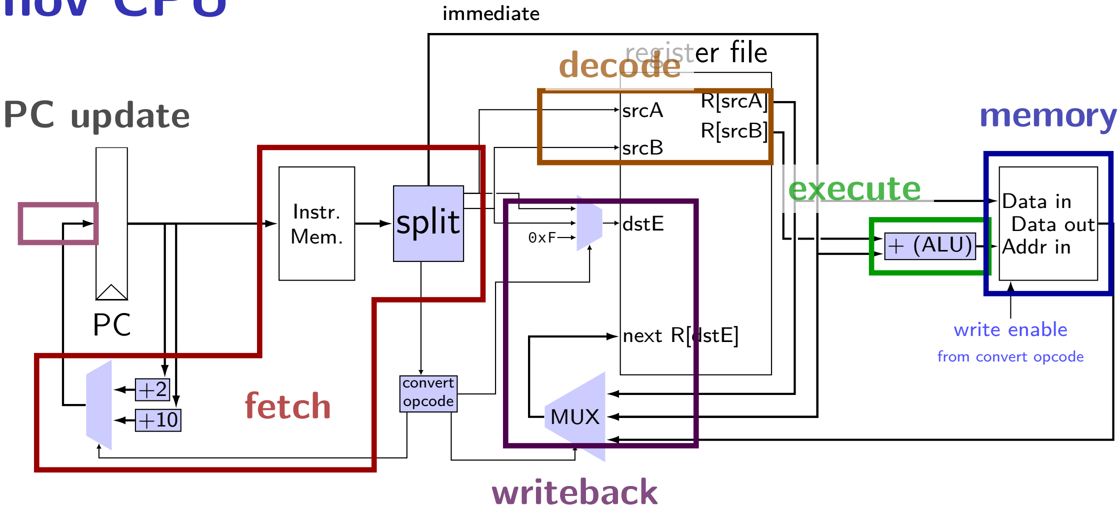


`rrmovq` *rA*, *rB*	2	0	*rA*	*rB*	▨
`irmovq` *V*, *rB*	3	0	F	*rB*	*V*
`mrmovq` *D*(*rB*), *rA*	5	0	*rA*	*rB*	*D*
`rmmovq` *rA*, *D*(*rB*)	4	0	*rA*	*rB*	*D*

mov CPU


immediate
register file
srcA
R[srcA]
srcB
R[srcB]
dstE
0xF
next R[dstE]
Instr.
Mem.
split
PC
+2
+10
convert
opcode
MUX
Data in
Data out
Addr in
+ (ALU)
write enable
from convert opcode

mov CPU


immediate
register file
decode
PC update
srcA
srcB
R[srcA]
R[srcB]
memory
execute
Instr.
Mem.
split
Data in
Data out
Addr in
+ (ALU)
0xF
dstE
PC
write enable
from convert opcode
next R[dstE]
+2
+10
convert opcode
fetch
MUX
writeback

Stages

conceptual division of instruction:

fetch — read instruction memory, split instruction, compute length

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

Order when these events happen `pushq %rax` instruction:

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

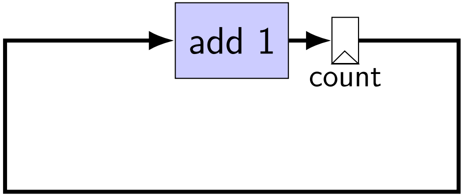
Hint: recall how registers, register files, memory works

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

describing hardware

how do we describe hardware?

pictures?



circuits with pictures?

yes, something you can do

such commercial tools exist, but...

not commonly used for processors

hardware description language

programming language for hardware

(typically) text-based representation of circuit

often abstracts away details like:

- how to build arithmetic operations from gates
- how to build registers from transistors
- how to build memories from transistors
- how to build MUXes from gates
- …

those details also not a topic in this course

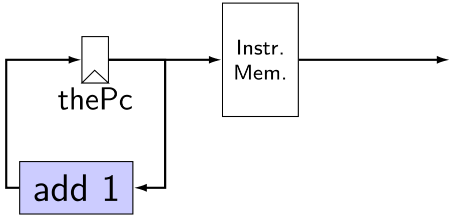
our tool: HCLRS

built for this course

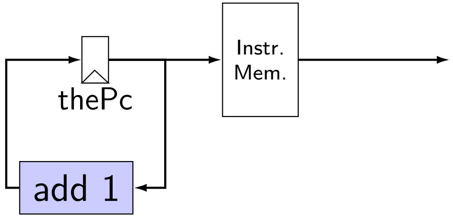
assumes you're making a processor

somewhat different from textbook's HCL

nop CPU

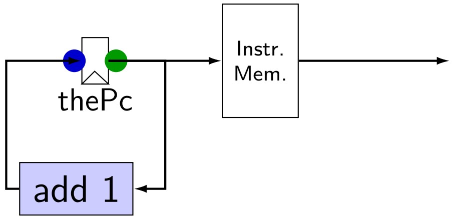


nop CPU



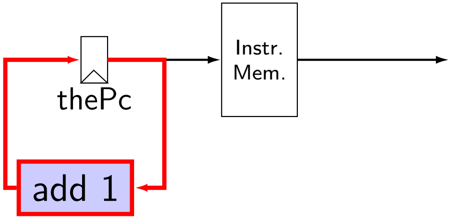
```
register pF {
    thePc : 64 = 0;
}
```

nop CPU



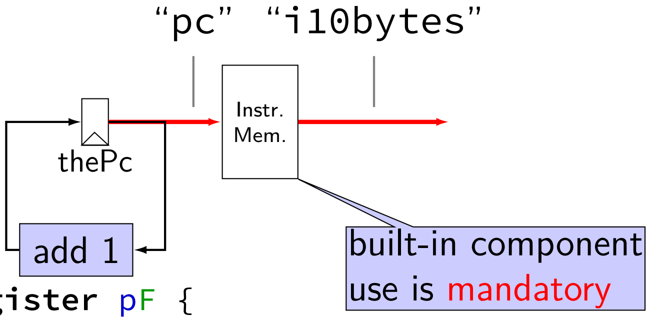
```
register pF {
    thePc : 64 = 0;
}
```

nop CPU



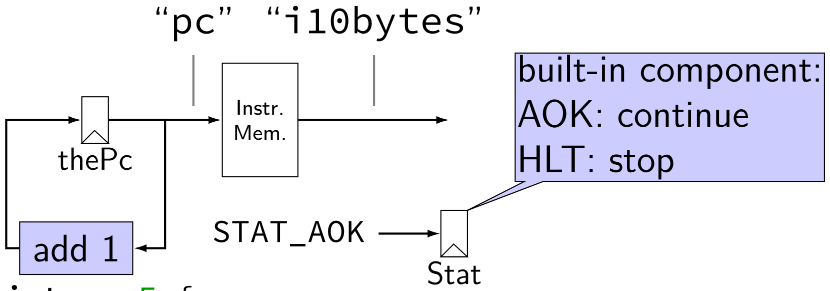
```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
```

nop CPU



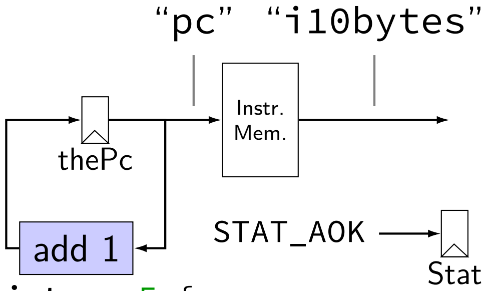
```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```

nop CPU



```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
Stat = STAT_AOK;
```

nop CPU



```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
Stat = STAT_AOK;
```

nop CPU: running

need a program in memory

- .yo file

`tools/yas` — convert `.ys` to `.yo`

`tools/yis` — reference interpreter for `.yo` files

- if your processor doesn't do the same thing...

can build tools by running `make`

nop CPU: creating a program

create assemby file: nops.ys:

```
    nop
    nop
    nop
    nop
    nop
```

assemble using `tools/yas nops.ys` or `make nops.yo`

nop.yo

more readable/simpler than normal executables:

```
0x000: 10                      | nop
0x001: 10                      | nop
0x002: 10                      | nop
0x003: 10                      | nop
0x004: 10                      | nop
                               |
```

loaded into data and program memory

parts left of | just comments

running a simulator (1)

```
Usage: ./hclrs [options] HCL-FILE [YO-FILE [TIMEOUT]]
Runs HCL_FILE on YO-FILE. If --check is specified, no YO-FILE may be supplied.
Default timeout is 9999 cycles.

