SEQ part 1

changelog

Changes since first lecture:

16 September 2021: example of pushq mapping to bit numbers: be explicit about next instruction placement; change slide title to mention i10bytes

16 September 2021: example of pushq mapping to bit numbers: encode pushq as A0 ... instead of AF ...

last time

Y86-64: simpler subset of x86-64

cmov*CC* instruction

Y86-64: single set of operands for every mnemonic

- split movq into separate instructions
- one way of specifying address
- arithmetic instructions only use registers
- jmp, call instruction only takes address constant

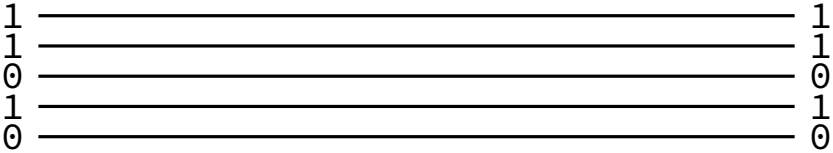
Y86-64 encoding:

- 4 most sig bits of first byte: which instruction (category)
- tables for which condition, which register
- rrmovq special case of cmovCC, jmp special case of jCC

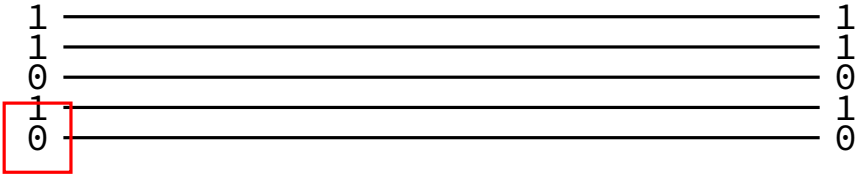
bundles of wires = 'N-bit wire'

HCLRS: wires as pseudo-variables

circuits: wires

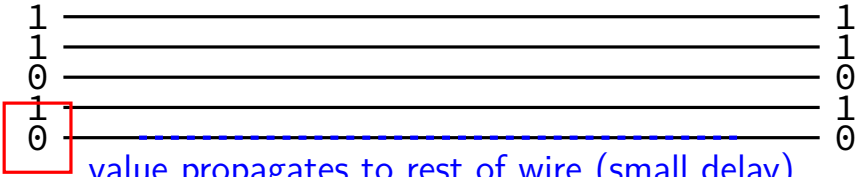


circuits: wires



binary value — actually voltage

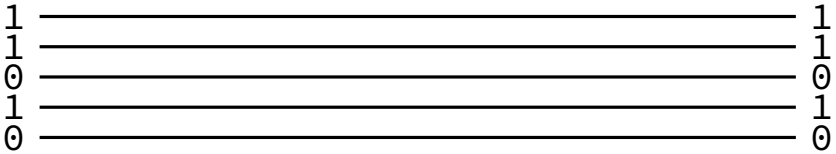
circuits: wires



value propagates to rest of wire (small delay)

