SEQ part 2

Changelog

22 September: mux exercise: change values

22 September: what is i10bytes?: fixup hilite on memory diagram with pc=0x1

22 September: addq cpu intro: say “2 byte” not “1 byte” instructions

last time

Y86-64 decoding

HCLs

wires and bundles of wires

- short-hand: wire for really a bundle

registers

- store value, change on rising clock edges

instruction memory

nop CPU

- single-cycle CPU — each (nop) instruction takes one cycle
- compute PC for next cycle

lab grading note

originally autograder gave wrong feedback on relocation for array

`irmovq $array, %rax` has the addres in its second byte

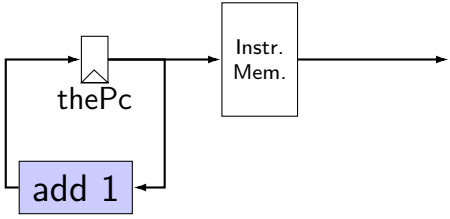
so relocation should point two bytes into the instruction

quiz 3 grading note

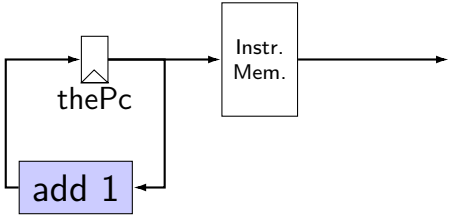
I am aware that quiz 3 regrades + comment handling is not done

(an internal course staff communication issue...)

nop CPU

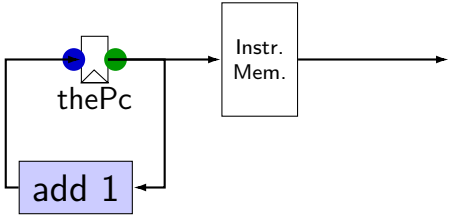


nop CPU



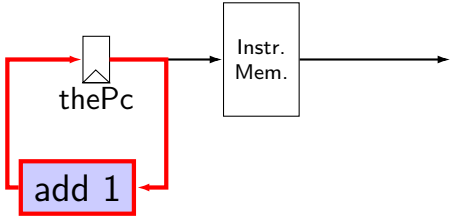

```
register pF {
    thePc : 64 = 0;
}
```

nop CPU



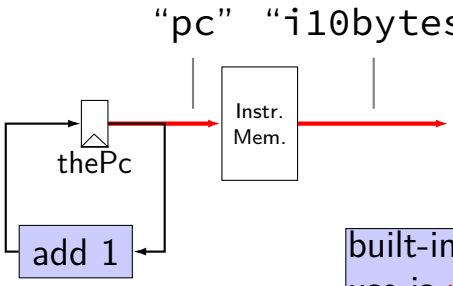

```
register pF {
    thePc : 64 = 0;
}
```

nop CPU




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

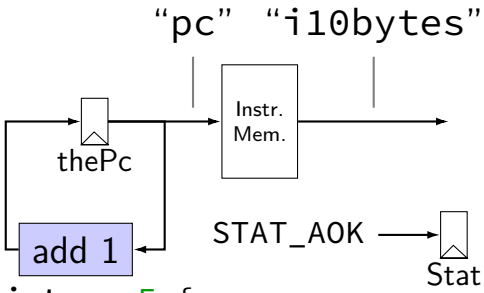
nop CPU



built-in component use is mandatory

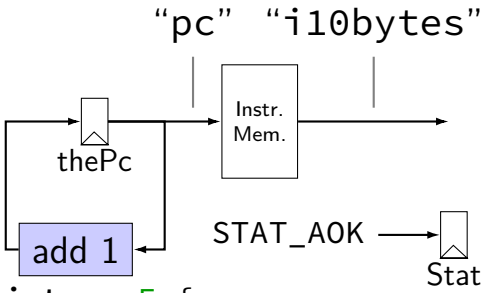
```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```

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]
....
+------------ timed out after  9999 cycles in state: -------------------+
| 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=000000000000270f                                |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                        |
+-----------------------------------------------------------------------+
```

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]
....
+------------ timed out after  9999 cycles in state: -------------------+
| 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=000000000000270f                                |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                        |
+-----------------------------------------------------------------------+
```

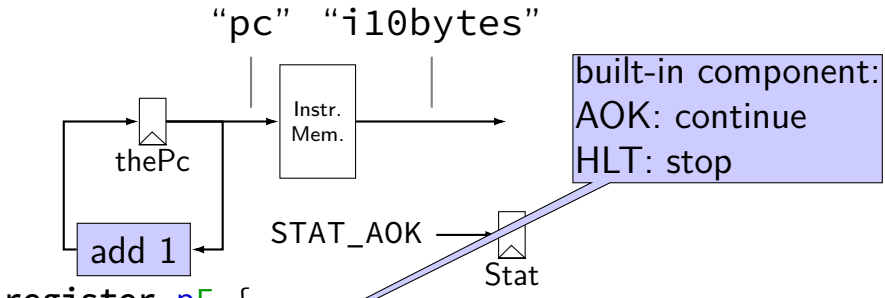
running a simulator (2)

```
$ ./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]
....
+------------ timed out after  9999 cycles in state: -------------------+
| 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=000000000000270f                                |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                        |
+-----------------------------------------------------------------------+
```

running a simulator (2)

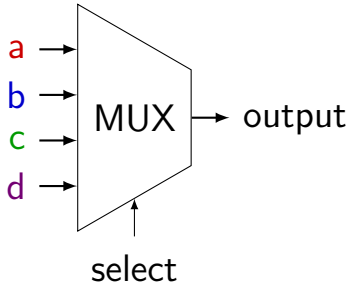
```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:                0   RCX:                0   RDX:                0 |
| RBX:                0   RSP:                0   RBP:                0 |
| RSI:                0   RDI:                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]
....
+------------ timed out after  9999 cycles in state: -------------------+
| 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=000000000000270f                                |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 10 10 10  10                                        |
+-----------------------------------------------------------------------+
```

nop CPU

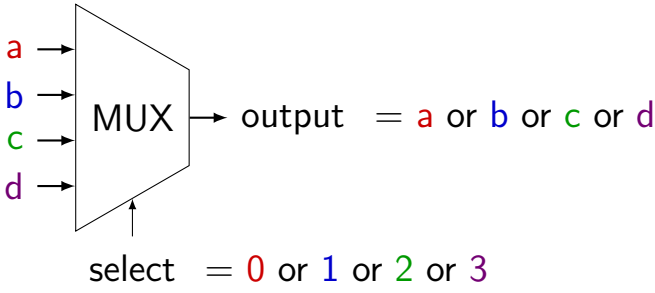



```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
Stat = STAT_AOK;
```

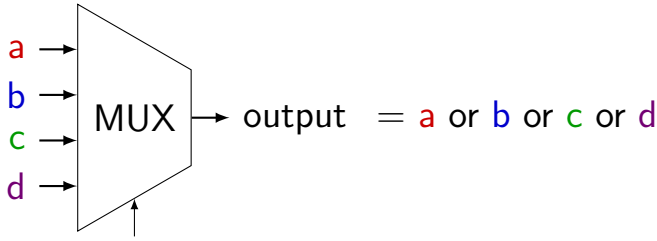
multiplexers



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

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

MUX exercise