Options:
    -c, --check         check syntax only
    -d, --debug         output wire values after each cycle and other debug
                        output
    -q, --quiet         only output state at the end
    -t, --testing       do not output custom register banks (for autograding)
    -h, --help          print this help menu
    -i, --interactive   prompt after each cycle
        --trace-assignments
                        show assignments in the order they are simulated
        --version       print version number
```

running a simulator (2)

```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:               0   RCX:               0   RDX:               0 |
| RBX:               0   RSP:               0   RBP:               0 |
| RSI:               0   RDI:               0   R8:                0 |
| R9:                0   R10:               0   R11:               0 |
| R12:               0   R13:               0   R14:               0 |
| register pF(N)  thePc=0000000000000000                            |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f   |
|  0x0000000_:   10 10 10 10  10                                       |
+----------------------------------------------------------------------+
pc = 0x0; loaded [10 : nop]
+------------------- between cycles    1 and    2 ----------------------+
....
```

running a simulator (2)

```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:               0   RCX:               0   RDX:               0 |
| RBX:               0   RSP:               0   RBP:               0 |
| RSI:               0   RDI:               0   R8:                0 |
| R9:                0   R10:               0   R11:               0 |
| R12:               0   R13:               0   R14:               0 |
| register pF(N)  thePc=0000000000000000                            |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                       |
+----------------------------------------------------------------------+
pc = 0x0; loaded [10 : nop]
+------------------- between cycles    1 and    2 ----------------------+
....
```

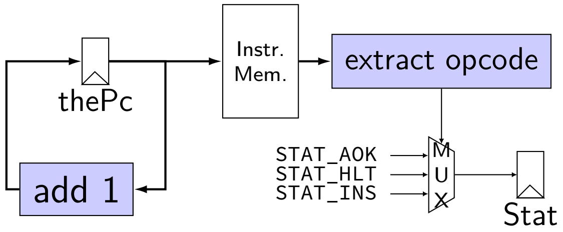
running a simulator (2)

```
$ ./hclrs nop_cpu.hcl nops.yo
+-------------------- between cycles    0 and    1 ----------------------+
| RAX:               0   RCX:               0   RDX:               0 |
| RBX:               0   RSP:               0   RBP:               0 |
| RSI:               0   RDI:               0   R8:                0 |
| R9:                0   R10:               0   R11:               0 |
| R12:               0   R13:               0   R14:               0 |
| register pF(N)  thePc=0000000000000000                            |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                        |
+-----------------------------------------------------------------------+
pc = 0x0; loaded [10 : nop]
+-------------------- between cycles    1 and    2 ----------------------+
....
```

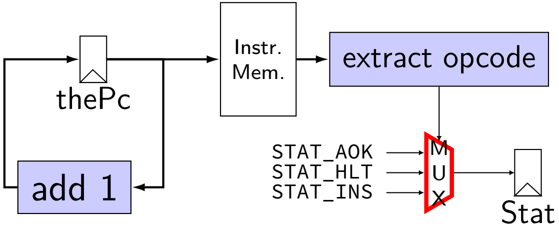
running a simulator (2)

