SEQ part 1

changelog

19 September 2022: “what is i10bytes?”: update table of i10bytes values to be consistent with hilited parts of memory values (fixing off-by-one error where one additional byte was included)

last time

Y86-64 instruction set
- small number of instructions
- name → operands
- mov instructions for accessing memory
- constant non-addresses only on irmovq

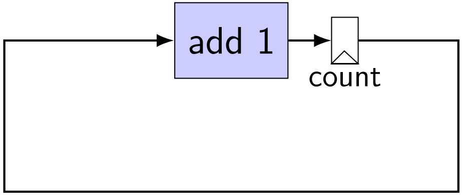
Y86-64 encoding/decoding
- *opcode* in consistent position
- some instructions have secondary opcode
- registers identified by index
- constants as 8-byte little endian numbers

hardware description languages

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

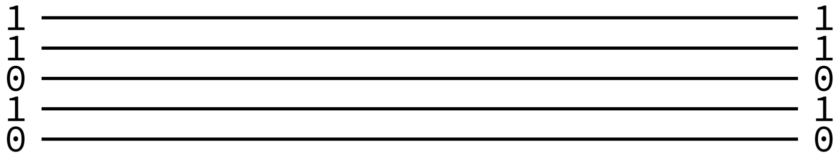
our tool: HCLRS

built for this course

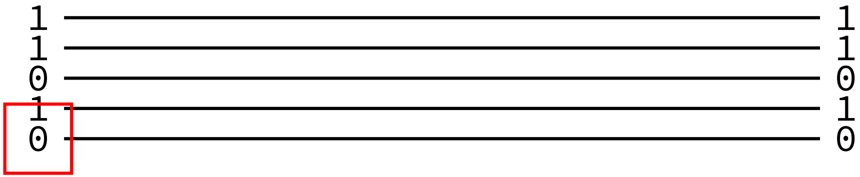
assumes you're making a processor

somewhat different from textbook's HCL

circuits: wires

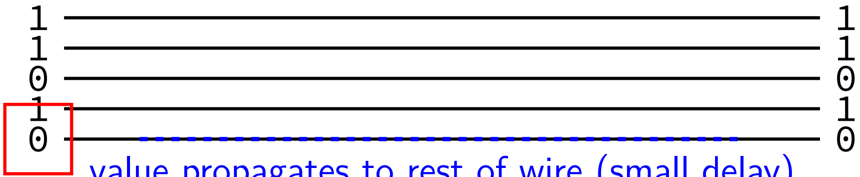


circuits: wires



binary value — actually voltage

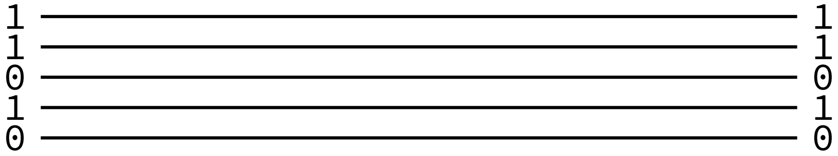
circuits: wires



value propagates to rest of wire (small delay)

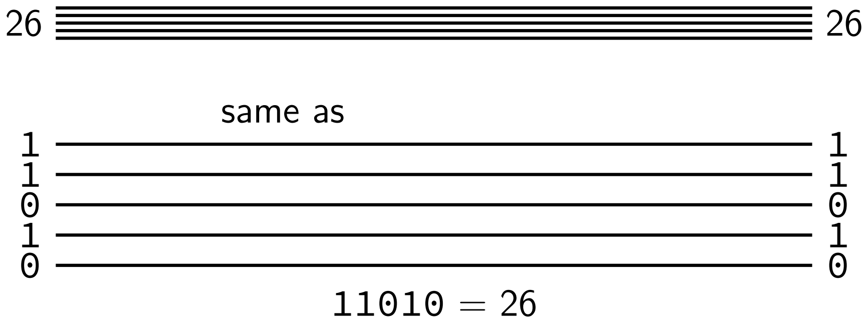
binary value — actually voltage

circuits: wire bundles

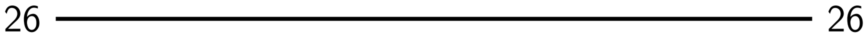


`11010` $= 26$

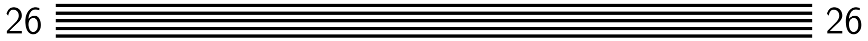
circuits: wire bundles



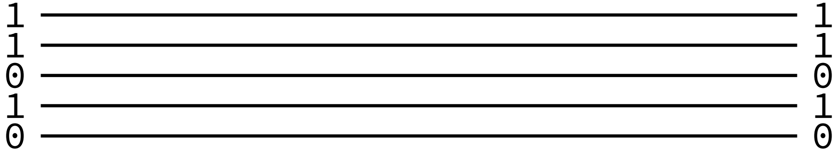
circuits: wire bundles



same as

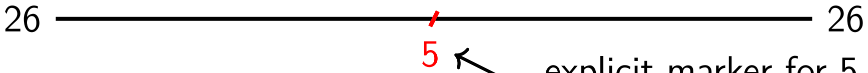


same as



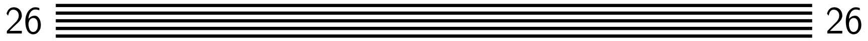
11010 $=$ 26

circuits: wire bundles

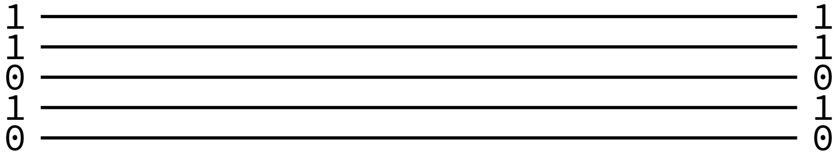


explicit marker for 5-bit wire bundle
often omitted to avoid clutter

same as

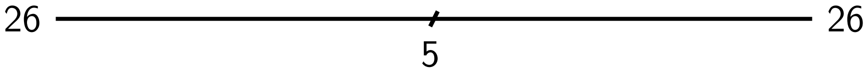


same as

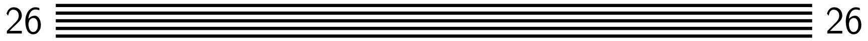


`11010` = 26

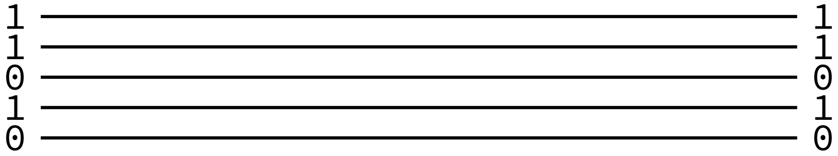
circuits: wire bundles



same as

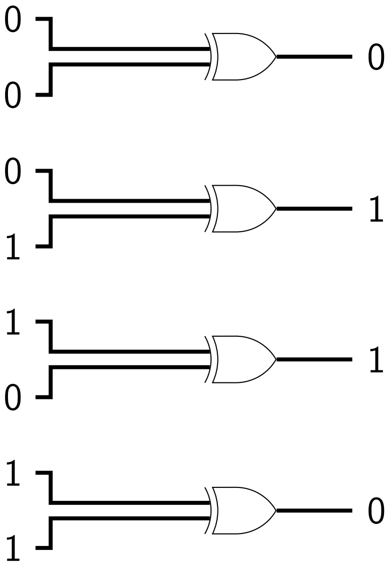


same as



`11010` = 26

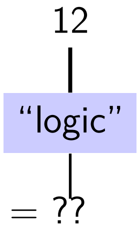
circuits: gates



circuits: logic

want to do calculations?

generalize gates:



function(12) = ??

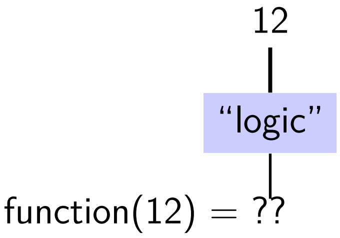
circuits: logic

want to do calculations?

generalize gates:

output wires contain result of function on input

changes as input changes (with delay)



circuits: logic

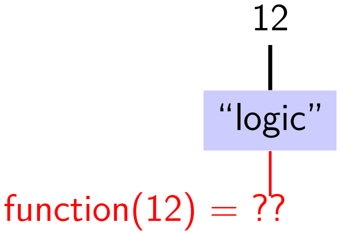
want to do calculations?

generalize gates:

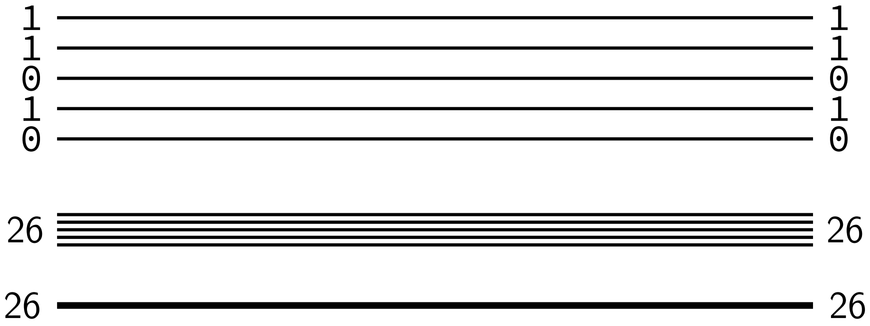
output wires contain result of function on input

changes as input changes (with delay)

need not be same width as output

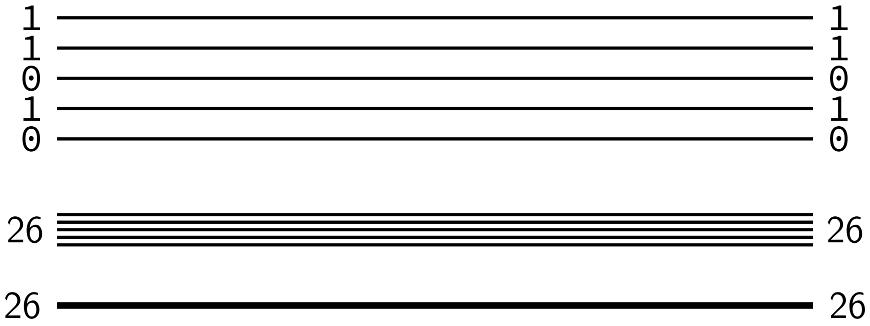


HCLRS: wire (bundle)s



```
wire foo : 5; foo = 0b11010;
```
OR
```
wire foo : 5; foo = 26;
```
OR
```
wire foo : 5; foo = 0x1a;
```

HCLRS: wire (bundle)s

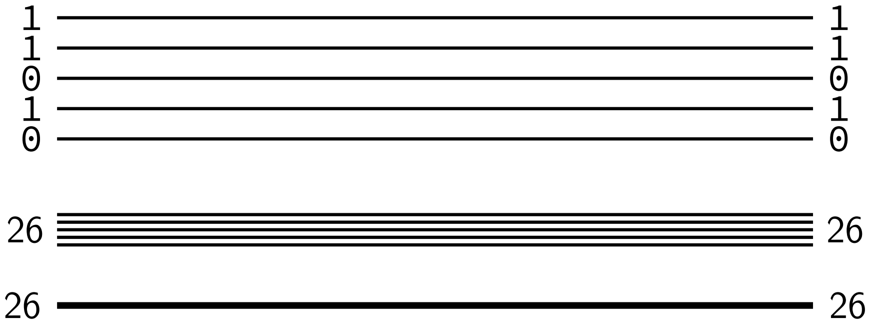


```
wire foo : 5; foo = 0b11010;
```
OR

```
wire foo : 5; foo = 26;
```
OR

```
wire foo : 5; foo = 0x1a;
```
name

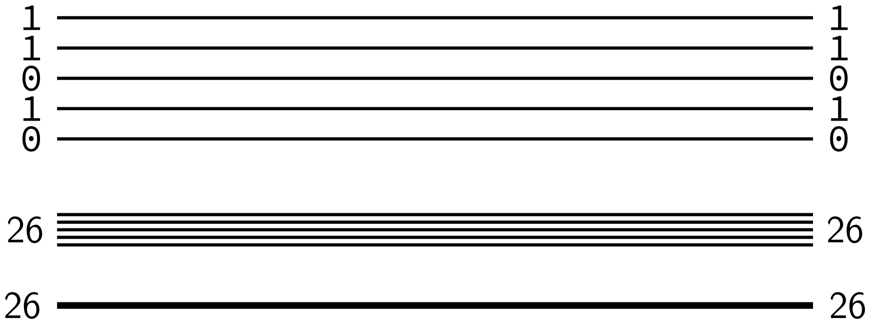
HCLRS: wire (bundle)s



```
wire foo : 5; foo = 0b11010;     OR
wire foo : 5; foo = 26;          OR
wire foo : 5; foo = 0x1a;
```

width (in bits)

HCLRS: wire (bundle)s



```
wire foo : 5; foo = 0b11010;
```

OR

```
wire foo : 5; foo = 26;
```

OR

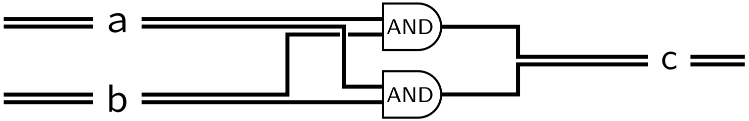
```
wire foo : 5; foo = 0x1a;
```

assignment

indicates wire is *connected* to value

HCLRS: gates + calcuations (1)

```
wire a : 2; wire b : 2; wire c : 2;
c = b & a;
a = 0b10;
b = 0b11;
```



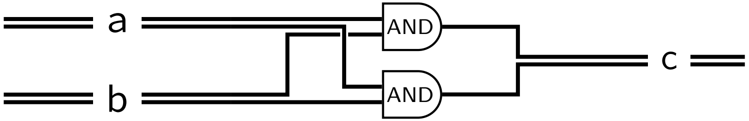
HCLRS: gates + calcuations (1)

```
wire a : 2; wire b : 2; wire c : 2;
```

$$\left.\begin{array}{l}\texttt{c = b \& a;}\\ \texttt{a = 0b10;}\\ \texttt{b = 0b11;}\end{array}\right\}\ \text{same as}\ \left\{\begin{array}{l}\texttt{a = 0b10;}\\ \texttt{b = 0b11;}\\ \texttt{c = b \& a;}\end{array}\right.$$

order doesn't matter

connected or not

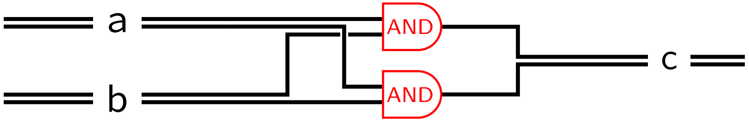


HCLRS: gates + calcuations (1)

```
wire a : 2; wire b : 2; wire c : 2;
c = b & a;
a = 0b10;
b = 0b11;
```

C-like expressions supported

`0b10 & 0b11 = 0b10`

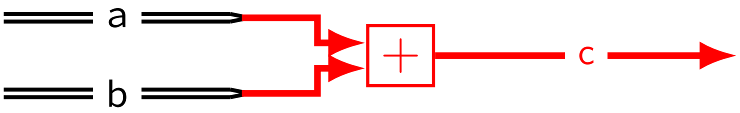


HCLRS: gates + calcuations (2)

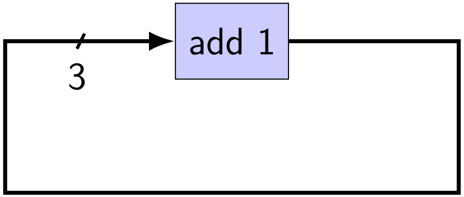
```
wire a : 2; wire b : 2; wire c : 2;
c = b + a; /* was ~~b & a~~ */
a = 0b10;
b = 0b11;
```

more than bitwise operators supported

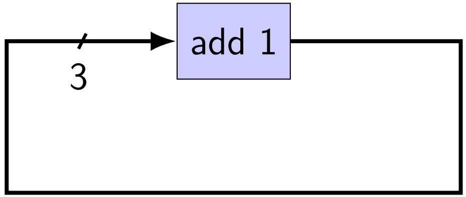
0b10 + 0b11 = 0b101 → 0b01 (extra bits lost)



example: (broken) counter circuit (1)

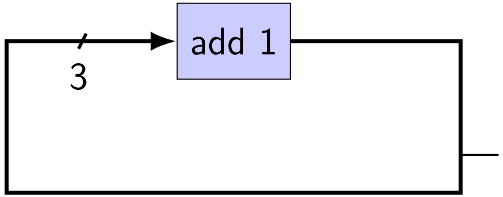


example: (broken) counter circuit (1)



```
wire x : 3;
x = x + 1;
```

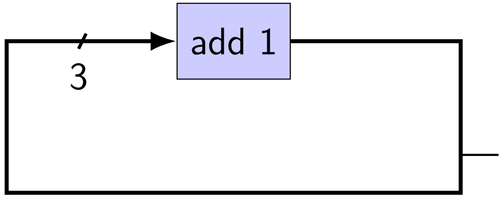
example: (broken) counter circuit (1)



time 0: `000`

```
wire x : 3;
x = x + 1;
```

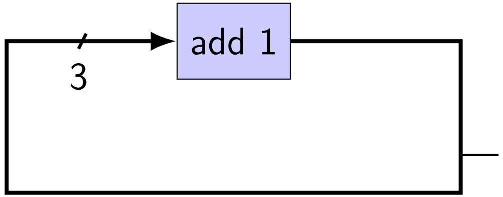
example: (broken) counter circuit (1)



time 0: `000` ← set how???

```
wire x : 3;
x = x + 1;
```

example: (broken) counter circuit (1)



time 0: `000`

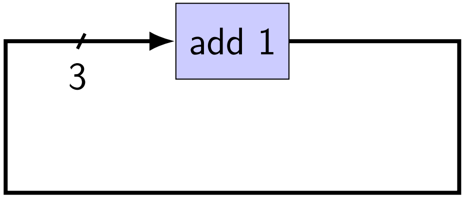
time 1: `001`?

time 2: `010`?

time 3: `011`?

```
wire x : 3;
x = x + 1;
```

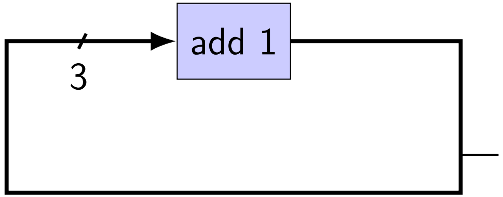
example: (broken) counter circuit (2)



~~```
wire x : 3;
x = x + 1;
```~~

HCLRS: compile error
“`Circular dependency detected:`
`x depends on x`”
```