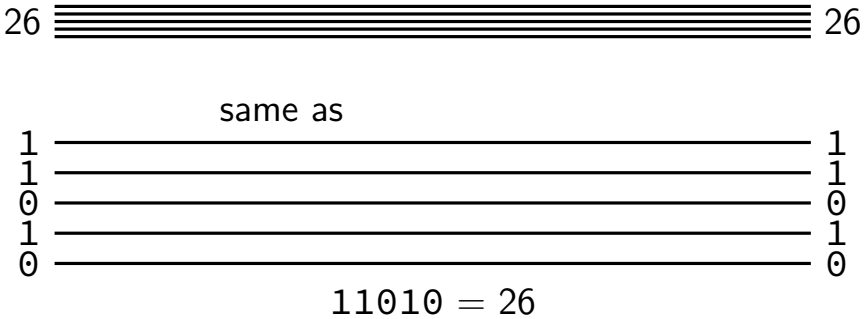
binary value — actually voltage

circuits: wire bundles

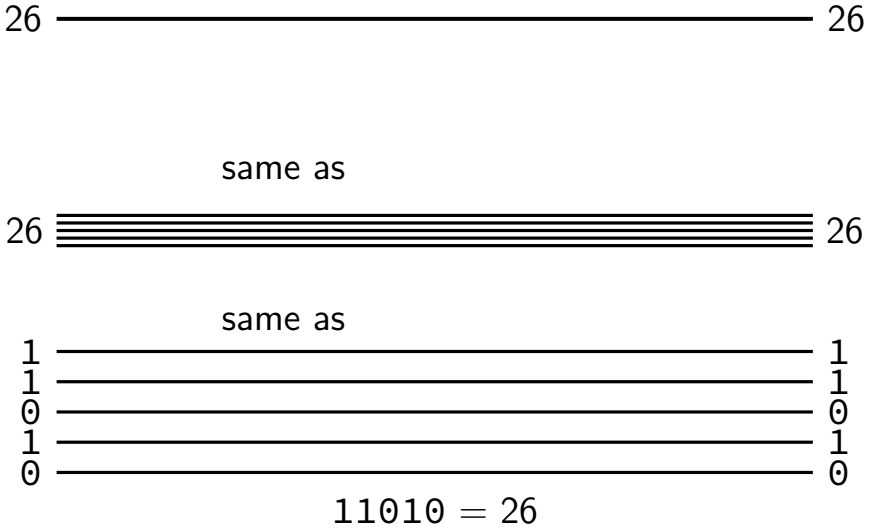


$\texttt{11010} = 26$

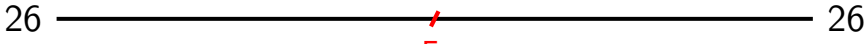
circuits: wire bundles



circuits: wire bundles



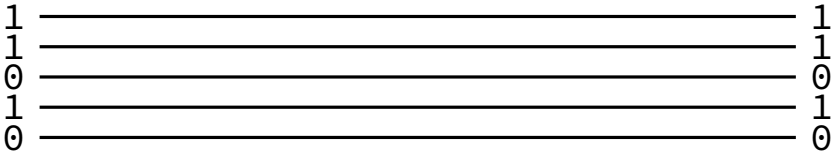
circuits: wire bundles



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

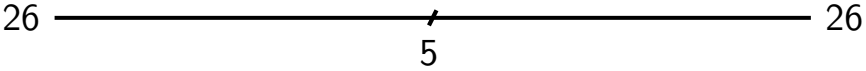
same as

same as



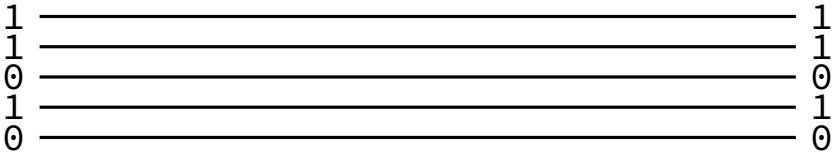
$\texttt{11010} = 26$

circuits: wire bundles



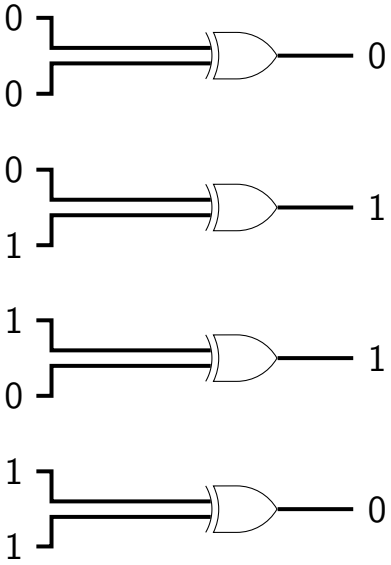
same as

same as



$\texttt{11010} = 26$

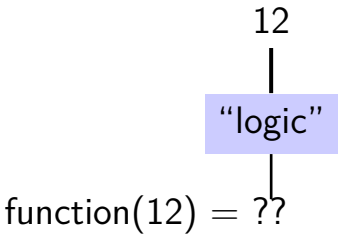
circuits: gates



circuits: logic

want to do calculations?

generalize gates:



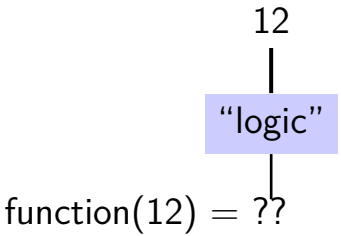
circuits: logic

want to do calculations?