```
$ ./hclrs nop_cpu.hcl nops.yo
+-------------------- between cycles    0 and    1 ----------------------+
| RAX:               0   RCX:               0   RDX:               0 |
| RBX:               0   RSP:               0   RBP:               0 |
| RSI:               0   RDI:               0   R8:                0 |
| R9:                0   R10:               0   R11:               0 |
| R12:               0   R13:               0   R14:               0 |
| register pF(N)  thePc=0000000000000000                              |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f   |
|  0x0000000_:   10 10 10 10  10                                       |
+----------------------------------------------------------------------+
pc = 0x0; loaded [10 : nop]
+-------------------- between cycles    1 and    2 ----------------------+
....
```

nop/halt CPU


Instr.
Mem.
extract opcode
thePc
add 1
STAT_AOK
STAT_HLT
STAT_INS
MUX
Stat

nop/halt CPU


thePc
add 1
Instr.
Mem.
extract opcode
STAT_AOK
STAT_HLT
STAT_INS
MUX
Stat

MUXes in HCLRS

book calls “case expression”

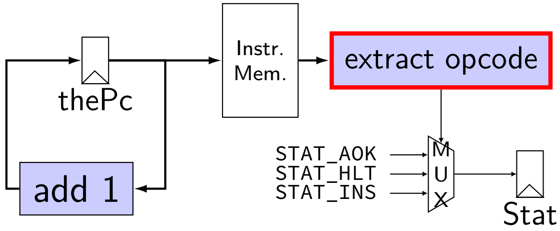
conditions evaluated (as if) in order

first match is output: `result = [`