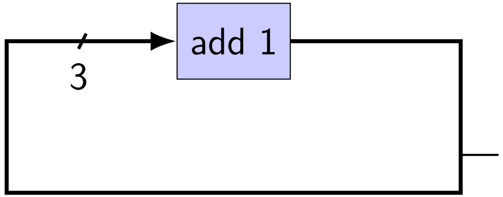
example: (broken) counter circuit (3)



time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

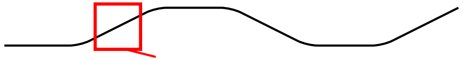
```
wire x : 3;
x = x + 1;
```

example: (broken) counter circuit (3)



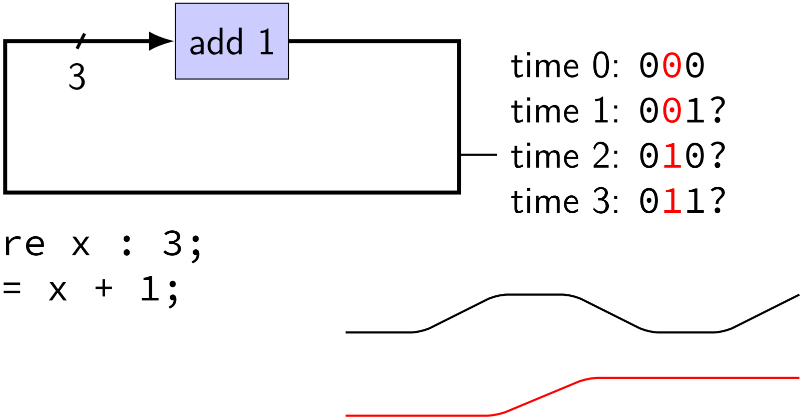
time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

```
wire x : 3;
x = x + 1;
```



problem 1: how will "add 1" react to this value?
(not zero or one) …

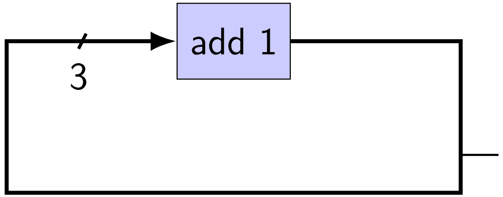
example: (broken) counter circuit (3)



time 0: 000
time 1: 001?
time 2: 010?
time 3: 011?

```
wire x : 3;
x = x + 1;
```

example: (broken) counter circuit (3)



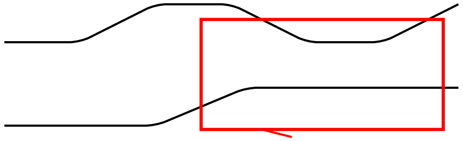
time 0: `000`

time 1: `001?`

time 2: `010?`

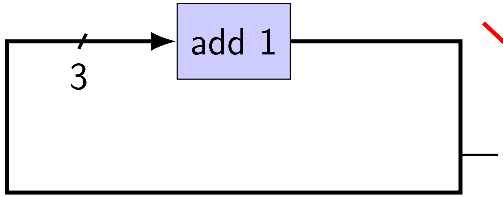
time 3: `011?`

```
wire x : 3;
x = x + 1;
```



problem 2: changes not in sync?

example: (broken) counter circuit (4)



~~time 0: 000~~
~~time 1: 001?~~
~~time 2: 010?~~
~~time 3: 011?~~

```
wire x : 3;
x = x + 1;
```

circuit is not stable
transient values during changes
hard to predict behavior

circuits: state

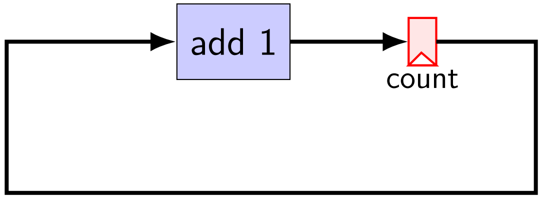
logic performs calculations all the time

never stores values!

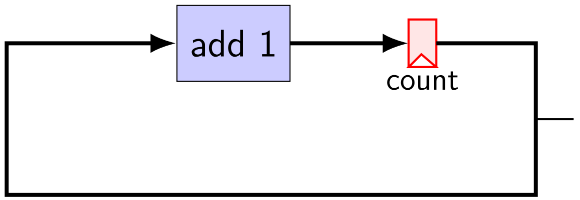
need extra elements to store values

- registers, memory

example: counter circuit (corrected)



example: counter circuit (corrected)



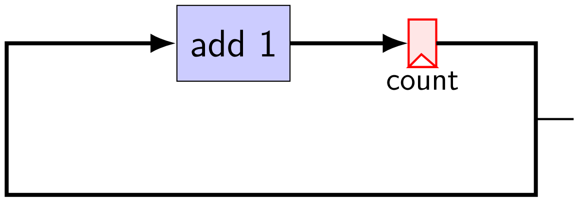
time 0: `000`

time 1: `001`

time 2: `010`

time 3: `011`

example: counter circuit (corrected)



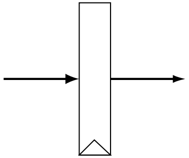
time 0: `000`
time 1: `001`
time 2: `010`
time 3: `011`

add register to store current count

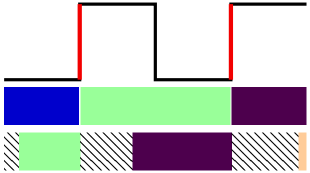
updates based on "clock signal" (not shown)

avoids intermediate updates

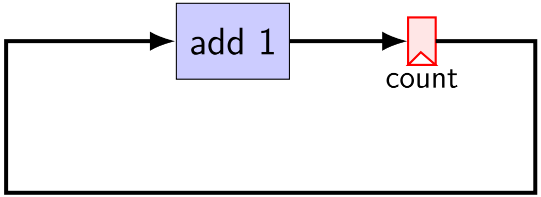
registers



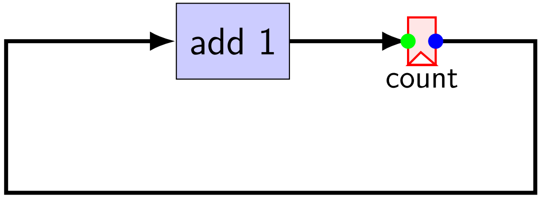
updates every clock cycle



example: counter circuit (real HCLRS)

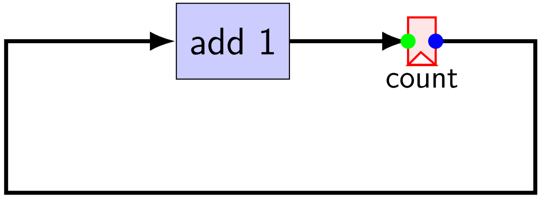


example: counter circuit (real HCLRS)



```
register xY {
      count : 3 = 0b000 ;
}
 x_count = Y_count + 0b001;
```

example: counter circuit (real HCLRS)

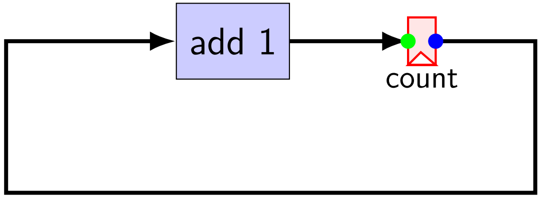


```
register xY {
     count : 3 = 0b000 ;
}
 x_count = Y_count + 0b001;
```

register “bank”

can have multiple (related) register

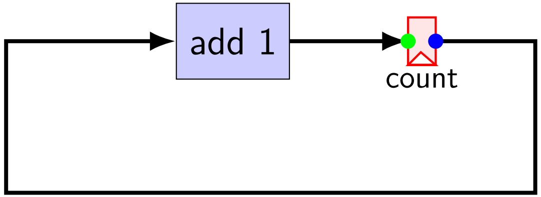
example: counter circuit (real HCLRS)



label for left/right side of registers
x: label for input side (always lowercase)
Y: label for output side (always uppercase)

```
register xY {
    count : 3 = 0b000 ;
}
x_count = Y_count + 0b001;
```

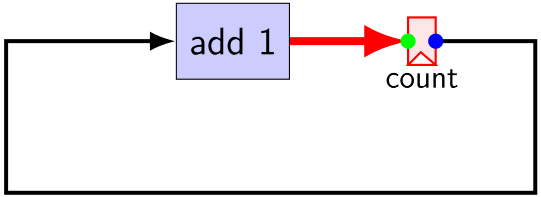
example: counter circuit (real HCLRS)



register "name"
input/output = *prefix*_name

```
register  xY  {
     count  : 3 =  0b000 ;
}
 x_count  =  Y_count  + 0b001;
```

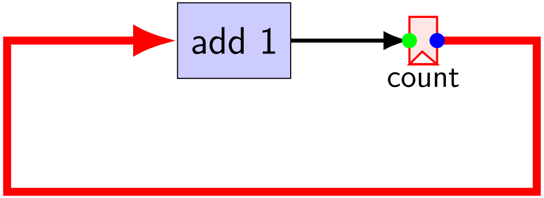
example: counter circuit (real HCLRS)



```
register xY {
     count : 3 = 0b000 ;
}
x_count = Y_count + 0b001;
```

input wire to register

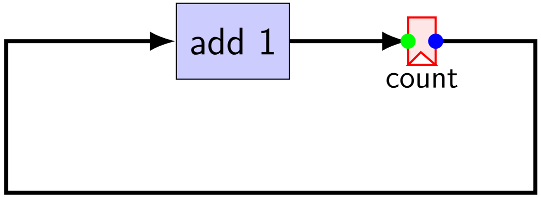
example: counter circuit (real HCLRS)



```
register xY {
     count : 3 = 0b000 ;
}
 x_count = Y_count + 0b001;
```

output wire of register

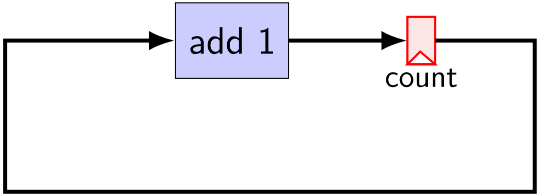
example: counter circuit (real HCLRS)



initial value of register
first value for output wire (Y_count)