```
foo = [
    bar > 10 : 100;
    (bar & 1) == 1 : 200;
    bar < 20 : 300;
    1 : 400;
]
```

exercise 1: if `bar` is 9, what is foo?
exercise 2: if `bar` is 10, what is foo?
exercise 3: if `bar` is 11, what is foo?

Simple ISA: nop/halt CPU

nop

encoding `10`

halt

encoding `00`

Simple ISA: nop/halt CPU

nop

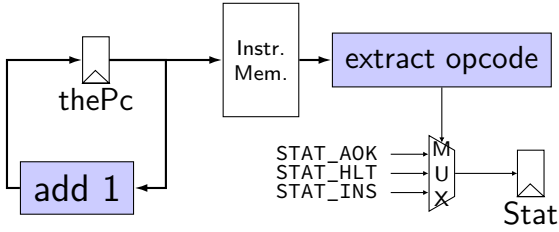
encoding `10`

halt

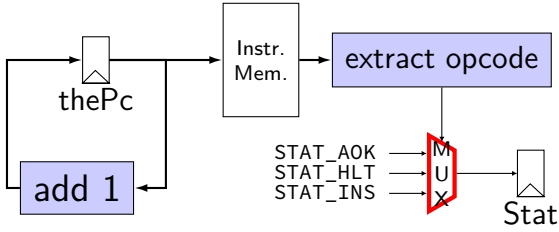
encoding `00`

our strategy: MUX to decide using opcode

nop/halt CPU

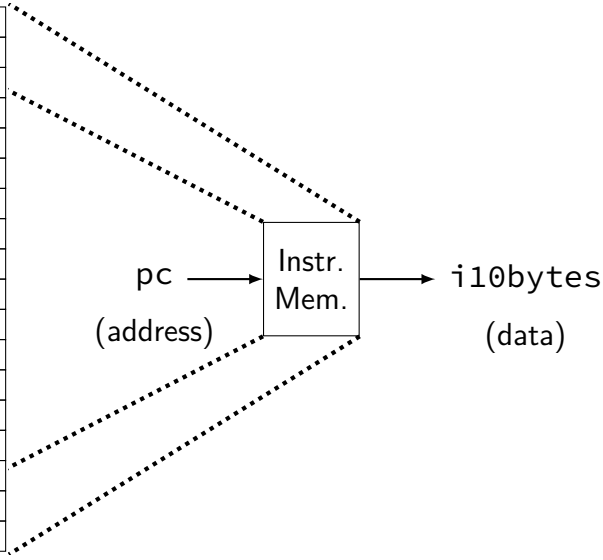

thePc
Instr.
Mem.
extract opcode
STAT_AOK
STAT_HLT
STAT_INS
M
U
X
Stat
add 1

nop/halt CPU


thePc
add 1
Instr.
Mem.
extract opcode
STAT_AOK
STAT_HLT
STAT_INS
M
U
X
Stat

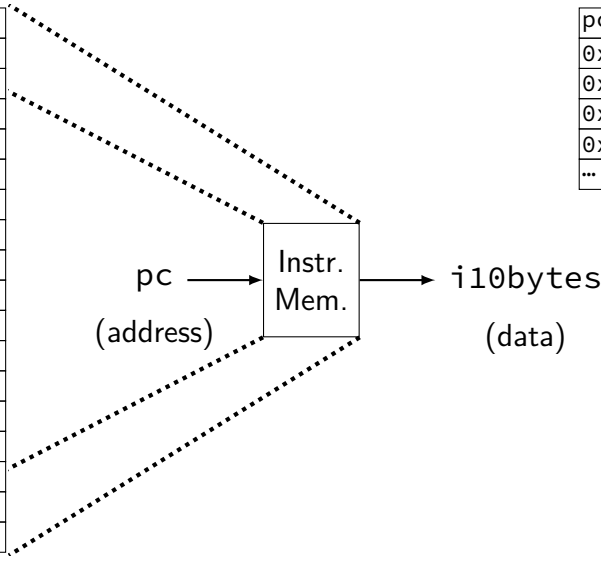
what is i10bytes?

addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



what is i10bytes?

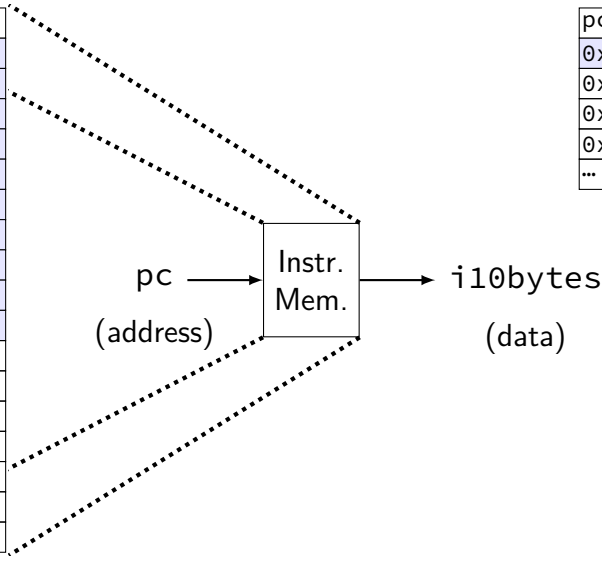
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x01000000000021611260
0x001	0x00010000000000216112
0x002	0x00000100000000002161
0x003	0x00000001000000000021
…	…

what is i10bytes?

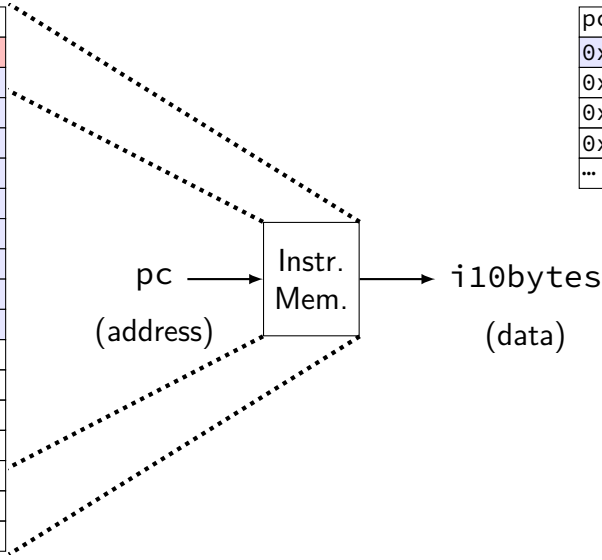
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x01000000000021611260
0x001	0x00010000000000216112
0x002	0x00000100000000002161
0x003	0x00000001000000000021
…	…

what is i10bytes?

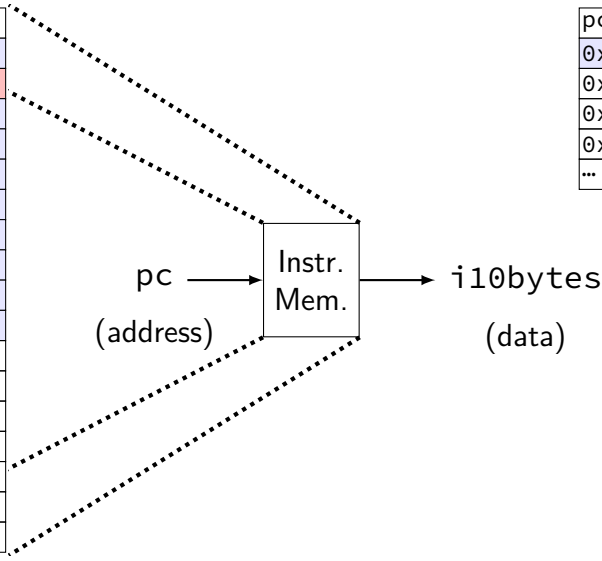
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000021611260
0x001	0x0001000000000216112
0x002	0x0000100000000002161
0x003	0x0000000100000000021