```
    x == 5: 1;
    x in {0, 6}: 2;
    x > 2: 3;
    1: 4;
];
```

x = 5: result is 1
x = 6: result is 2
x = 3: result is 3
x = 4: result is 3
x = 1: result is 4

nop/halt CPU



Instr.
Mem.
extract opcode
thePc
add 1
STAT_AOK
STAT_HLT
STAT_INS
MUX
Stat

subsetting bits in HCLRS

extracting bits 2 (inclusive)–9 (exclusive): `value[2..9]`

least significant bit is bit 0

bit numbers and instructions

value from instruction memory in `i10bytes`

HCLRS numbers bits from LSB to MSB

80-bit integer, little-endian order:

first byte is least significant byte

HCLRS bit '0' is least significant bit

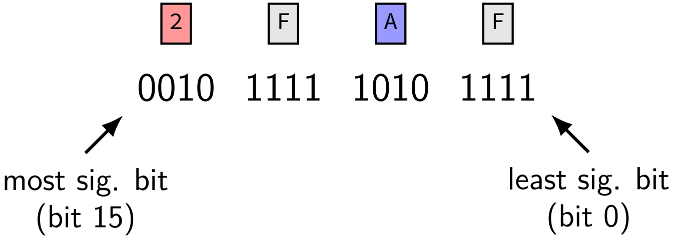
example

`pushq %rbx` at memory address x: A F 2 F

memory at $x+0$: pushq F; at $x+1$: rbx F

$x+0$: A F; at $x+1$: 2 F

as a little-endian 2-byte number in typical English order:



Y86 encoding table

byte:	0		1		2	3	4	5	6	7	8	9
`halt`	0	0										
`nop`	1	0										
`rrmovq/cmovCC` *rA*, *rB*	2	*cc*	*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*							
OP`q` *rA*, *rB*	6	*fn*	*rA*	*rB*								
`j`*CC* *Dest*	7	*cc*	*Dest*									
`call` *Dest*	8	0	*Dest*									
`ret`	9	0										
`pushq` *rA*	A	0	*rA*	F								
`popq` *rA*	B	0	*rA*	F								

Y86 encoding table

byte:	0		1		2	3	4	5	6	7	8	9
`halt`	0	0										
`nop`	1	0										
`rrmovq/cmovCC` *rA*, *rB*	2	*cc*	*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*							
OP`q` *rA*, *rB*	6	*fn*	*rA*	*rB*								
`j`*CC* *Dest*	7	*cc*	*Dest*									
`call` *Dest*	8	0	*Dest*									
`ret`	9	0										
`pushq` *rA*	A	0	*rA*	F								
`popq` *rA*	B	0	*rA*	F								