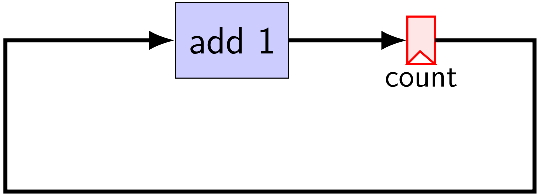
```
register xY {
      count : 3 = 0b000;
}
 x_count = Y_count + 0b001;
```

example: counter circuit



```
register  xY  {
      count  : 3 =  0b000 ;
}
 x_count  =  Y_count  + 0b001;
```

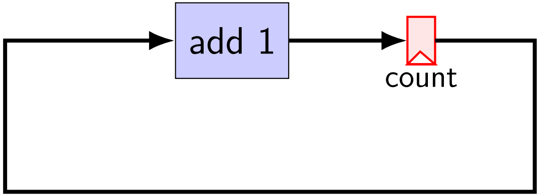
example: counter circuit



```
register xY {
        count : 3 = 0b000 ;
}
 x_count = Y_count + 0b001;
```

time	Y_count	x_count
start	000	001
start + 1 rising edge	001	010
start + 2 rising edges	010	011
start + 3 rising edges	011	100
…	…	…

example: counter circuit



```
register  xY  {
        count  : 3 =  0b000 ;
}
 x_count  =  Y_count  + 0b001;
```

time	Y_count	x_count
start	000	001
start + 1 rising edge	001	010
start + 2 rising edges	010	011
start + 3 rising edges	011	100
...	...	...

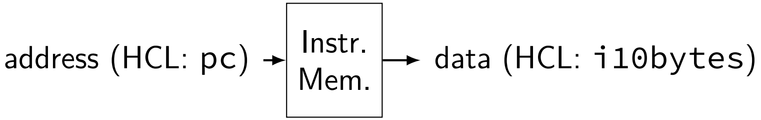
HCL circuit with registers

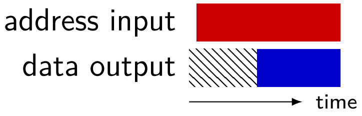
```
register xY {
    a : 4 = 1;  /* <-- initial Y_a */
    b : 4 = 1;  /* <-- initial Y_b */
}
x_b = x_a + Y_a;
x_a = Y_a + Y_b;
```

exercise: value of Y_a, Y_b after two rising edges of clock?

A. Y_a = 2, Y_b = 3
B. Y_a = 2, Y_b = 2
C. Y_a = 3, Y_b = 5
D. Y_a = 3, Y_b = 7
E. Y_a = 3, Y_b = 11
F. Y_a = 5, Y_b = 7
G. Y_a = 7, Y_b = 11
H. none of the above

instruction memory





Stat signal

how do we stop the simulated machine?

hard-wired mechanism — `Stat` wire

possible values:

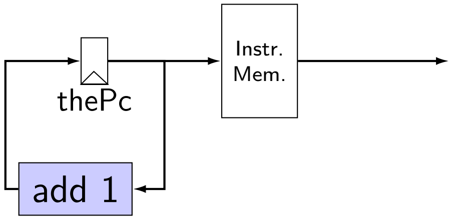
- `STAT_AOK` — keep going
- `STAT_HLT` — stop, normal shtdown
- `STAT_INS` — invalid instruction
- …(and more errors)

(predefined 3-bit constants)

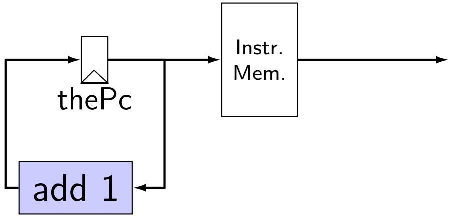
must be set

determines if simulator keeps going

nop CPU

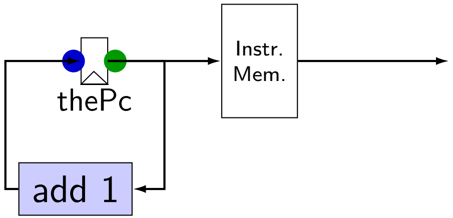


nop CPU



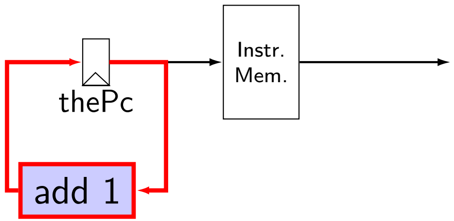
```
register pF {
    thePc : 64 = 0;
}
```

nop CPU



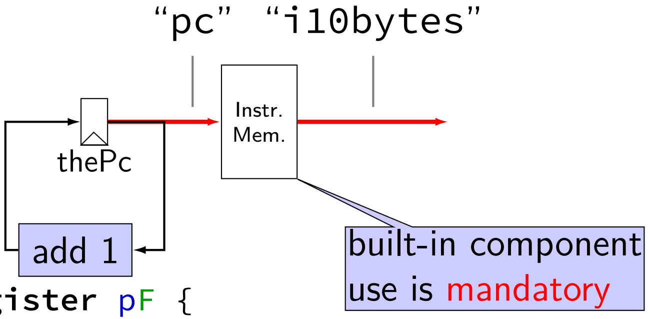
```
register pF {
    thePc : 64 = 0;
}
```

nop CPU



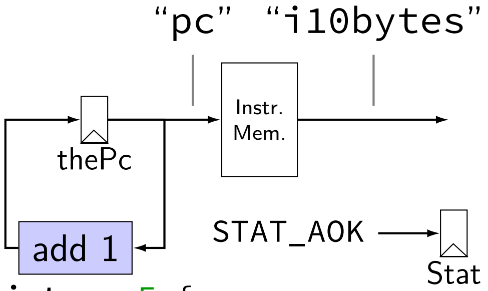
```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
```

nop CPU



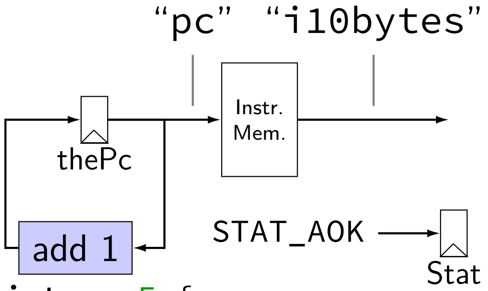
```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```

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]
....
+------------- 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                                        |
+-----------------------------------------------------------------------+
```

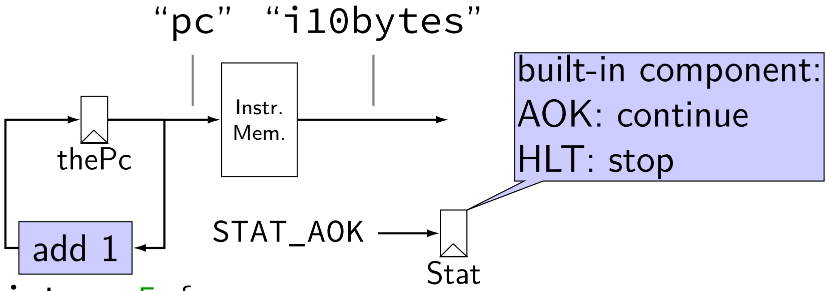
running a simulator (2)

```
$ ./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)

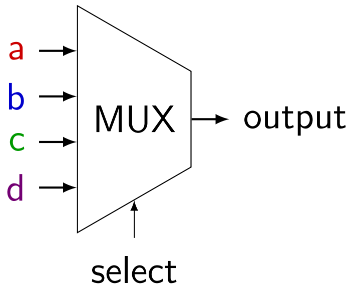
```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:                0   RCX:                0   RDX:                0 |
| RBX:                0   RSP:                0   RBP:                0 |
| RSI:                0   RDI:                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