…	…

what is i10bytes?

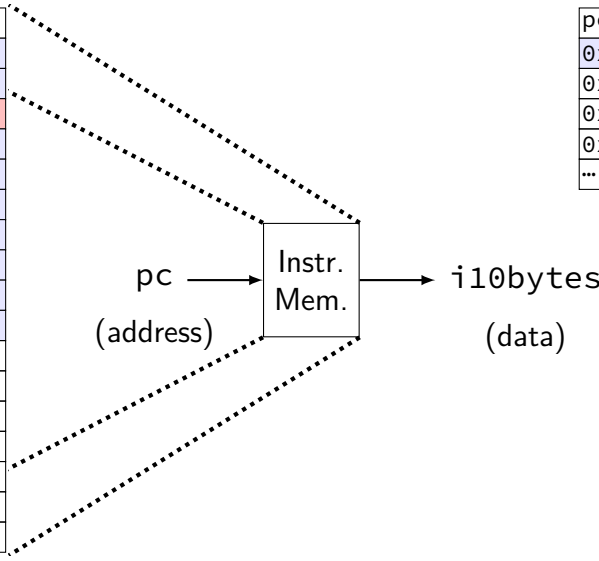
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000002161**12**60
0x001	0x0001000000000216112
0x002	0x0000100000000002161
0x003	0x0000000100000000021
…	…

what is i10bytes?

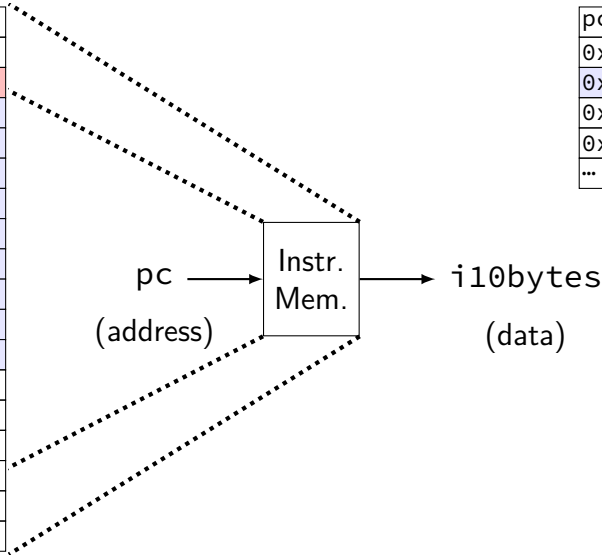
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000002161 1260
0x001	0x0001000000000216112
0x002	0x0000100000000002161
0x003	0x0000000100000000021
…	…

what is i10bytes?

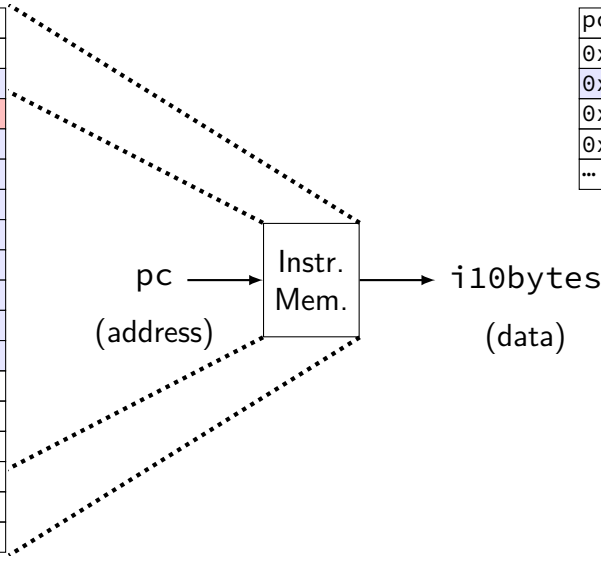
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000021611260
0x001	0x0001000000000021611 2
0x002	0x0000010000000000 2161
0x003	0x0000000100000000 0021
…	…

what is i10bytes?

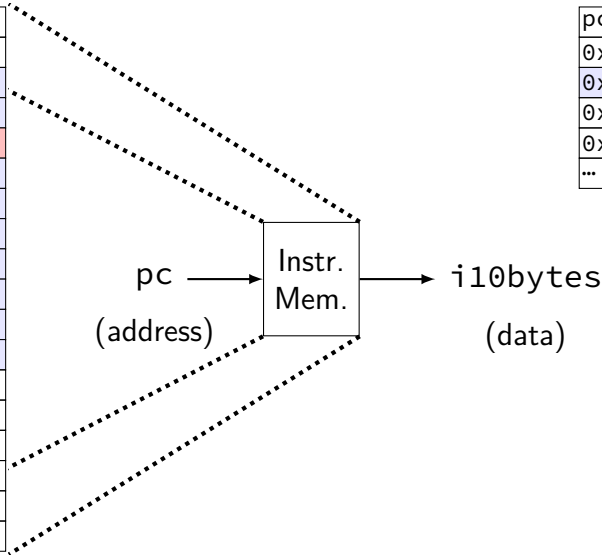
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000021611260
0x001	0x0001000000000021612
0x002	0x0000100000000002161
0x003	0x0000000100000000021
…	…

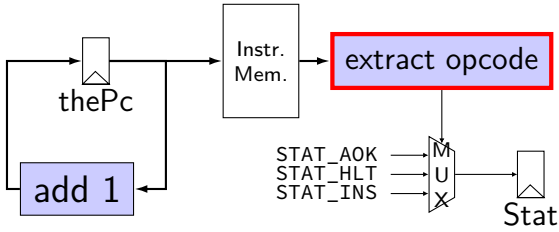
what is i10bytes?

addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x0100000000021611260
0x001	0x0001000000000021 6112
0x002	0x0000100000000002161
0x003	0x0000000100000000021
…	…

nop/halt CPU


thePc
Instr.
Mem.
extract opcode
STAT_AOK
STAT_HLT
STAT_INS
M
U
X
Stat
add 1

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

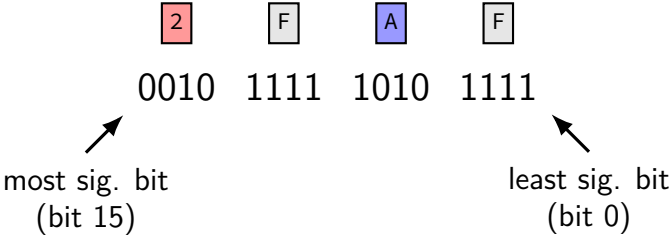
example

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

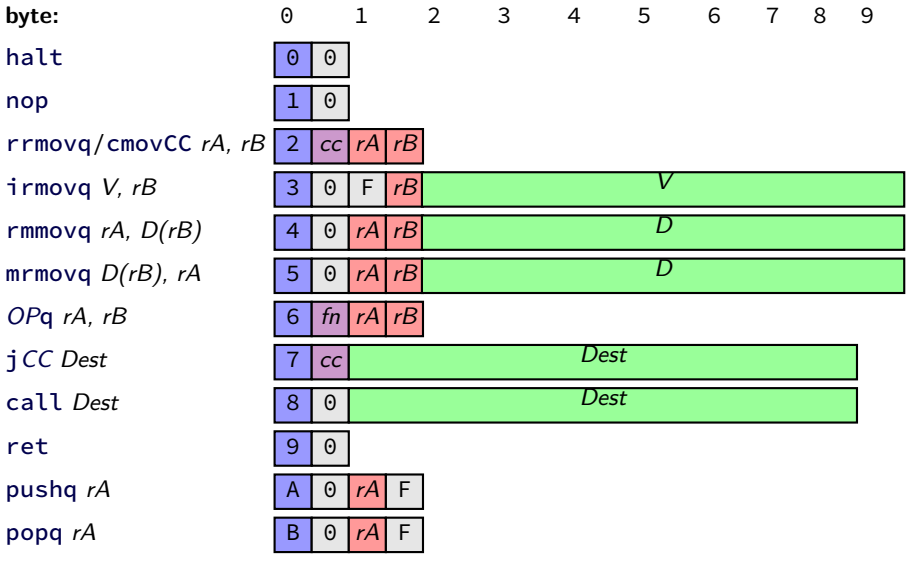
memory at $x + 0$: pushq F; at $x + 1$: rbx F

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