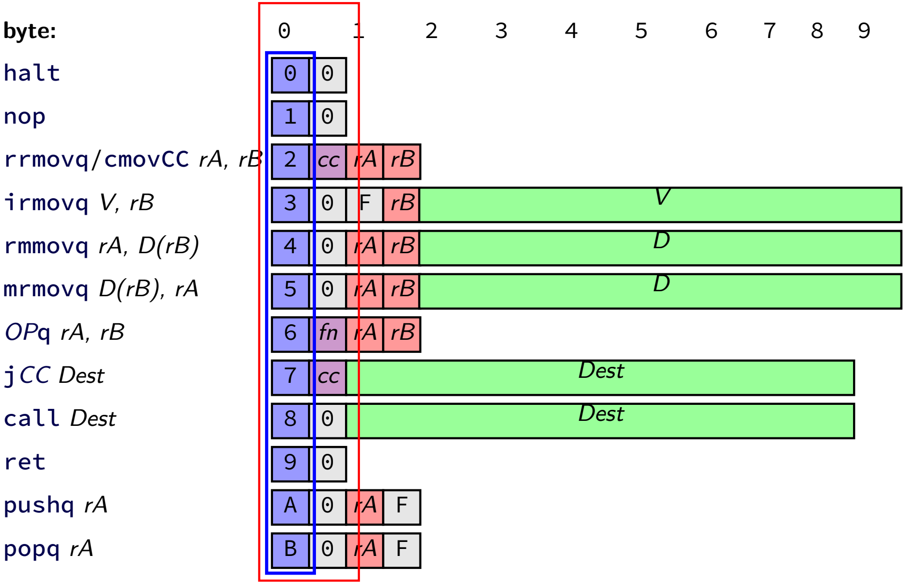
byte 0: bits 0–7

Y86 encoding table

byte:	0		1		2	3	4	5	6	7	8	9
`halt`	0	0										
`nop`	1	0										
`rrmovq/cmovCC` *rA*, *rB*	2	*cc*	*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*							
OP`q` *rA*, *rB*	6	*fn*	*rA*	*rB*								
`j`*CC* *Dest*	7	*cc*	*Dest*									
`call` *Dest*	8	0	*Dest*									
`ret`	9	0										
`pushq` *rA*	A	0	*rA*	F								
`popq` *rA*	B	0	*rA*	F								

least sig. 4 bits of byte 0: bits 0–4

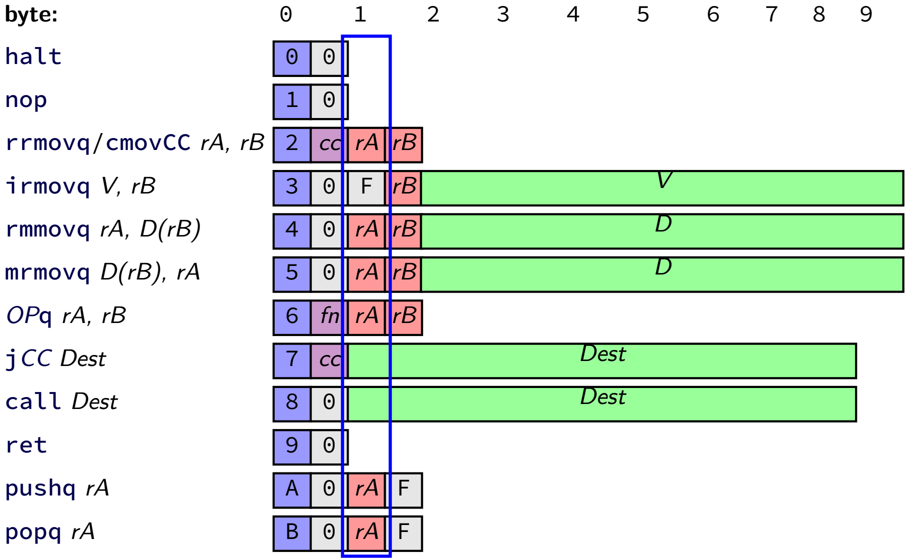
Y86 encoding table



byte:	0		1		2	3	4	5	6	7	8	9
`halt`	0	0										
`nop`	1	0										
`rrmovq/cmovCC` *rA, rB*	2	*cc*	*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*							
OP`q` *rA, rB*	6	*fn*	*rA*	*rB*								
`j`*CC Dest*	7	*cc*	*Dest*									
`call` *Dest*	8	0	*Dest*									
`ret`	9	0										
`pushq` *rA*	A	0	*rA*	F								
`popq` *rA*	B	0	*rA*	F								

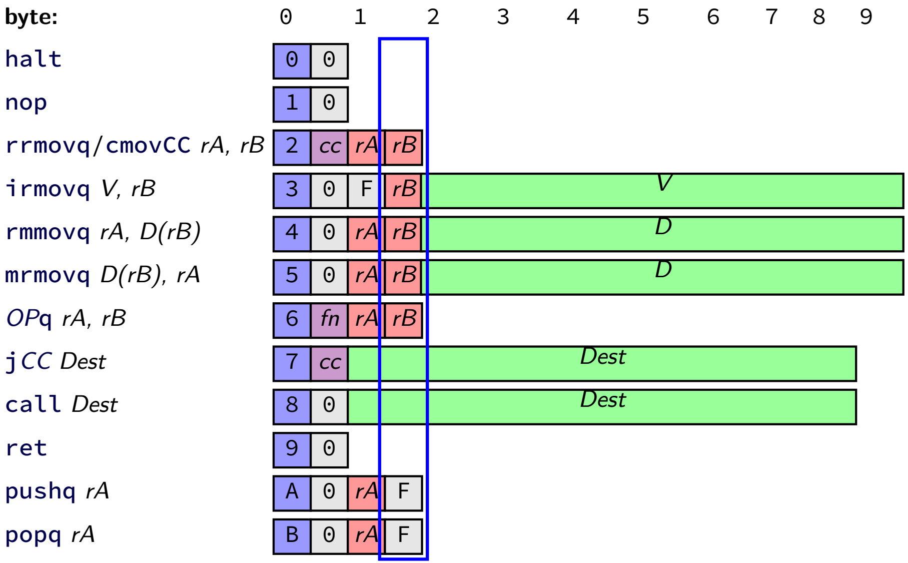
most sig. 4 bits of byte 0: bits 4–8

Y86 encoding table



most sig. 4 bits of byte 1: bits 12–16

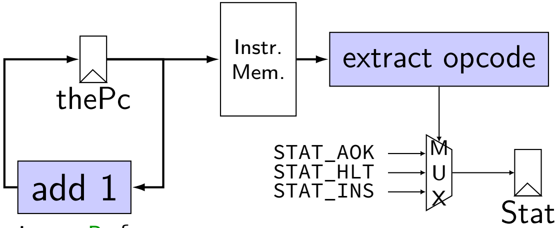
Y86 encoding table



byte:	0		1		2 3 4 5 6 7 8 9
`halt`	0	0			
`nop`	1	0			
`rrmovq/cmovCC` *rA*, *rB*	2	*cc*	*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*
OP`q` *rA*, *rB*	6	*fn*	*rA*	*rB*	
`j`*CC* *Dest*	7	*cc*	*Dest*		
`call` *Dest*	8	0	*Dest*		
`ret`	9	0			
`pushq` *rA*	A	0	*rA*	F	
`popq` *rA*	B	0	*rA*	F	

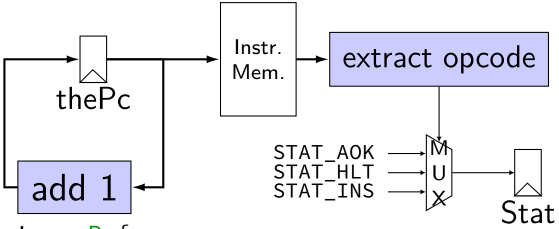
least sig. 4 bits of byte 1: bits 8–12

nop/halt CPU



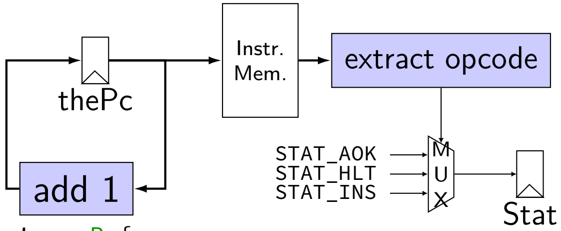
```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