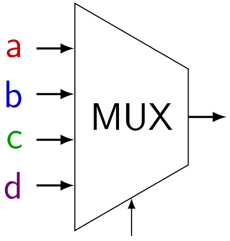


```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
Stat = STAT_AOK;
```

multiplexers



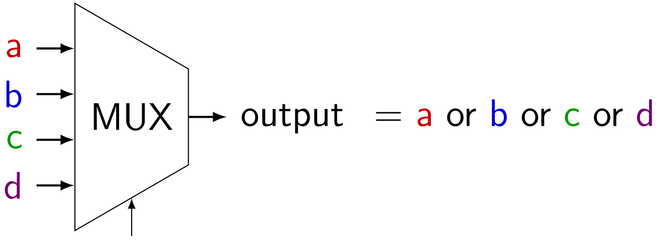
multiplexers



output = a or b or c or d

select = 0 or 1 or 2 or 3

multiplexers



select = 0 or 1 or 2 or 3

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

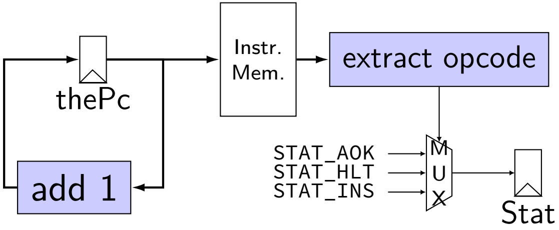
    encoding `10`

halt

    encoding `00`

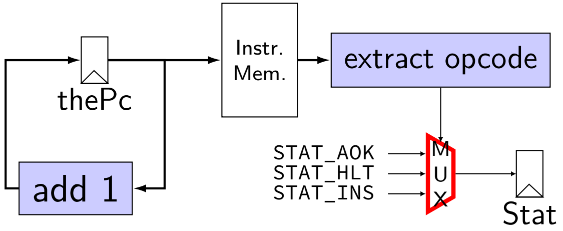
our strategy: MUX to decide using opcode

nop/halt CPU



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

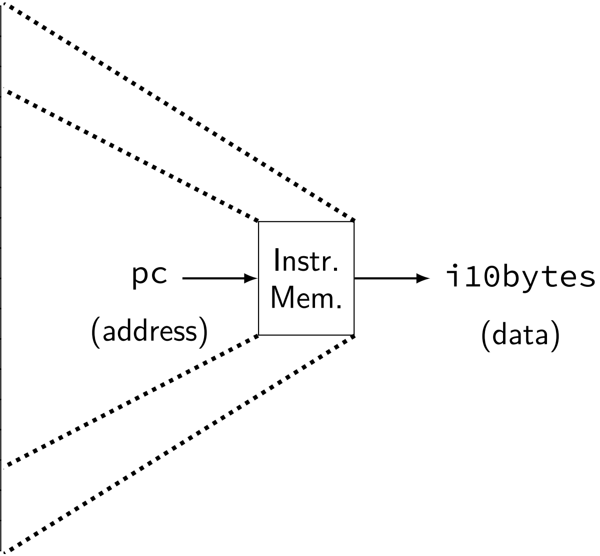
nop/halt CPU



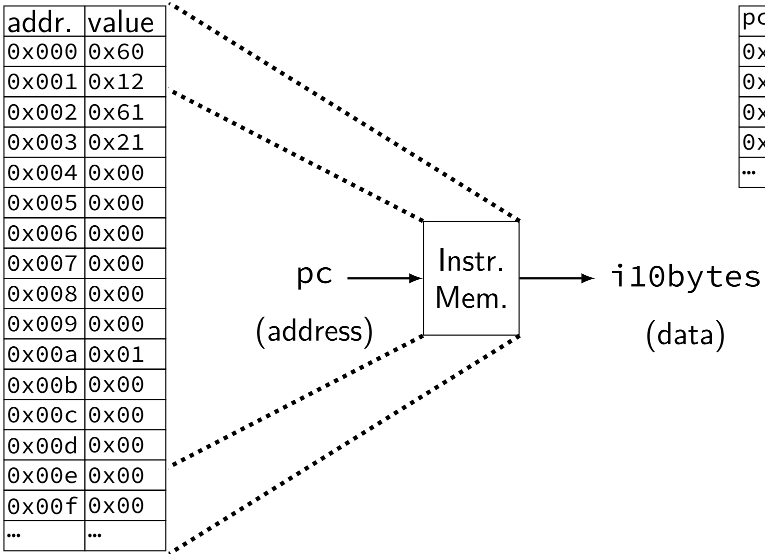
Instr. Mem.
extract opcode
thePc
add 1
STAT_AOK
STAT_HLT
STAT_INS
MUX
Stat

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?

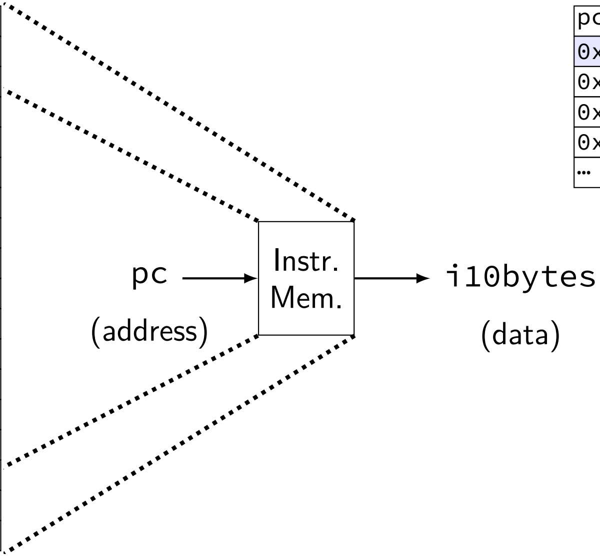


addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…

pc	i10bytes
0x000	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
…	…

what is i10bytes?

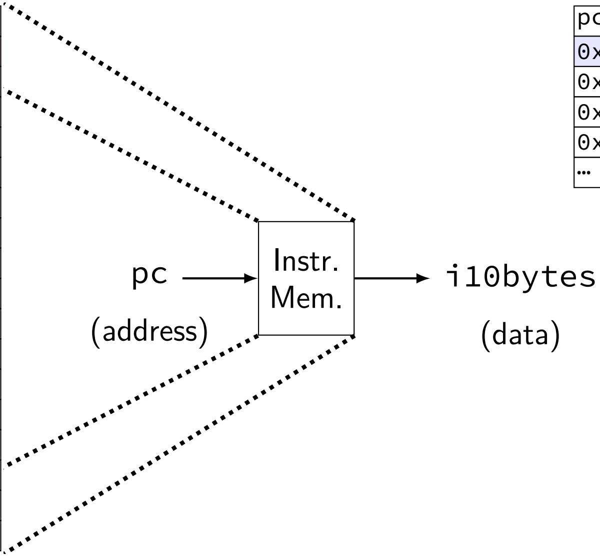
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x000000000021611260
0x001	0x010000000000216112
0x002	0x000100000000002161
0x003	0x000001000000000021
…	…

what is i10bytes?

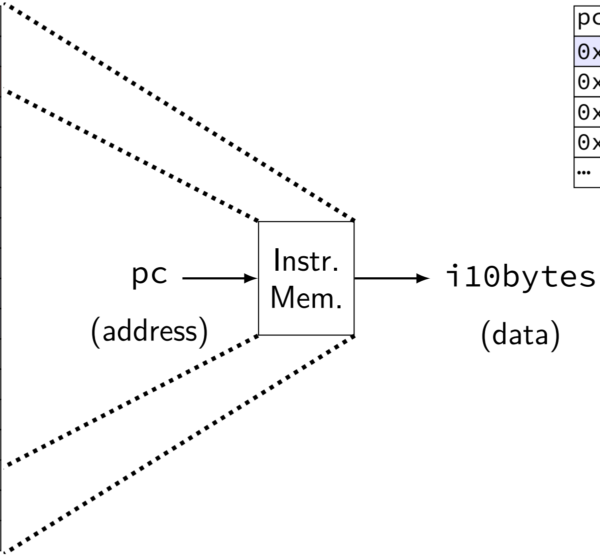
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
...	...



pc	i10bytes
0x000	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
...	...

what is i10bytes?

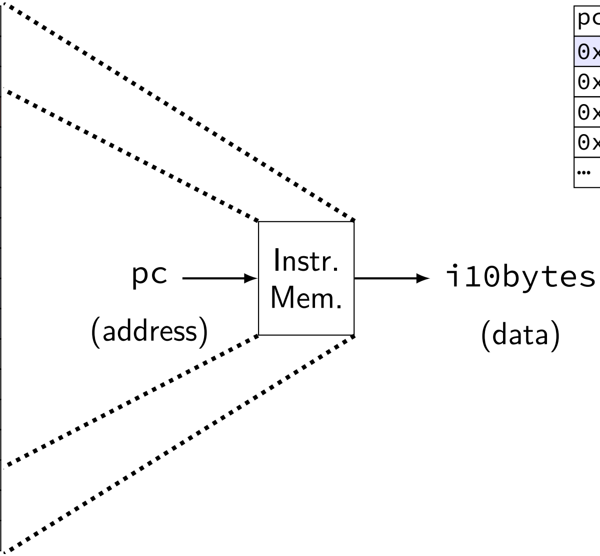
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
...	...



pc	i10bytes
0x000	0x00000000000216112 60
0x001	0x010000000000216112
0x002	0x000100000000002161
0x003	0x000001000000000021
...	...

what is i10bytes?

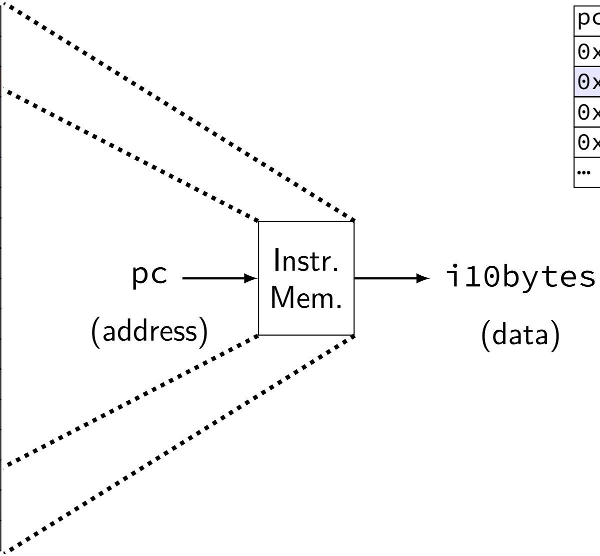
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
…	…

what is i10bytes?