generalize gates:

output wires contain result of function on input

- changes as input changes (with delay)



circuits: logic

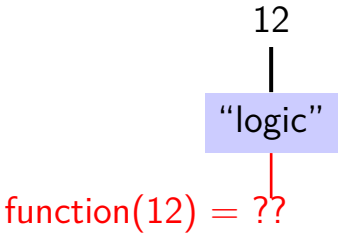
want to do calculations?

generalize gates:

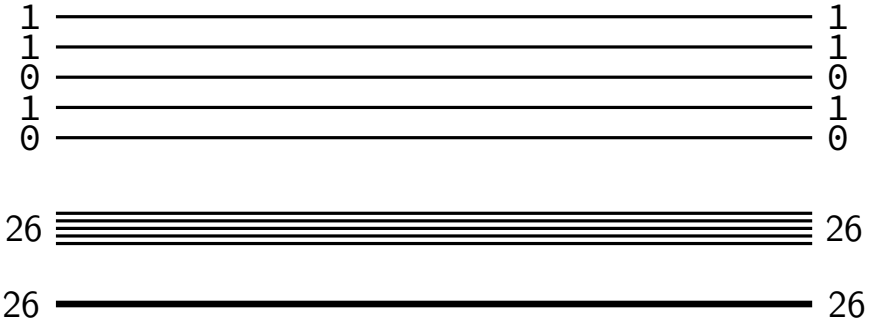
output wires contain result of function on input

- changes as input changes (with delay)

need not be same width as output



HCLRS: wire (bundle)s

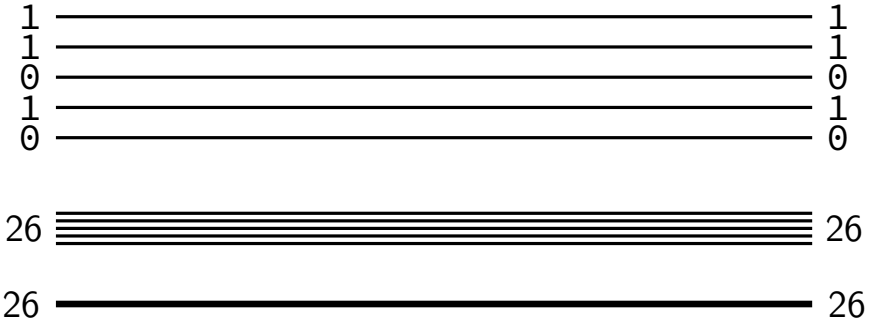



```
wire foo : 5; foo = 0b11010;
```
OR