nop/halt CPU



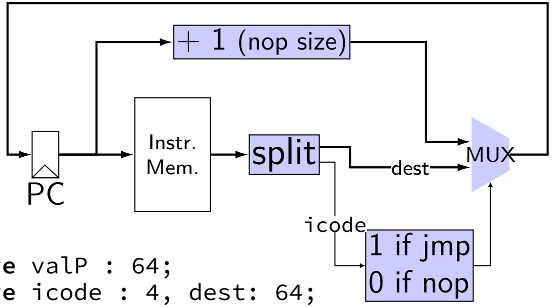
```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

nop/halt CPU



```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

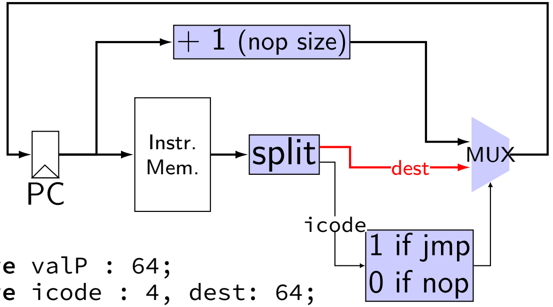
nop/jmp CPU



```
wire valP : 64;
wire icode : 4, dest: 64;
register pP {
    thePc : 64 = 0;
}
icode = i10bytes[4..8];
dest = i10bytes[8..72];
valP = [
    icode == NOP : P_thePc + 1;
    icode == JXX : dest;
    1: 0xBADBADBAD;
];
p_thePc = valP;
pc = P_thePc;
```

```
Stat = [
    (icode == NOP ||
     icode == JXX) : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

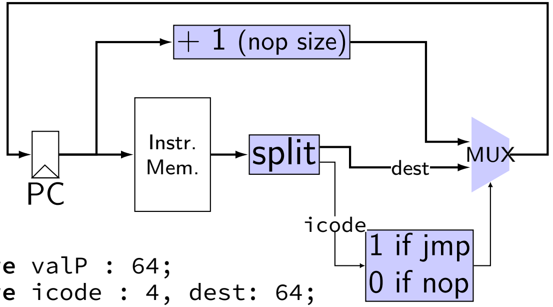
nop/jmp CPU



```
wire valP : 64;
wire icode : 4, dest: 64;
register pP {
    thePc : 64 = 0;
}
icode = i10bytes[4..8];
dest = i10bytes[8..72];
valP = [
    icode == NOP : P_thePc + 1;
    icode == JXX : dest;
    1: 0xBADBADBAD;
];
p_thePc = valP;
pc = P_thePc;
```

```
Stat = [
    (icode == NOP ||
     icode == JXX) : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

nop/jmp CPU



```
wire valP : 64;
wire icode : 4, dest: 64;
register pP {
    thePc : 64 = 0;
}
icode = i10bytes[4..8];
dest = i10bytes[8..72];
valP = [
    icode == NOP : P_thePc + 1;
    icode == JXX : dest;
    1: 0xBADBADBAD;
];
p_thePc = valP;
pc = P_thePc;
```

```
Stat = [
    (icode == NOP ||
     icode == JXX) : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

running nop/jmp/halt

`nopjmp.ys`:

```
     nop
     jmp C
B:   jmp D
C:   jmp B
D:   nop
     nop
     halt
```

…assemble with `yas`

nopjmp.yo

`nopjmp.yo`:

```
0x000: 10                         |       nop
0x001: 7013000000000000000        |       jmp C
0x00a: 701c000000000000000        |  B:   jmp D
0x013: 700a000000000000000        |  C:   jmp B
0x01c: 10                         |  D:   nop
0x01d: 10                         |       nop
0x01e: 00                         |       halt
```

nopjmp.yo

nopjmp.yo:

```
0x000: 10                         |       nop
0x001: 7013000000000000000        |       jmp C
0x00a: 701c000000000000000        | B:    jmp D
0x013: 700a000000000000000        | C:    jmp B
0x01c: 10                         | D:    nop
0x01d: 10                         |       nop
0x01e: 00                         |       halt
```

running nopjmp.yo

```
$ ./hclrs nopjmp_cpu.hcl nopjmp.yo
...
...
+--------------------- (end of halted state) ---------------------------+
Cycles run: 7
```