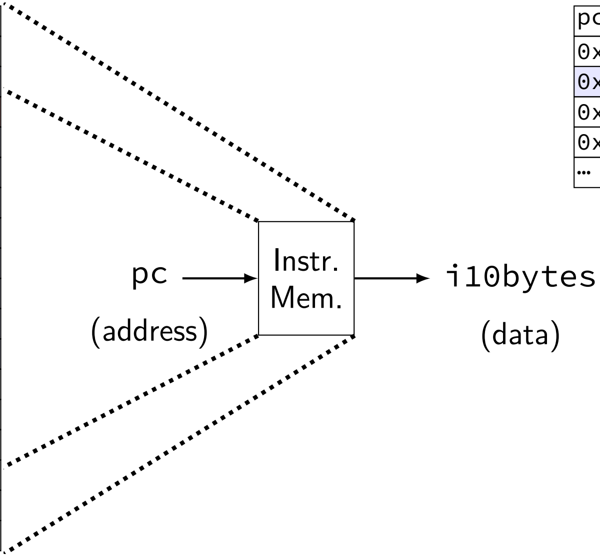
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
...	...



pc	i10bytes
0x000	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
...	...

what is i10bytes?

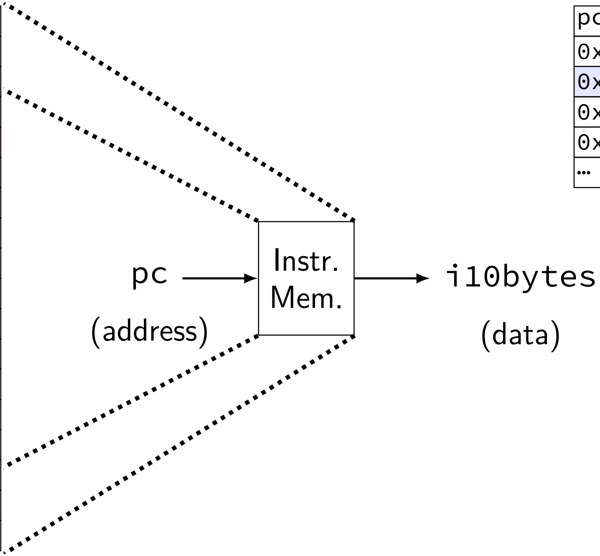
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
...	...



pc	i10bytes
0x000	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
...	...

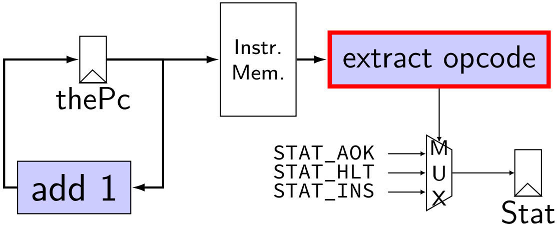
what is i10bytes?

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	0x00000000000021611260
0x001	0x01000000000000216112
0x002	0x00010000000000002161
0x003	0x00000100000000000021
…	…

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

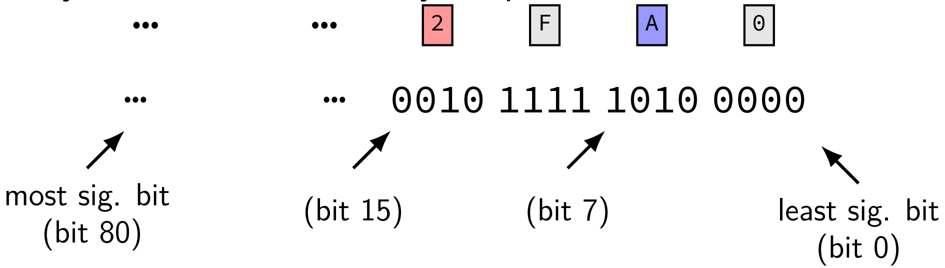
i10bytes example

`pushq %rbx` at memory address x: `A` `0` `2` `F`

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

$x + 0$: `A` `F`; at $x + 1$: `2` `F`; at $x + 2$: (next instruction)

10-byte instruction memory output:



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								

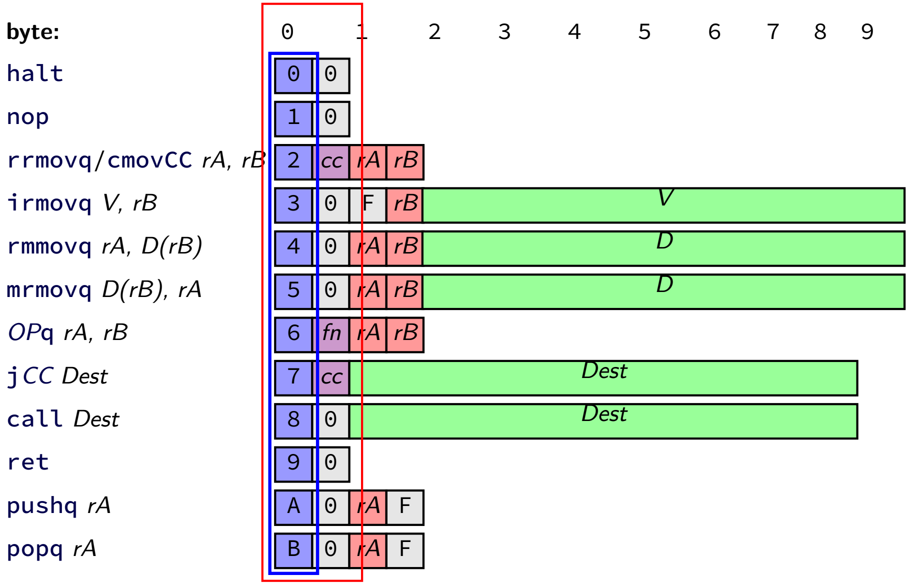
byte 0: bits 0–7

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

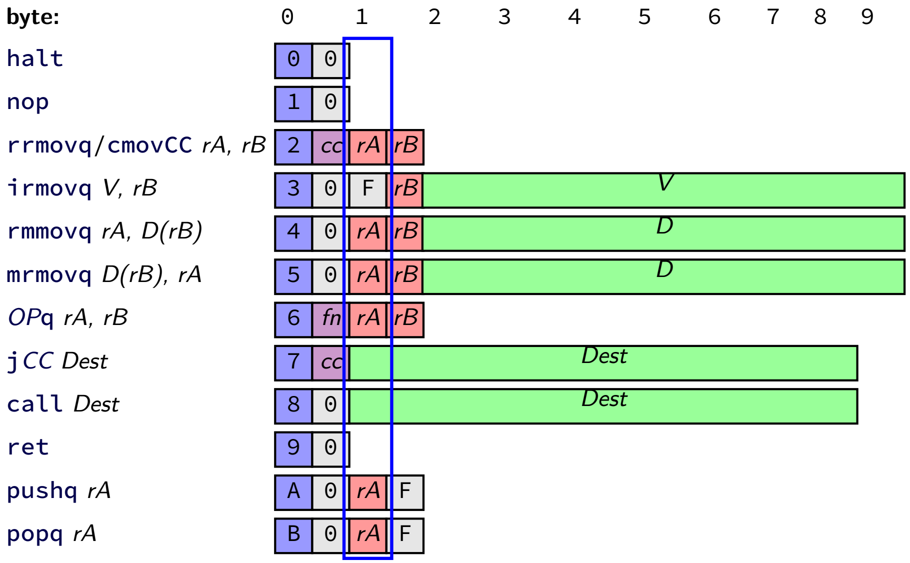
Y86 encoding table



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



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								

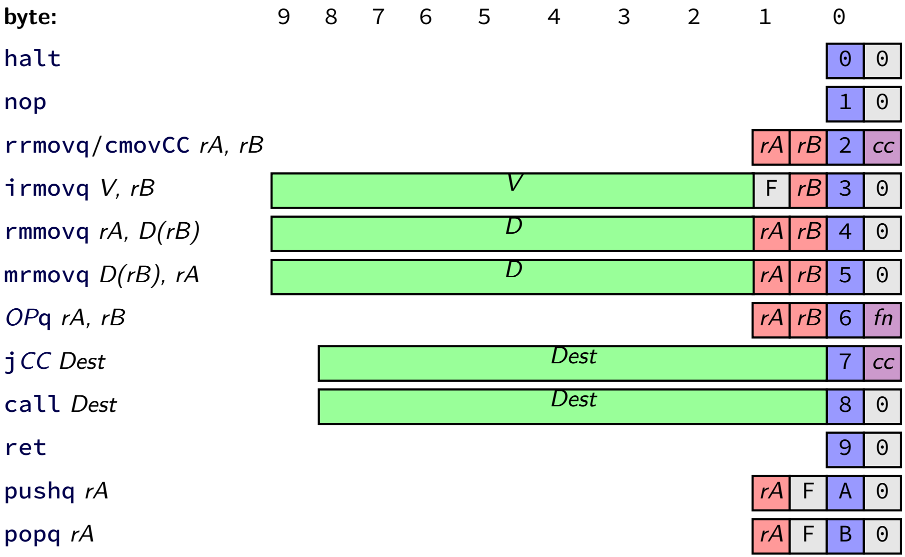
most sig. 4 bits of byte 1: bits 12–16

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								

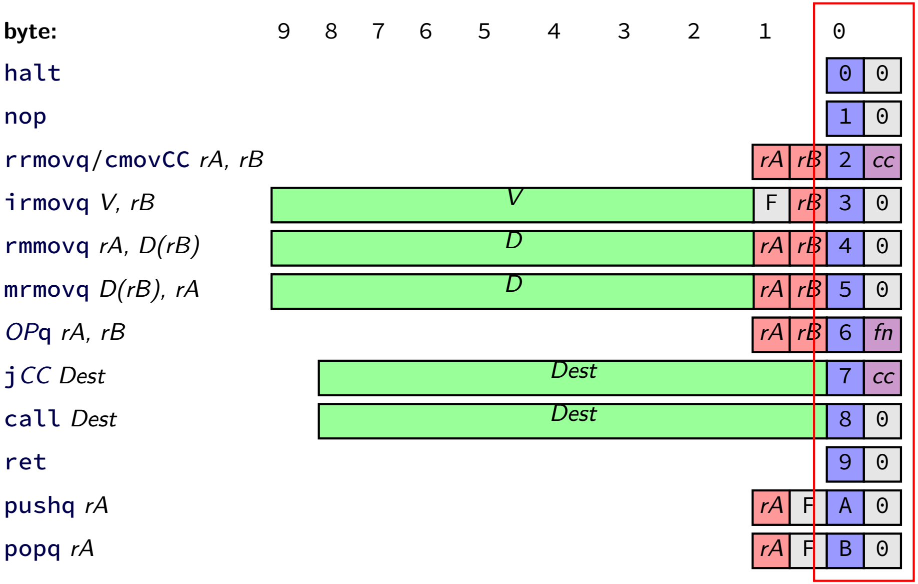
least sig. 4 bits of byte 1: bits 8–12

Y86 encoding table (written differently)



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

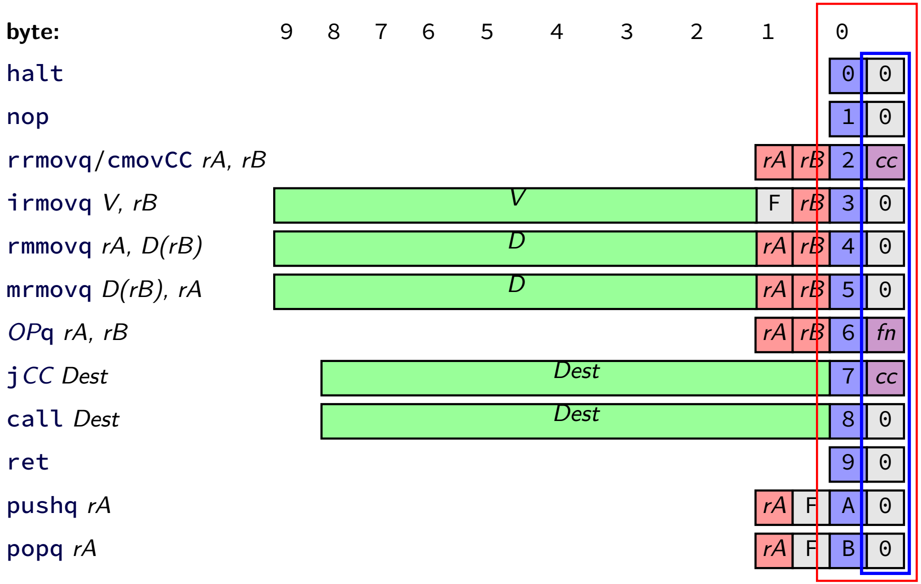
Y86 encoding table (written differently)



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

byte 0: bits 0–7

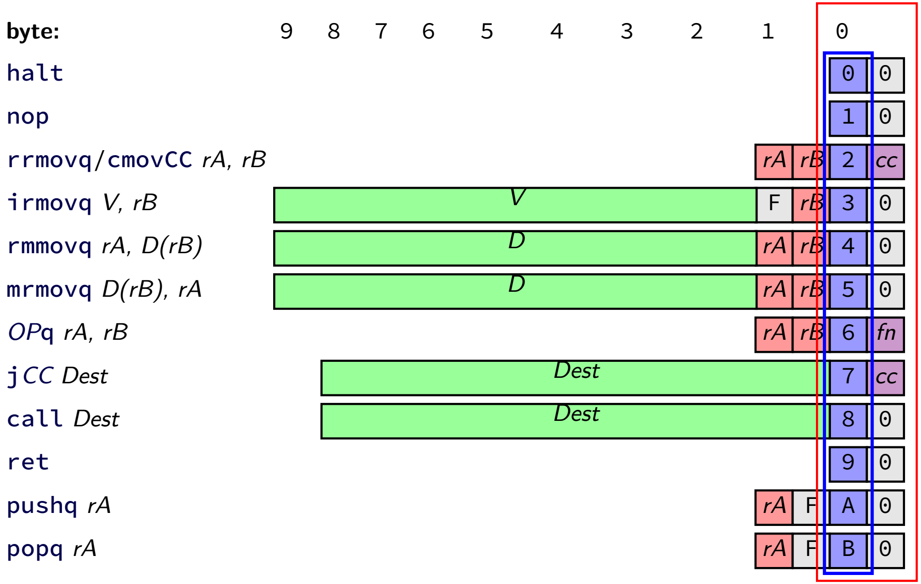
Y86 encoding table (written differently)



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

least sig. 4 bits of byte 0: bits 0–4

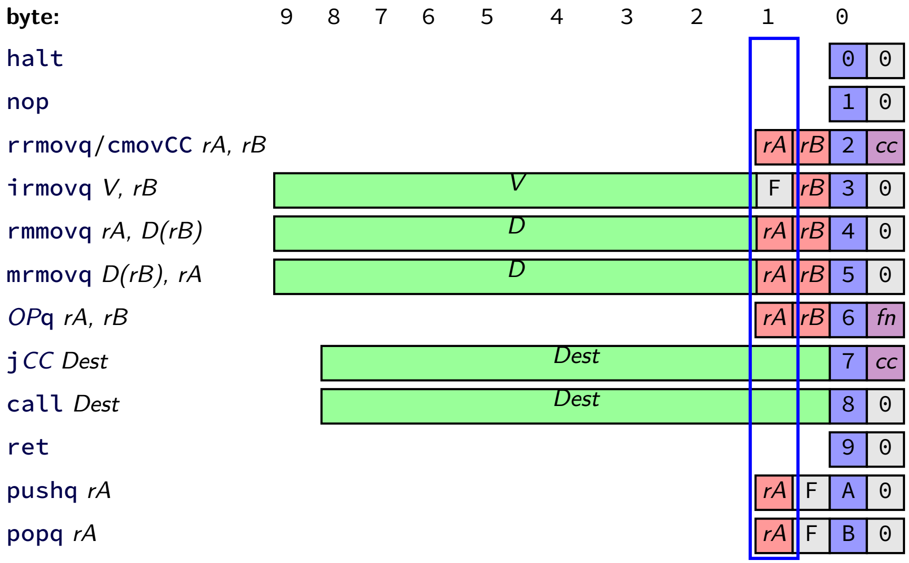
Y86 encoding table (written differently)



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

most sig. 4 bits of byte 0: bits 4–8

Y86 encoding table (written differently)



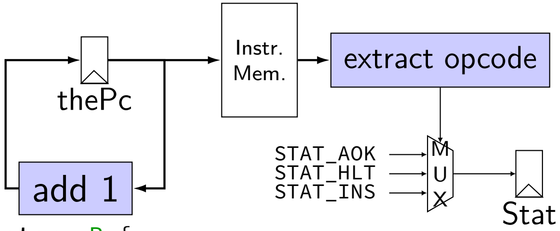
most sig. 4 bits of byte 1: bits 12–16

Y86 encoding table (written differently)

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

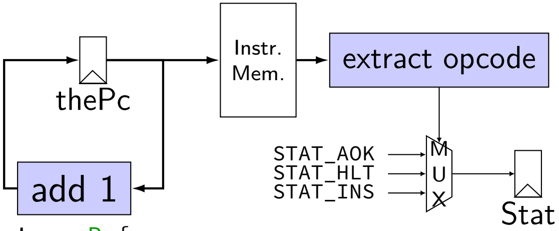
least sig. 4 bits of byte 1: bits 8–12

nop/halt CPU



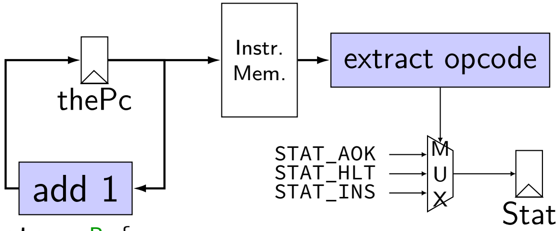
```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

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)
];
```