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

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

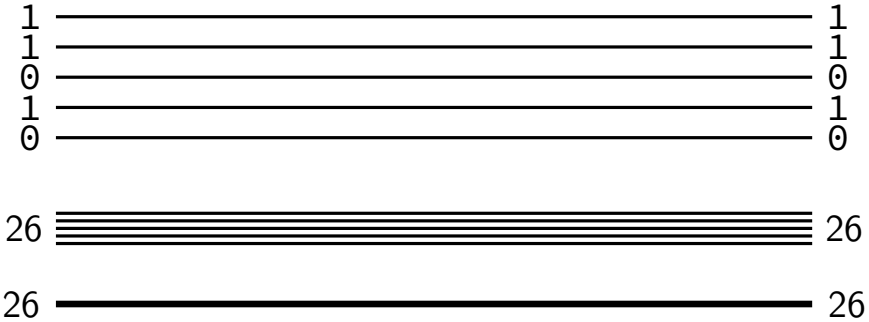
HCLRS: wire (bundle)s




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

name

HCLRS: wire (bundle)s



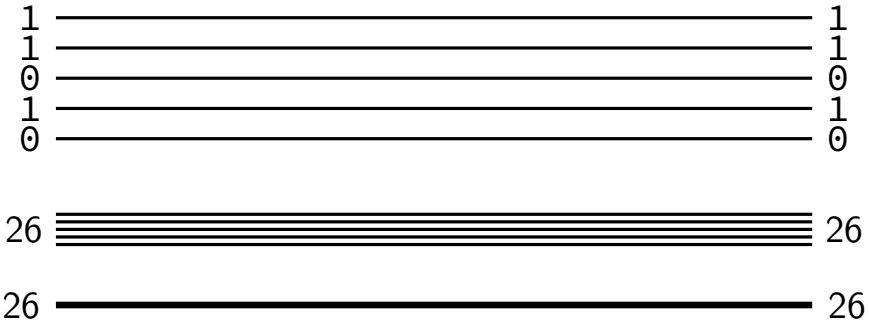

```
wire foo : 5; foo = 0b11010;
```
OR

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

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

width (in bits)

HCLRS: wire (bundle)s



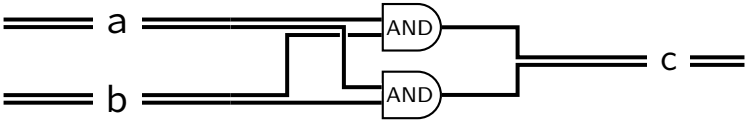

```
wire foo : 5;  foo = 0b11010;   OR
wire foo : 5;  foo = 26;        OR
wire foo : 5;  foo = 0x1a;
```

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)

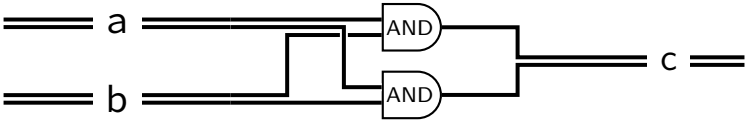
```
wire a : 2; wire b : 2; wire c : 2;
c = b & a;
a = 0b10;
b = 0b11;
```

same as

```
a = 0b10;
b = 0b11;
c = b & a;
```

order doesn't matter

connected or not

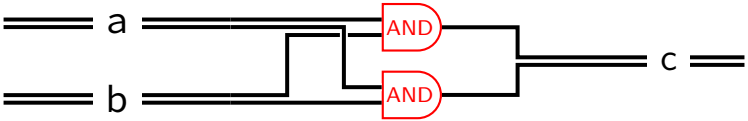


HCLRS: gates + calcuations (1)

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

C-like expressions supported

`0b10 & 0b11 = 0b10`

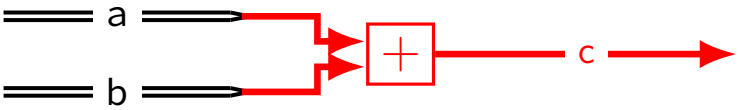


HCLRS: gates + calcuations (2)

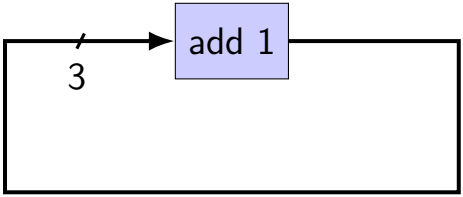
```
wire a : 2; wire b : 2; wire c : 2;
c = b + a; /* was b & a */
a = 0b10;
b = 0b11;
```

more than bitwise operators supported

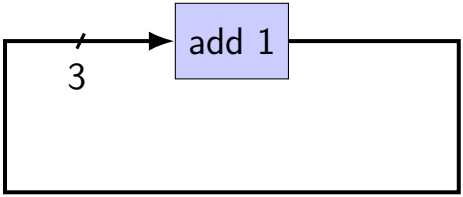
$0b10 + 0b11 = 0b101 \rightarrow 0b01$ (extra bits lost)



example: (broken) counter circuit (1)

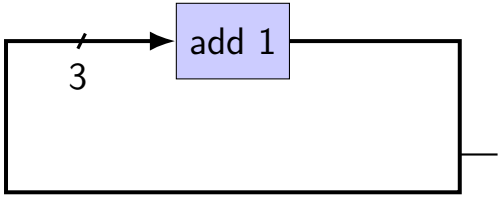


example: (broken) counter circuit (1)