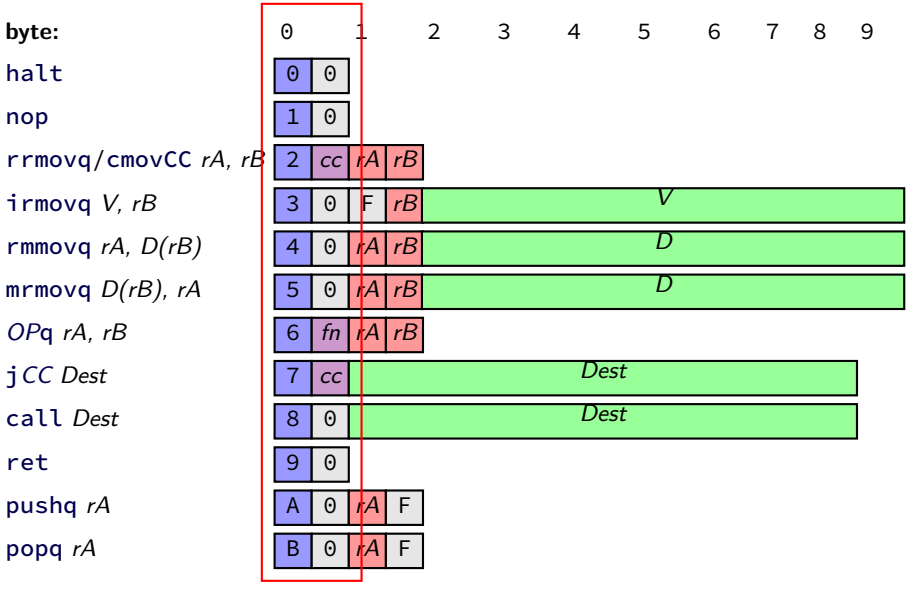
as a little-endian 2-byte number in typical English order:



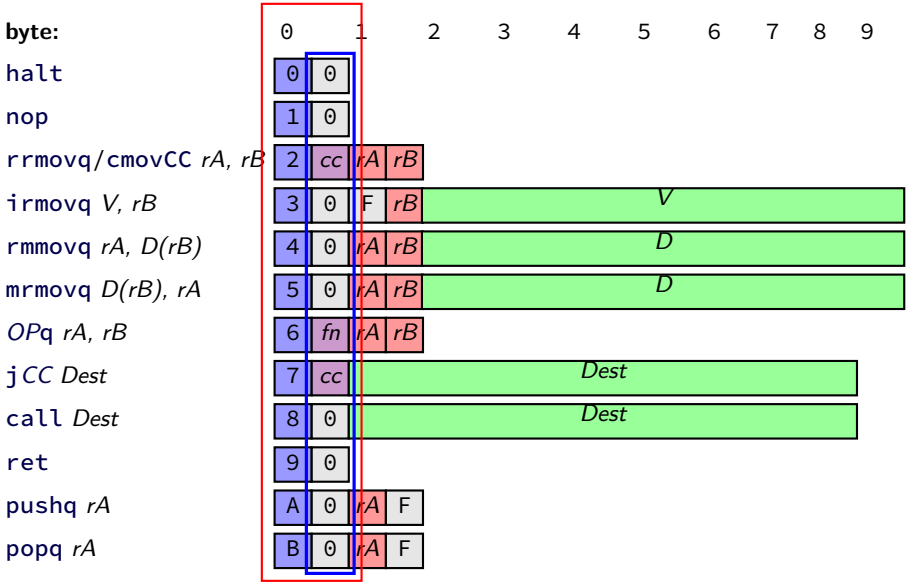
Y86 encoding table



Y86 encoding table

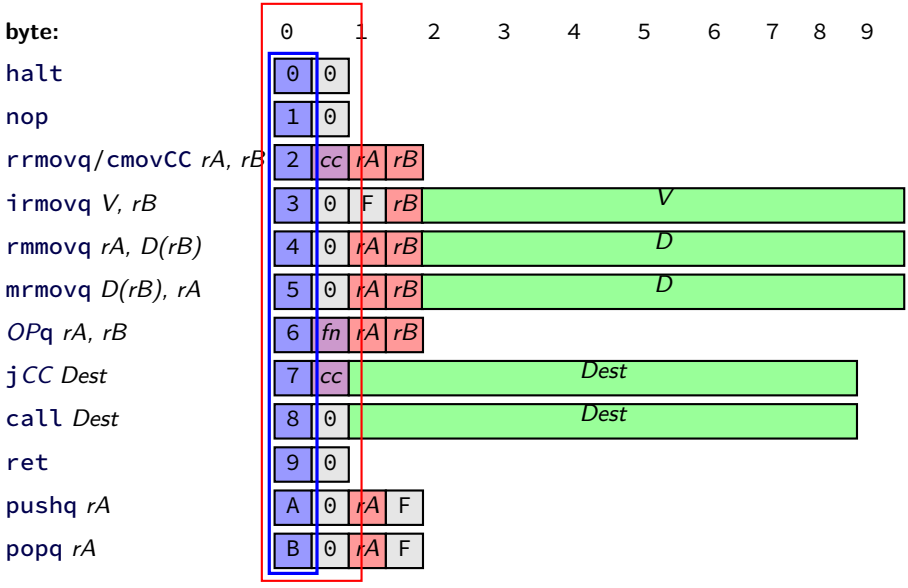


Y86 encoding table



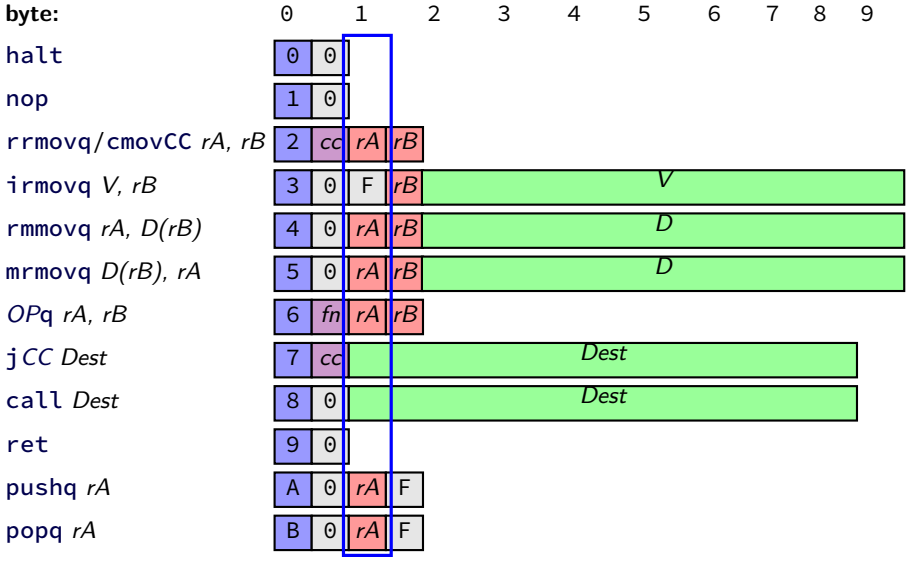
least sig. 4 bits of byte 0: bits 0–4

Y86 encoding table



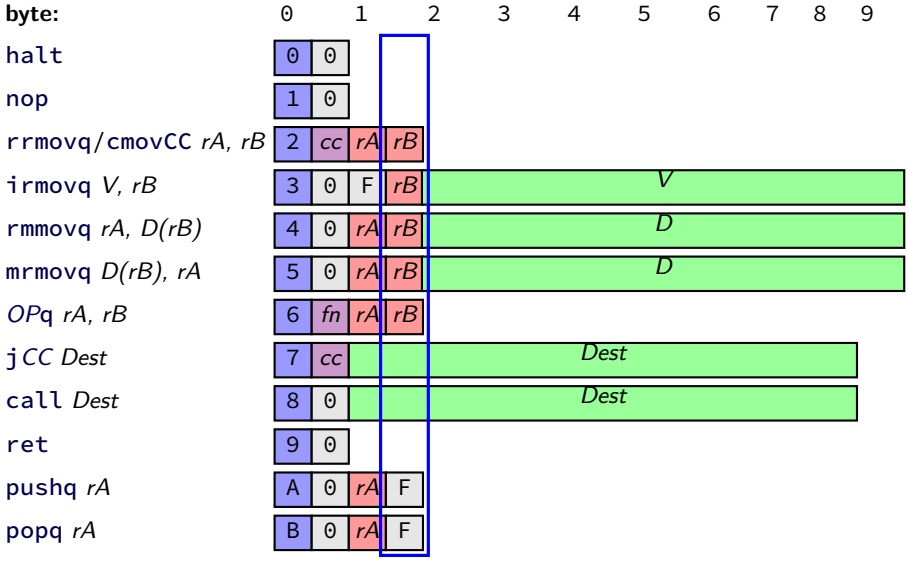
most sig. 4 bits of byte 0: bits 4–8

Y86 encoding table



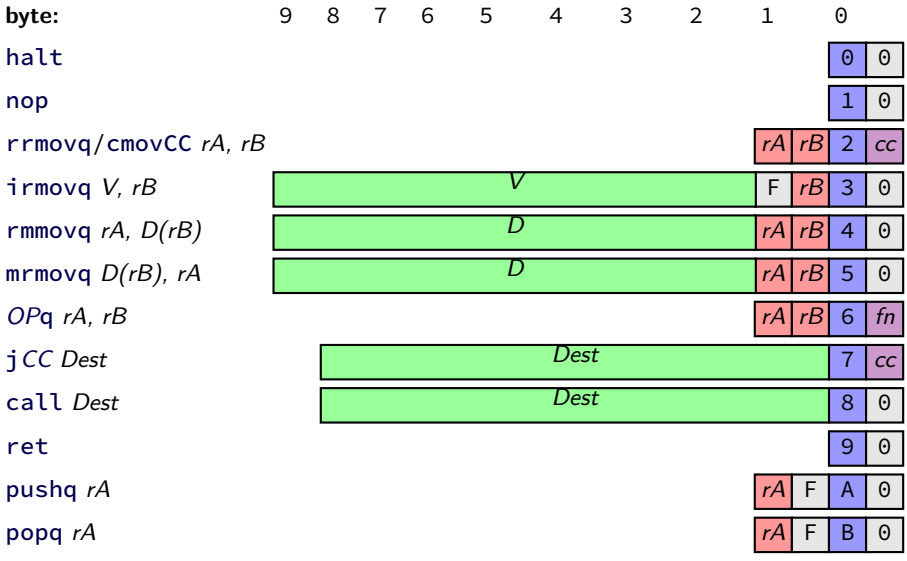
most sig. 4 bits of byte 1: bits 12–16

Y86 encoding table

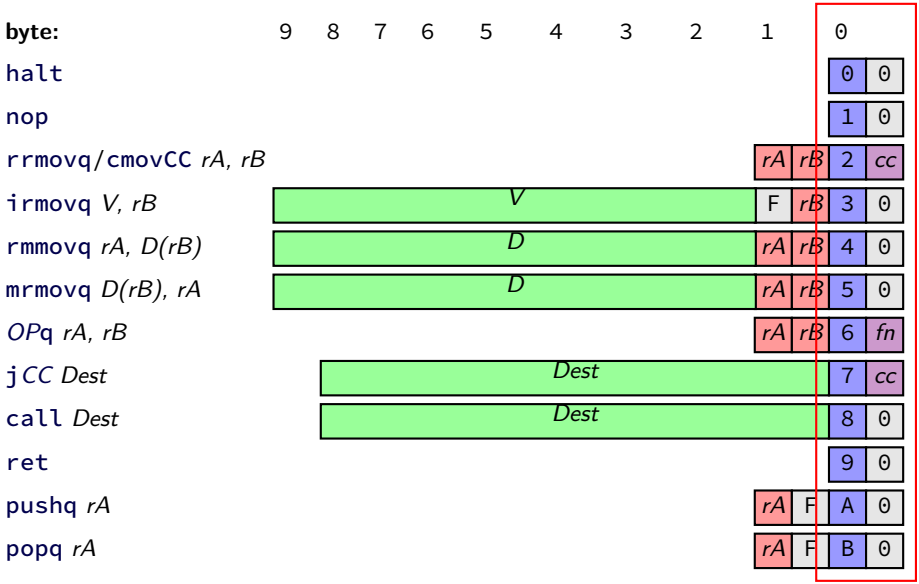


least sig. 4 bits of byte 1: bits 8–12

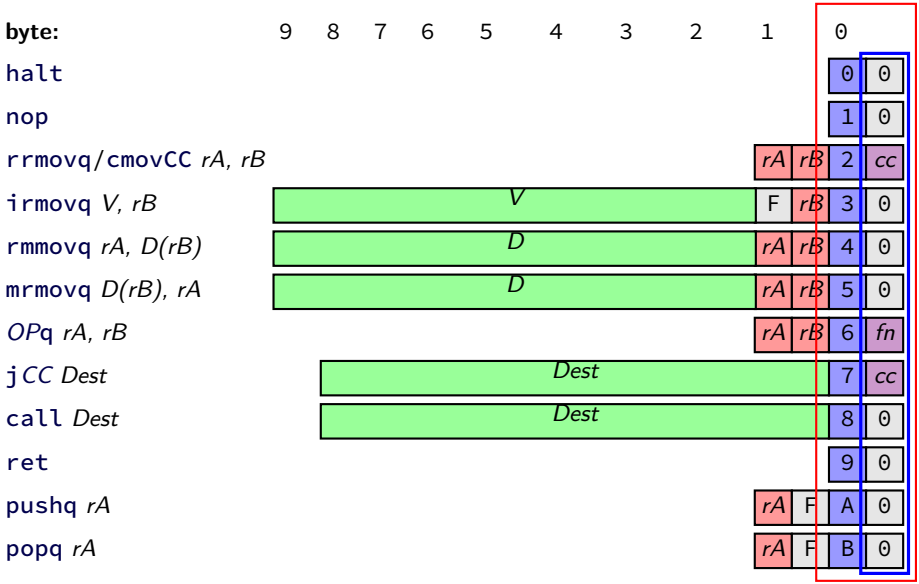
Y86 encoding table (written differently)



Y86 encoding table (written differently)

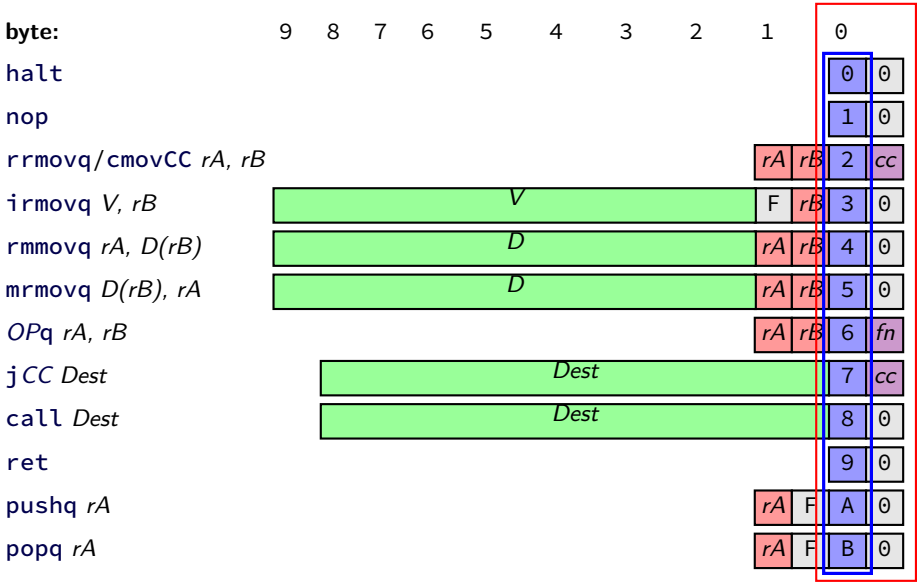


Y86 encoding table (written differently)



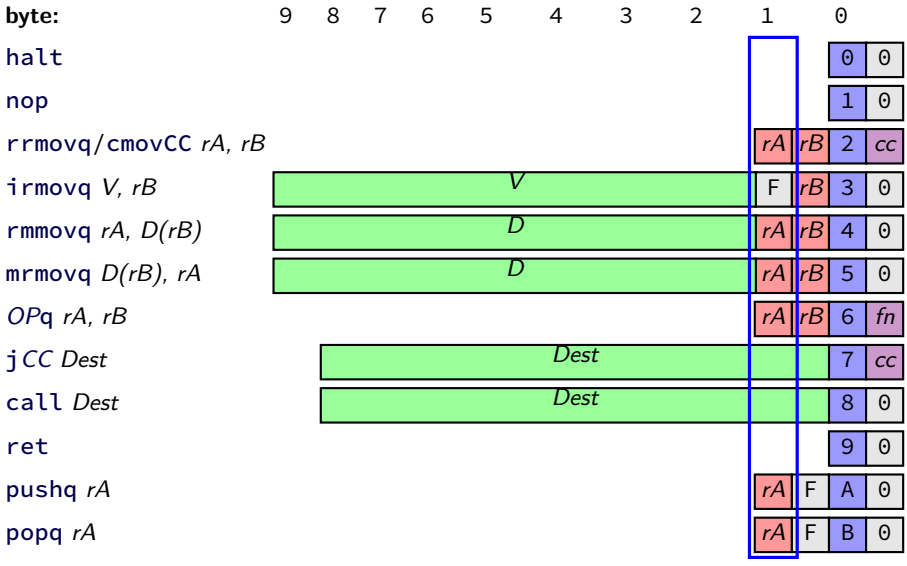
least sig. 4 bits of byte 0: bits 0–4

Y86 encoding table (written differently)



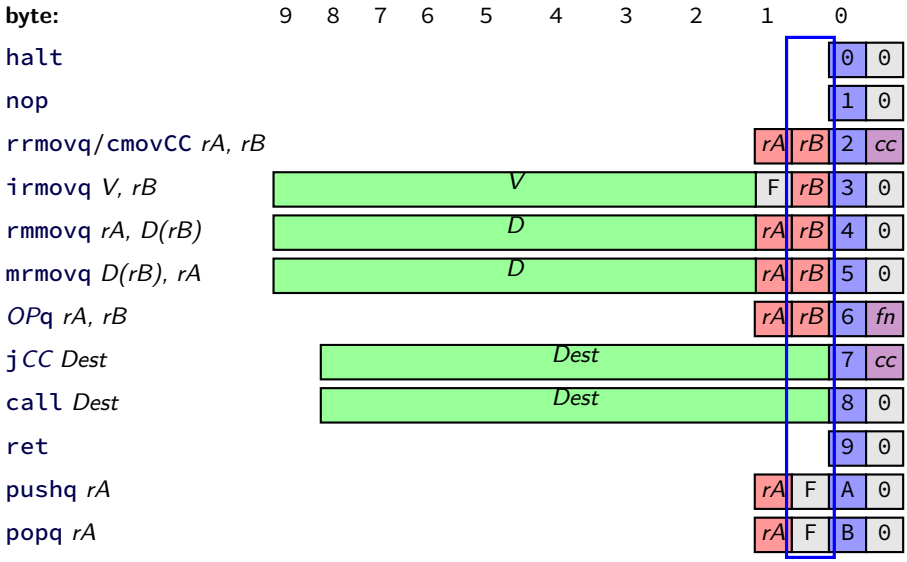
most sig. 4 bits of byte 0: bits 4–8

Y86 encoding table (written differently)



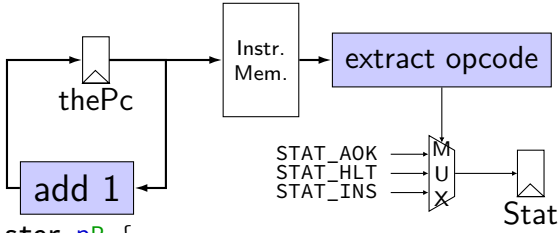
most sig. 4 bits of byte 1: bits 12–16

Y86 encoding table (written differently)



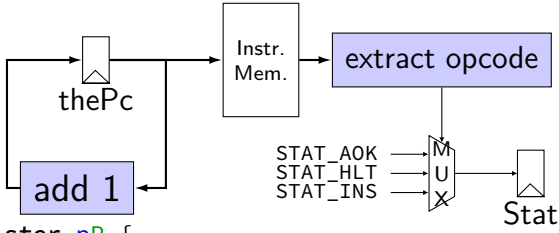
least sig. 4 bits of byte 1: bits 8–12

nop/halt CPU



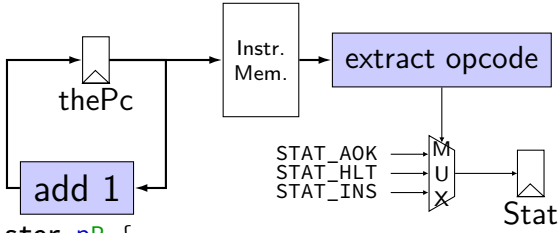

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

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)
];
```