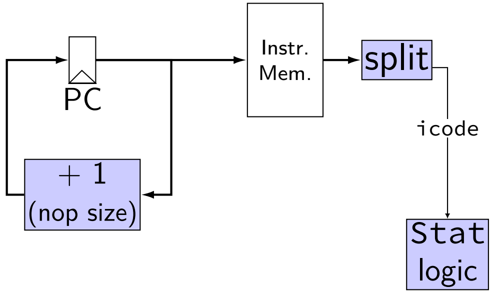
nop/halt CPU



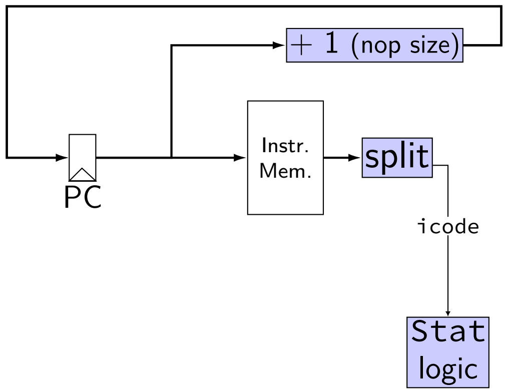
```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

demo

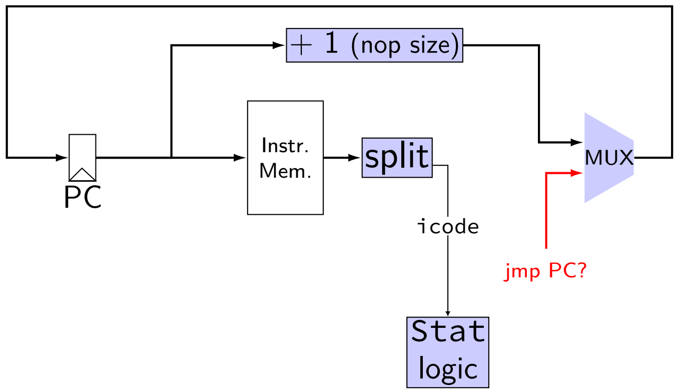
nop/halt → nop/jmp CPU



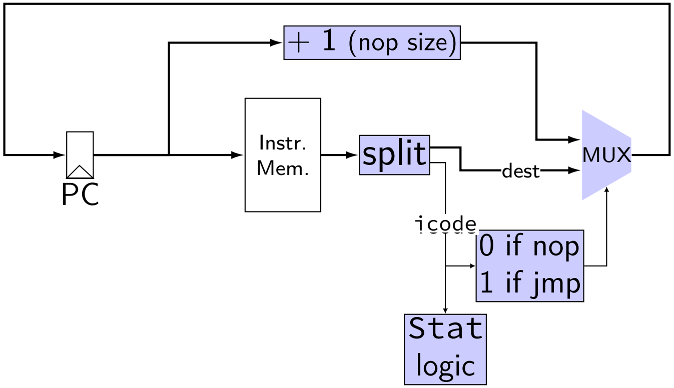
nop/halt → nop/jmp CPU



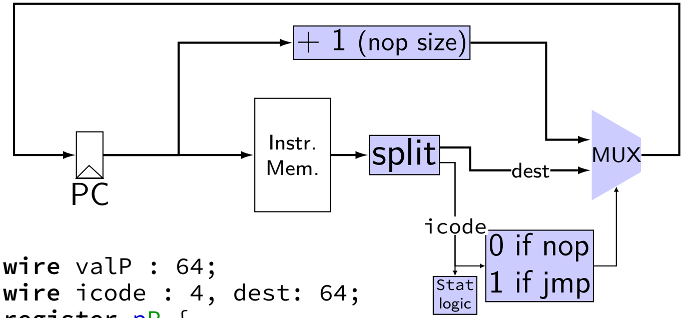
nop/halt → nop/jmp CPU



nop/halt → nop/jmp CPU



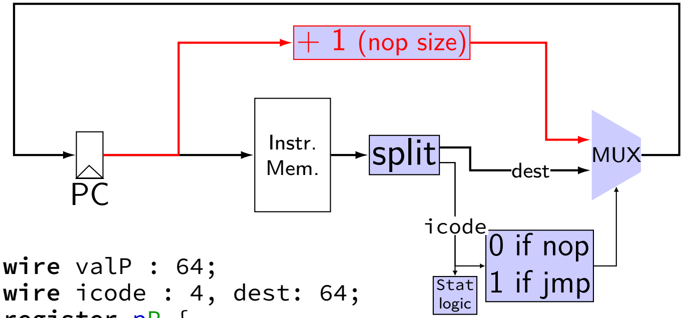
nop/jmp CPU



```
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;
];
```

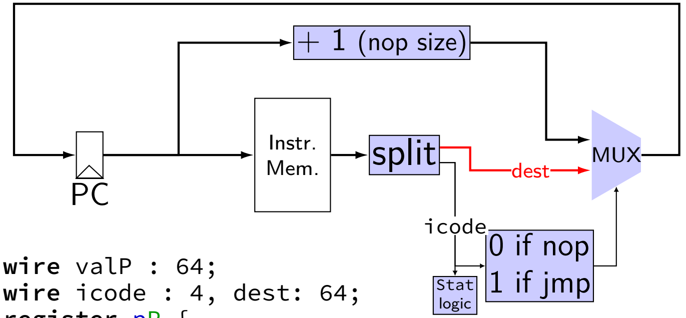
nop/jmp CPU



```
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;
];
```

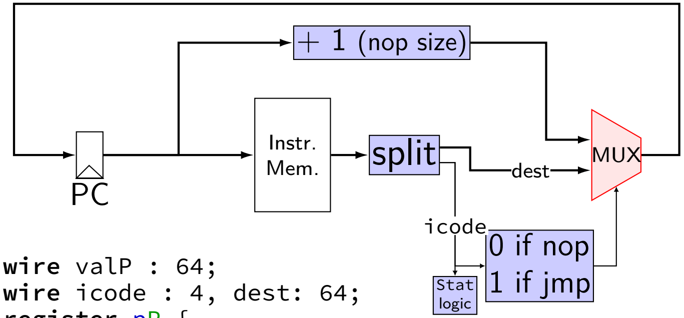
nop/jmp CPU



```
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;
];
```

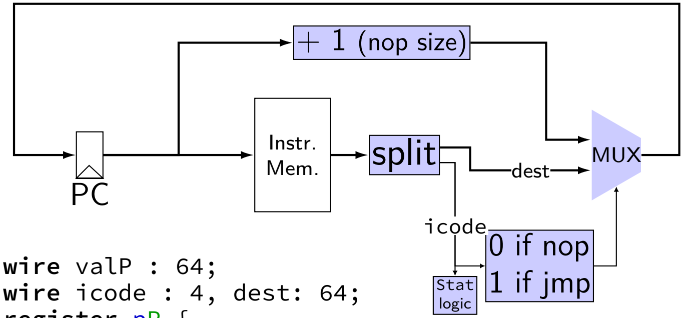
nop/jmp CPU



```
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: 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: 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: 701300000000000000       |      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
```

demo: debug and interactive mode

debugging mode

```
+------------------- 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 pP(N)  thePc=0000000000000000                                 |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 70 13 00  00 00 00 00   00 00 70 1c  00 00 00 00    |
|  0x0000001_:   00 00 00 70  0a 00 00 00   00 00 00 00  10 10 00       |
+-----------------------------------------------------------------------+
i10bytes set to 0x137010 (reading 10 bytes from memory at pc=0x0)
pc = 0x0; loaded [10 : nop]
Values of inputs to built-in components:
pc                     0x0000000000000000
Stat                                  0x1

Values of outputs of built-in components:
i10bytes           0x00000000000000137010

Values of register bank signals:
P_thePc                0x0000000000000000
p_thePc                0x0000000000000001

Values of other wires:
dest                   0x0000000000001370
icode                                 0x1
valP                   0x0000000000000001
```

debugging mode

```
+------------------- 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 pP(N)  thePc=0000000000000000                                      |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 70 13 00  00 00 00 00   00 00 70 1c  00 00 00 00    |
|  0x0000001_:   00 00 00 70  0a 00 00 00   00 00 00 00  10 10 00       |
+-----------------------------------------------------------------------+
i10bytes set to 0x137010 (reading 10 bytes from memory at pc=0x0)
pc = 0x0; loaded [10 : nop]
Values of inputs to built-in components:
pc                    0x0000000000000000
Stat                                 0x1

Values of outputs of built-in components:
i10bytes          0x00000000000000137010

Values of register bank signals:
P_thePc               0x0000000000000000
p_thePc               0x0000000000000001

Values of other wires:
dest                  0x0000000000001370
icode                                0x1
valP                  0x0000000000000001
```

interactive + debugging mode

```
$ ./nopjmp_cpu.exe -i -d nopjmp.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 pP(N)  thePc=0000000000000000                              |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 70 13 00  00 00 00 00   00 00 70 1c  00 00 00 00    |
|  0x0000001_:   00 00 00 70  0a 00 00 00   00 00 00 00  10 10 00       |
+-----------------------------------------------------------------------+
(press enter to continue)
i10bytes set to 0x137010 (reading 10 bytes from memory at pc=0x0)
pc = 0x0; loaded [10 : nop]
Values of inputs to built-in components:
....
```

interactive + debugging mode

```
$ ./nopjmp_cpu.exe -i -d nopjmp.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 pP(N)  thePc=0000000000000000                                    |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 70 13 00  00 00 00 00   00 00 70 1c  00 00 00 00    |
|  0x0000001_:   00 00 00 70  0a 00 00 00   00 00 00 00  10 10 00       |
+-----------------------------------------------------------------------+
(press enter to continue)
i10bytes set to 0x137010 (reading 10 bytes from memory at pc=0x0)
pc = 0x0; loaded [10 : nop]
Values of inputs to built-in components:
....
```

quiet mode

```
$ ./hclrs nopjmp_cpu.hcl -q nopjmp.yo
+----------------------- halted 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 pP(N) { thePc=0000000000000000 }                             |
| used memory:   _0 _1 _2 _3  _4 _5 _6 _7   _8 _9 _a _b  _c _d _e _f    |
|  0x0000000_:   10 70 13 00  00 00 00 00   00 00 70 1c  00 00 00 00    |
|  0x0000001_:   00 00 00 70  0a 00 00 00   00 00 00 00  10 10 00       |
+--------------------- (end of halted state) ---------------------------+
Cycles run: 7
```