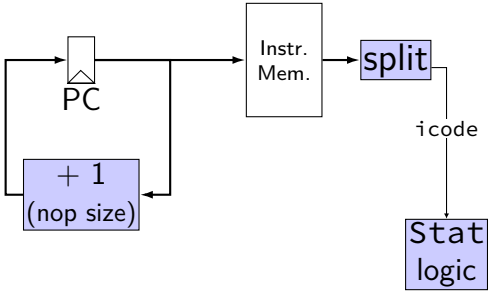
nop/halt CPU



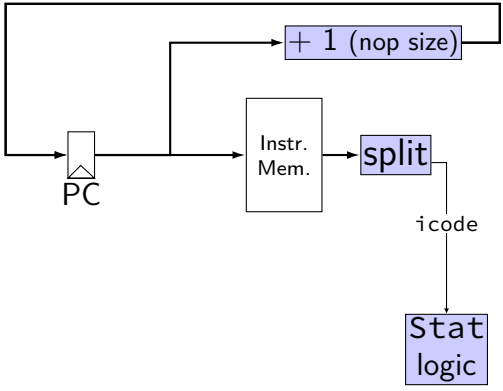

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

demo

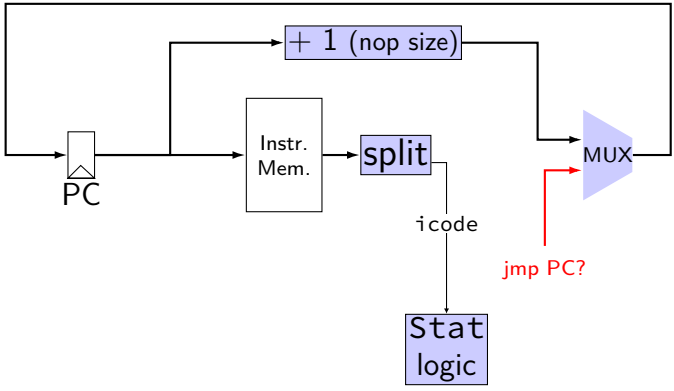
nop/halt → nop/jmp CPU



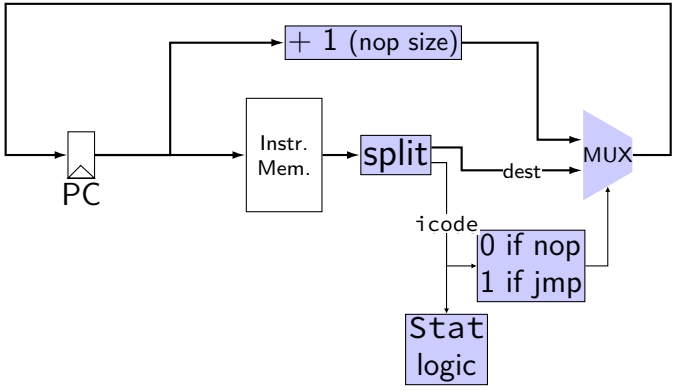
nop/halt → nop/jmp CPU



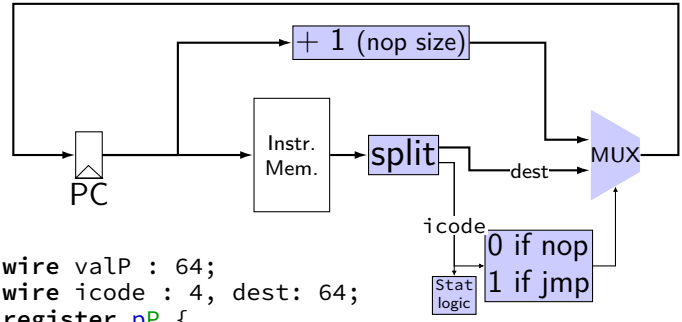
nop/halt → nop/jmp CPU



nop/halt → nop/jmp CPU



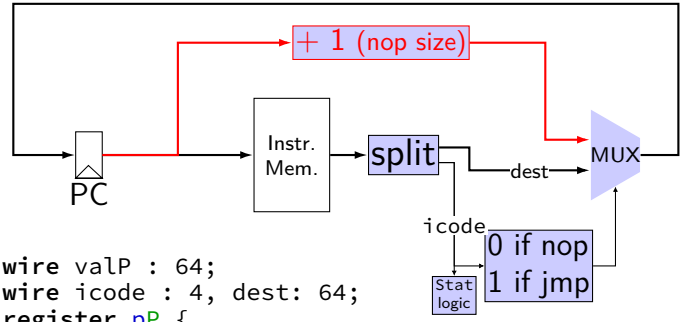
nop/jmp CPU




```
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;
];
```

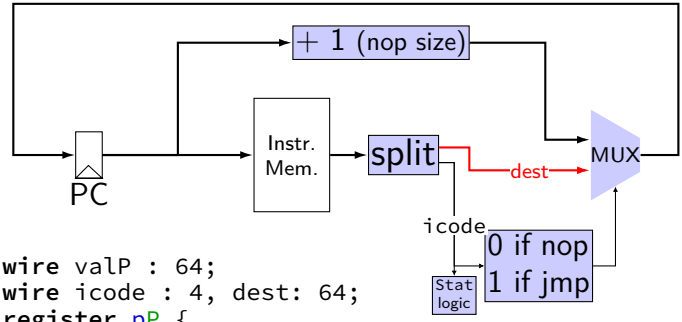
nop/jmp CPU




```
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;
];
```

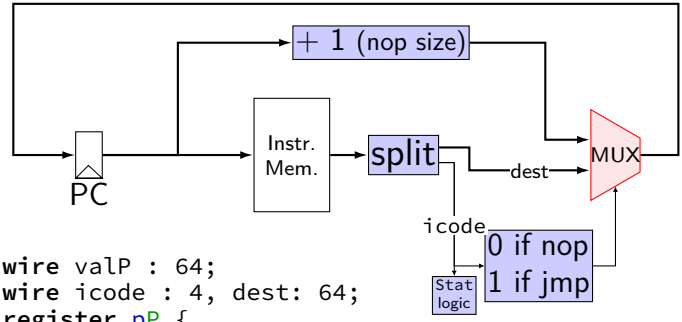
nop/jmp CPU




```
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;
];
```

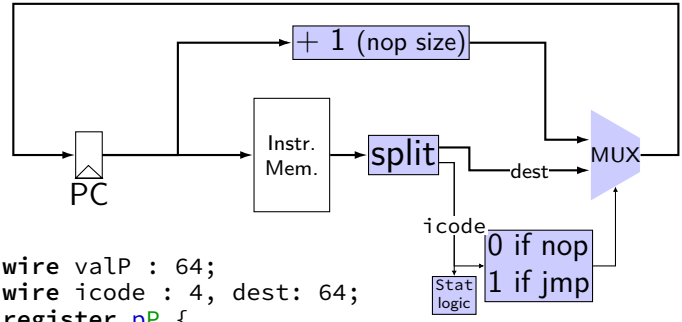
nop/jmp CPU




```
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;
];
```

demo: running nop/jmp

demo: debug and interactive mode

demo: yis

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: 701c00000000000000   | B:   jmp D
0x013: 700a00000000000000   | 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: 701c00000000000000   | B:   jmp D
0x013: 700a00000000000000   | 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
```

simple ISA: addq

`addq %rXX, %rYY`

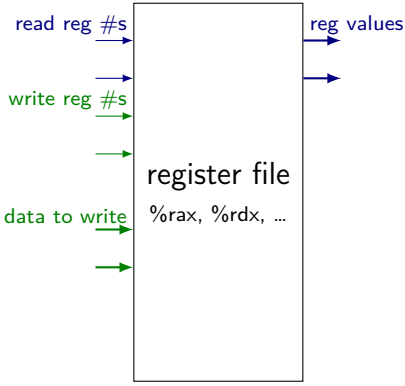
encoding: `6` `0` `%rXX` `%rYY` (two 4-bit register #s)

- 2 byte instructions, no opcode

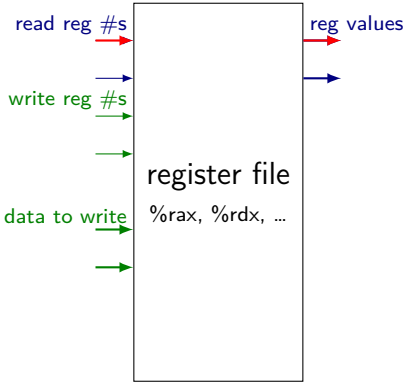
for now: no other instructions

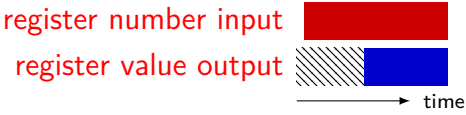
- later: adding support for nop+halt

register file

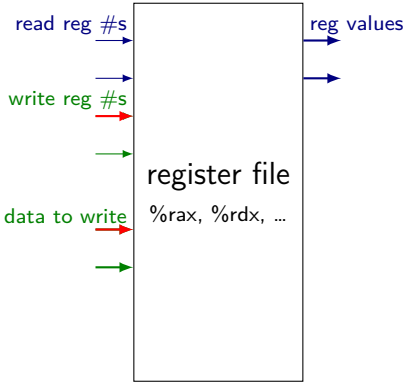


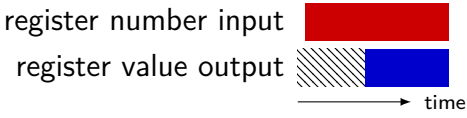
register file

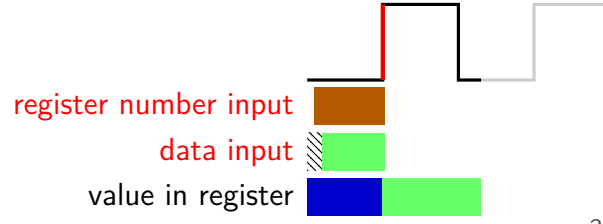




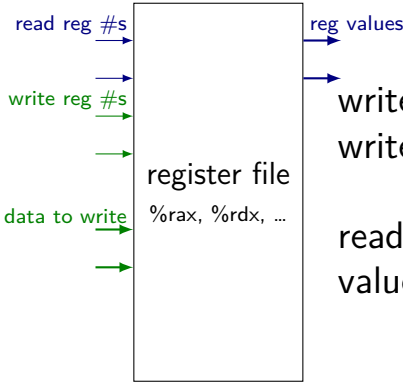
register file





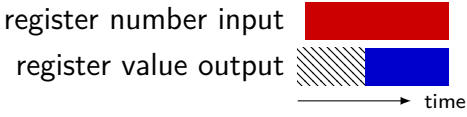


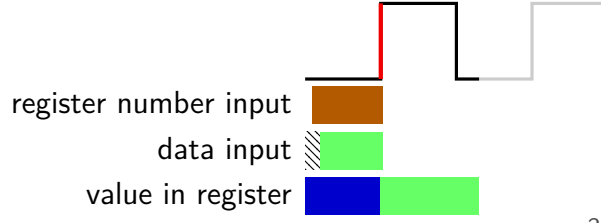
register file



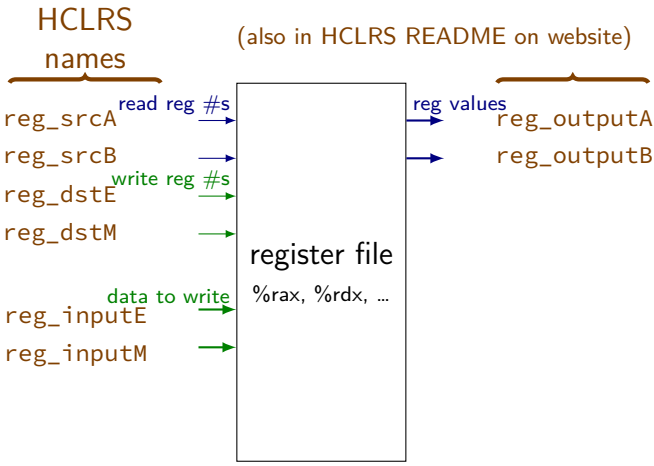
write register #15 (REG_NONE):
write is ignored

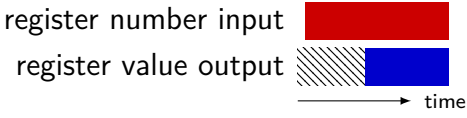
read register #15 (REG_NONE):
value is always 0

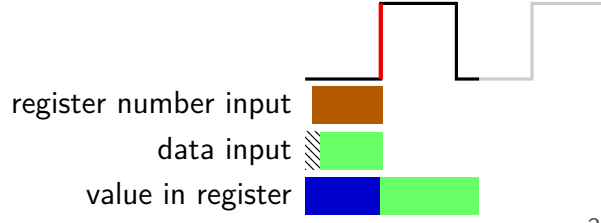




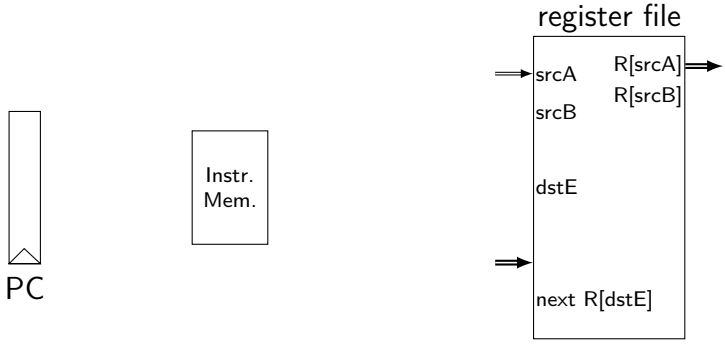
register file



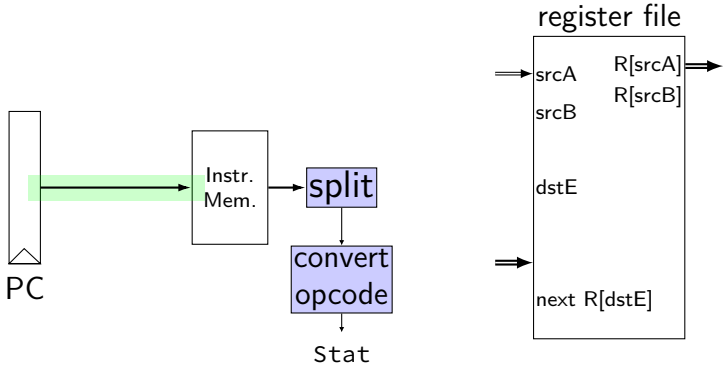




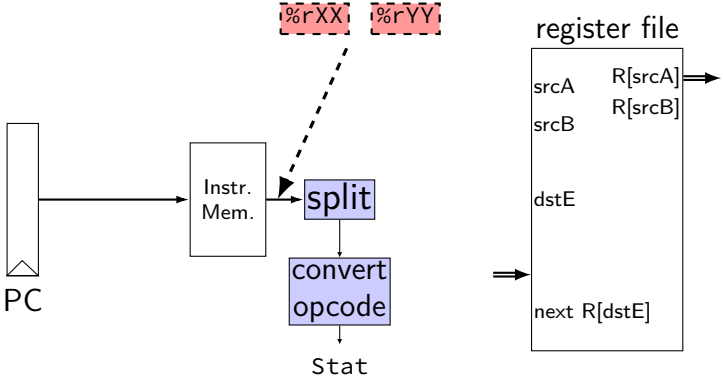
addq CPU


register file
srcA
srcB
R[srcA]
R[srcB]
dstE
next R[dstE]
Instr.
Mem.
PC

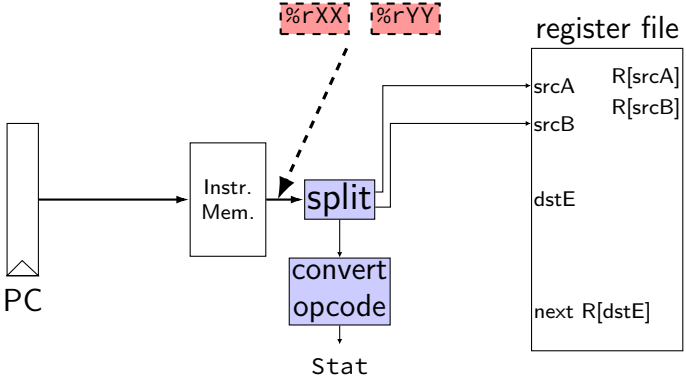
addq CPU


register file
srcA
srcB
R[srcA]
R[srcB]
dstE
next R[dstE]
PC
Instr.
Mem.
split
convert
opcode
Stat

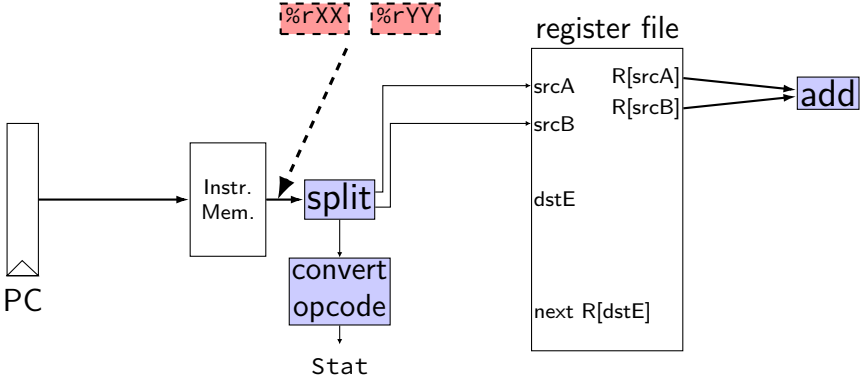
addq CPU


%rXX
%rYY
register file
srcA
srcB
R[srcA]
R[srcB]
dstE
next R[dstE]
Instr.
Mem.
split
convert
opcode
Stat
PC

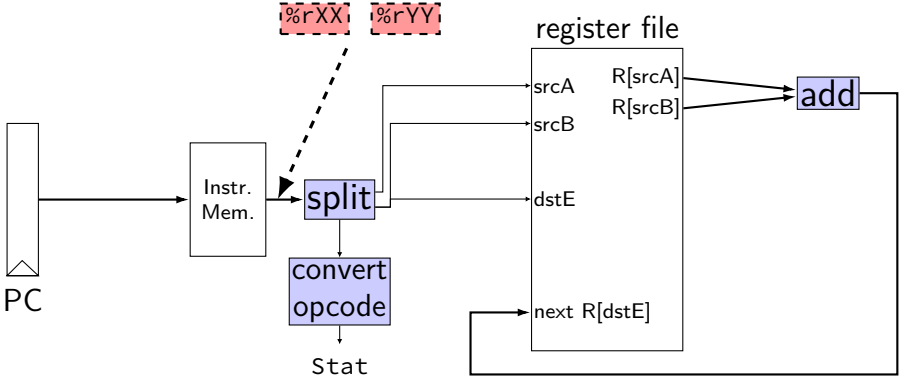
addq CPU


%rXX
%rYY
register file
srcA
srcB
R[srcA]
R[srcB]
dstE
next R[dstE]
Instr.
Mem.
split
convert
opcode
PC
Stat

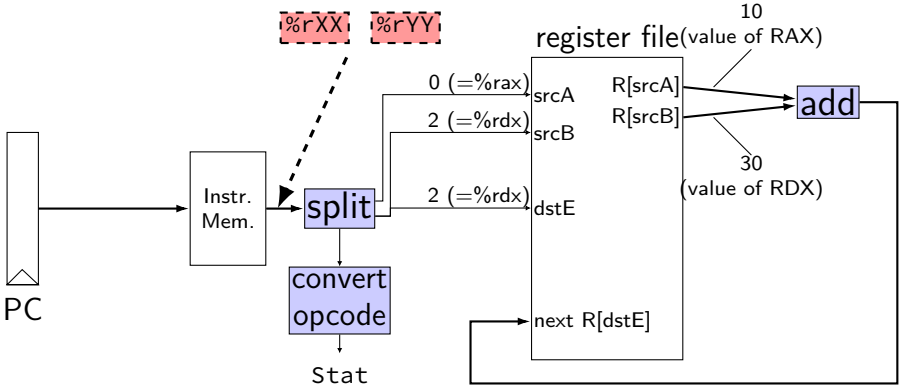
addq CPU


%rXX
%rYY
register file
srcA
srcB
dstE
next R[dstE]
R[srcA]
R[srcB]
add
Instr.
Mem.
split
convert
opcode
Stat
PC

addq CPU


%rXX
%rYY
register file
srcA
srcB
dstE
R[srcA]
R[srcB]
add
Instr.
Mem.
split
convert
opcode
Stat
PC
next R[dstE]

addq CPU

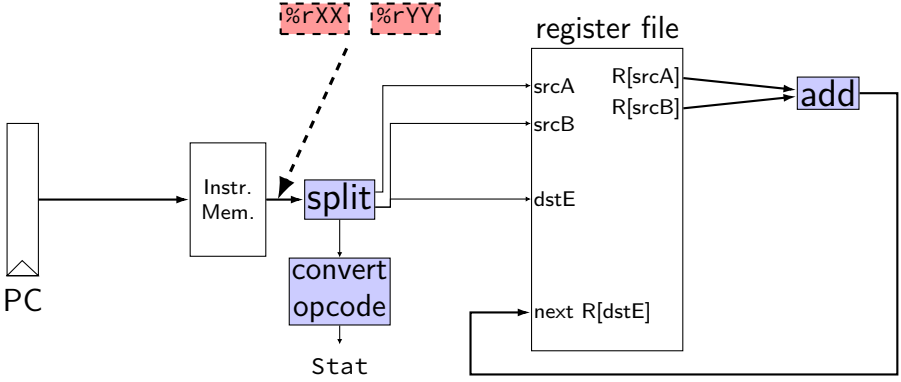



```
/* 0x00: */ addq %rax, %rdx
/* 0x02: */ addq %rbx, %rdx
```

initially: PC = 0x00, rax = 10, rbx = 20, rdx = 30

after cycle 1: PC = ????, rax = 10, rbx = 20, rdx = 40

addq CPU



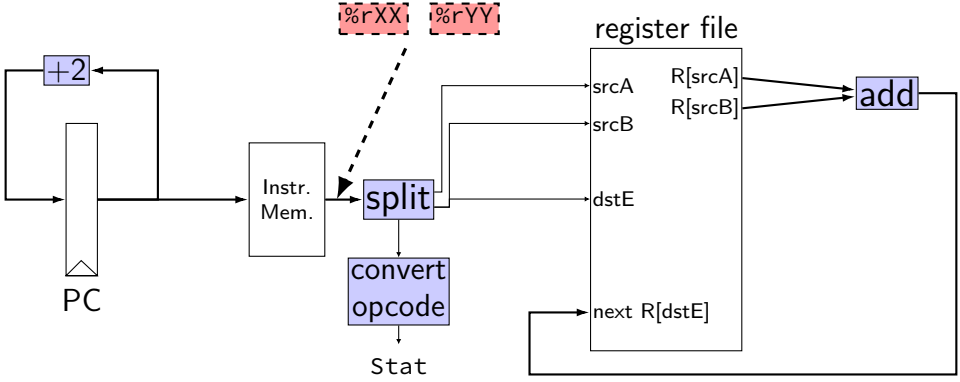

```
/* 0x00: */ addq %rax, %rdx
/* 0x02: */ addq %rbx, %rdx
```

initially: PC = 0x00, rax = 10, rbx = 20, rdx = 30

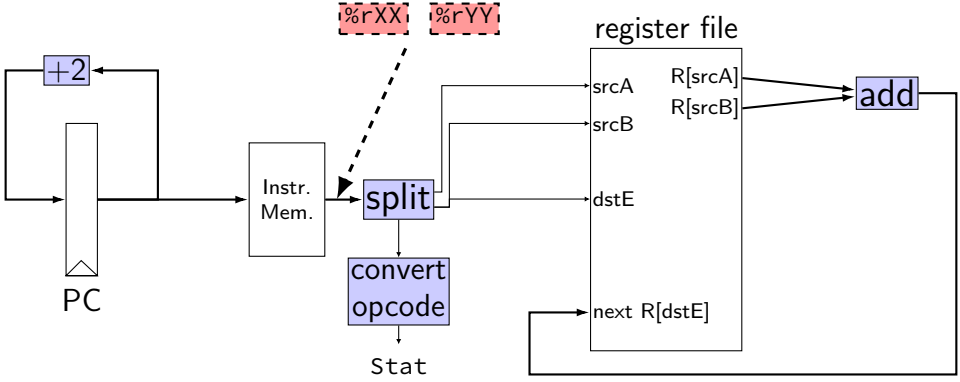
after cycle 1: PC = ????, rax = 10, rbx = 20, rdx = 40

after cycle 2: PC = ????, rax = ??, rbx = ??, rdx = ??

addq CPU


%rXX
%rYY
register file
+2
srcA
R[srcA]
R[srcB]
add
srcB
Instr.
Mem.
split
dstE
convert
opcode
PC
next R[dstE]
Stat

addq CPU



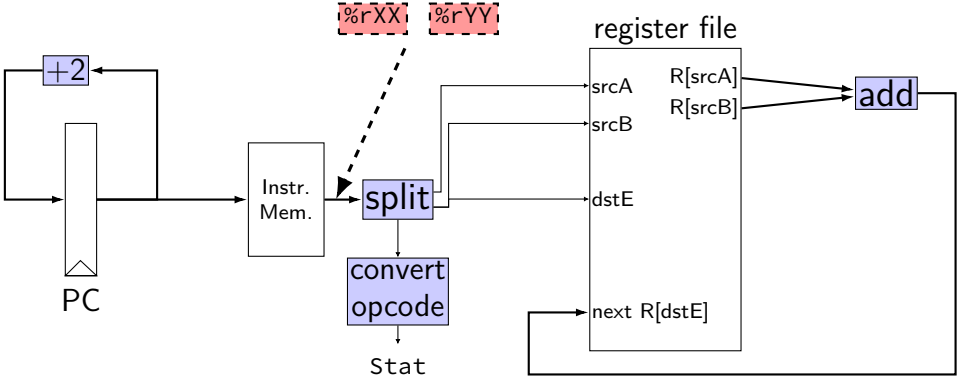

```
/* 0x00: */ addq %rax, %rdx
/* 0x02: */ addq %rbx, %rdx
```

initially: PC = 0x00, rax = 10, rbx = 20, rdx = 30

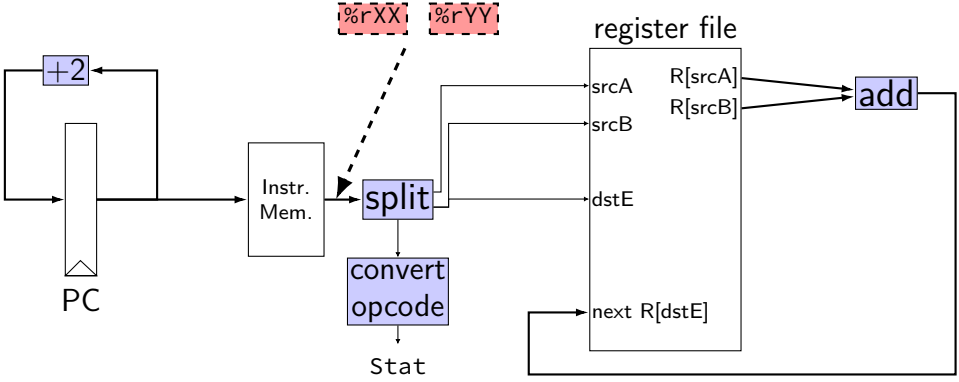
after cycle 1: PC = 0x02, rax = 10, rbx = 20, rdx = 40

after cycle 2: PC = 0x04, rax = 10, rbx = 20, rdx = 60

addq CPU: HCL


%rXX
%rYY
register file
+2
srcA
srcB
dstE
R[srcA]
R[srcB]
add
Instr.
Mem.
split
convert
opcode
PC
next R[dstE]
Stat

addq CPU: HCL

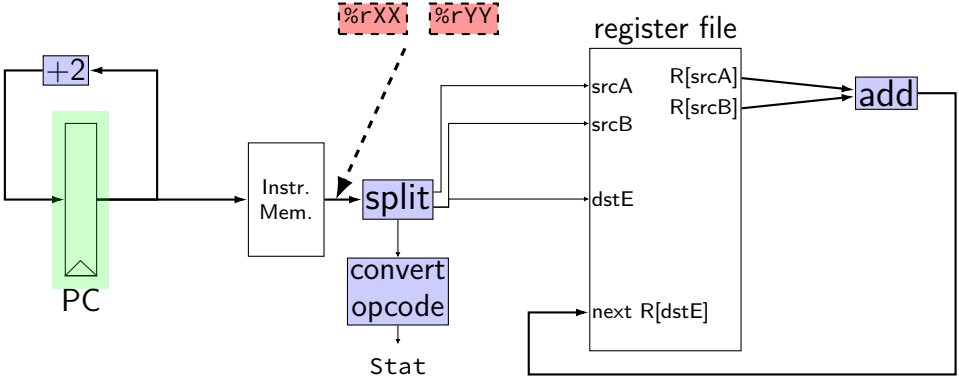



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL

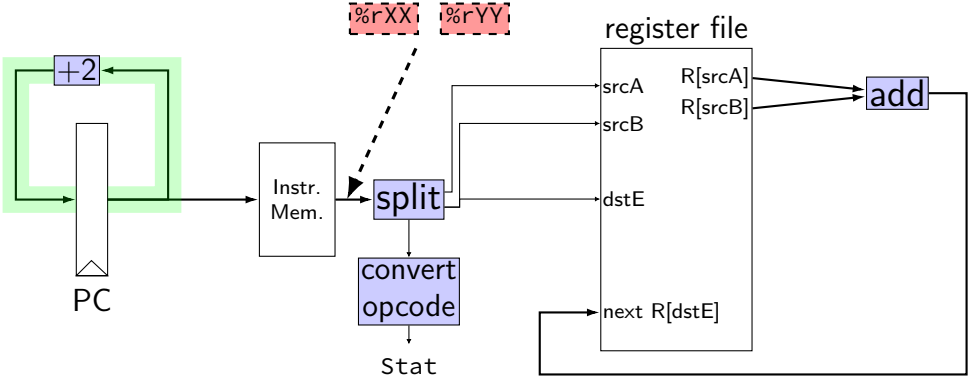



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL

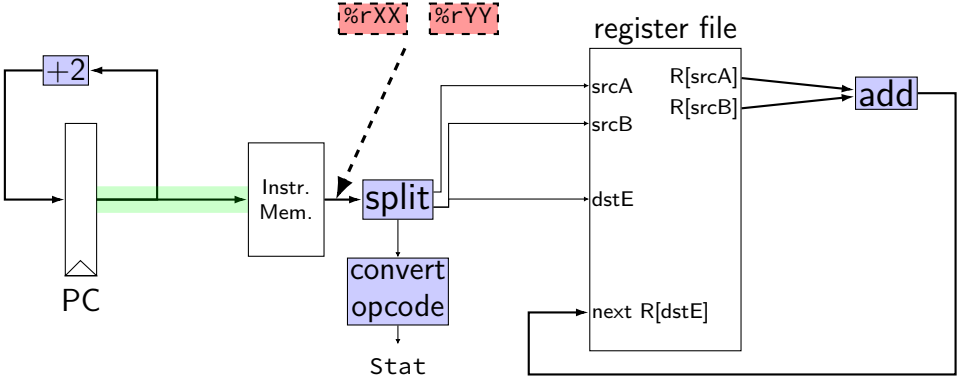



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL

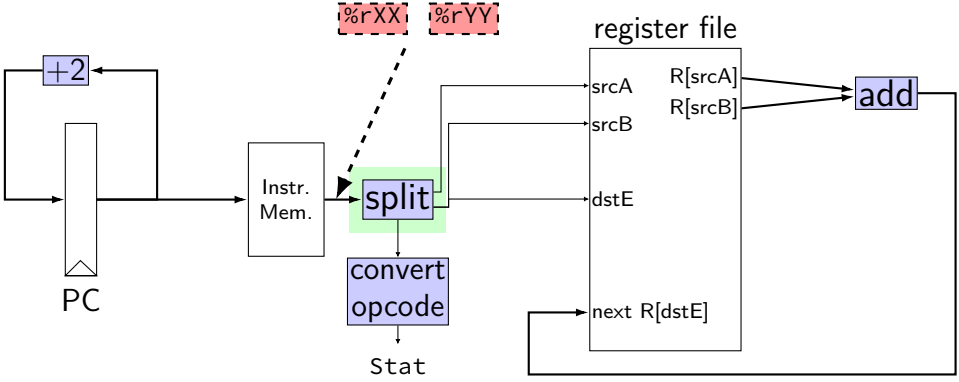



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL

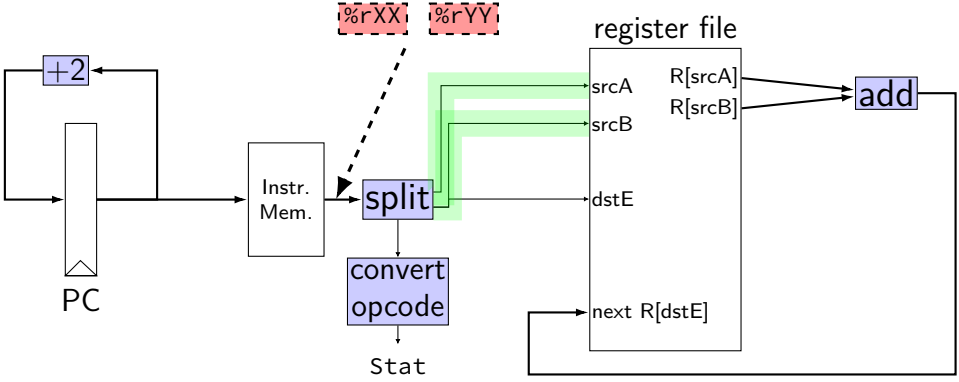



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL

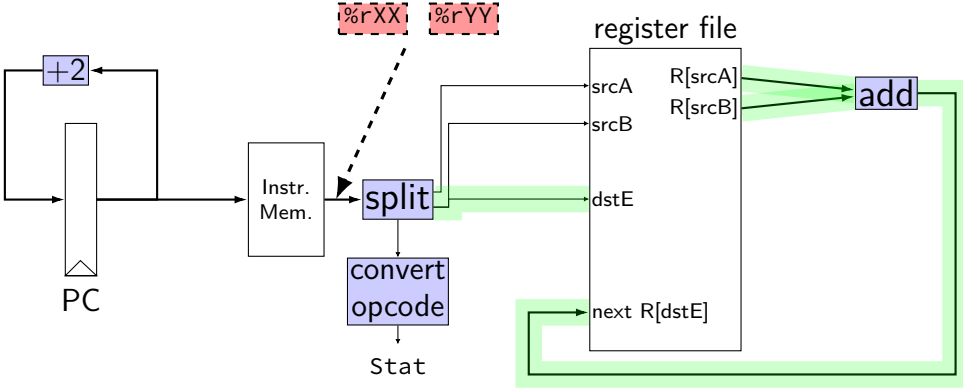



```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

addq CPU: HCL




```
register pP {
     pc : 64 = 0;
}
p_pc = P_pc + 2;
pc = P_pc;
```

```
wire opcode : 4;
wire rA : 4, rB : 4;
opcode = i10bytes[4..8];
rA = i10bytes[12..16];
rB = i10bytes[8..12];
```

```
reg_srcA = rA;
reg_srcB = rB;
reg_dstE = rB;
reg_inputE =
      reg_outputA +
      reg_outputB;
```

differences from book

wire not **b**ool or **i**nt

book uses names like `valC` — not required!

- author's environment limited adding new wires

MUXes must have default (`1 : something`) case

implement your own ALU

differences from book

wire not **b**ool or **i**nt

book uses names like `valC` — not required!

author's environment limited adding new wires

MUXes must have default (`1 : something`) case

implement your own ALU

differences from book

wire not **b**ool or **i**nt

book uses names like `valC` — not required!

author's environment limited adding new wires

MUXes must have default (`1 : something`) case

implement your own ALU