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

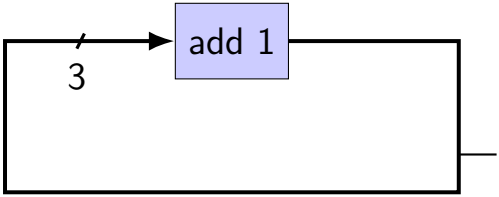
example: (broken) counter circuit (1)



time 0: `000`

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

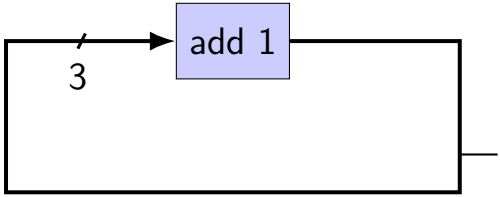
example: (broken) counter circuit (1)



time 0: `000` ← set how???

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

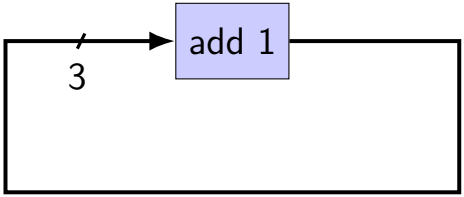
example: (broken) counter circuit (1)



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

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

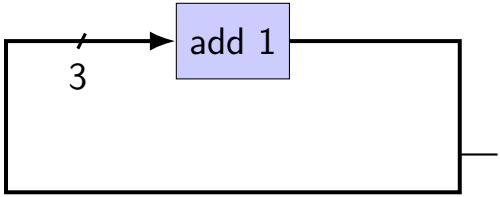
example: (broken) counter circuit (2)




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

HCLRS: compile error
"`Circular dependency detected: x depends on x`"

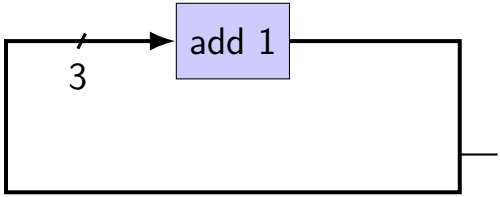
example: (broken) counter circuit (3)



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

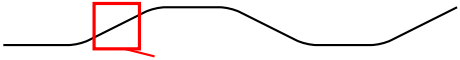
```
wire x : 3;
x = x + 1;
```

example: (broken) counter circuit (3)



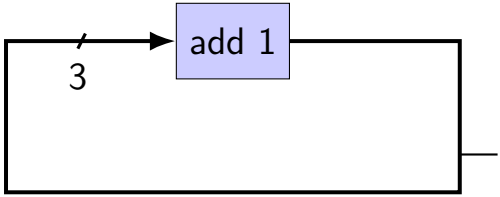
time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

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



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

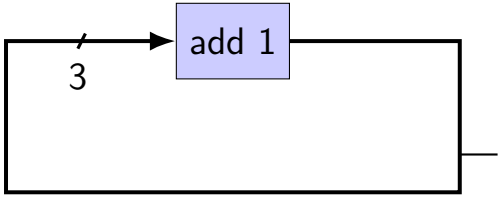
example: (broken) counter circuit (3)



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

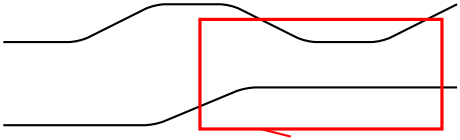
```
wire x : 3;
x = x + 1;
```

example: (broken) counter circuit (3)



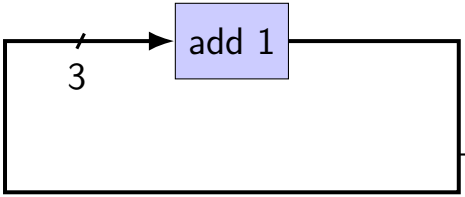
time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

```
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

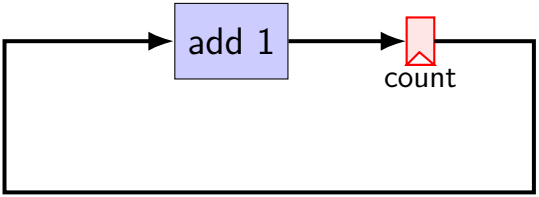
logic performs calculations all the time

never stores values!

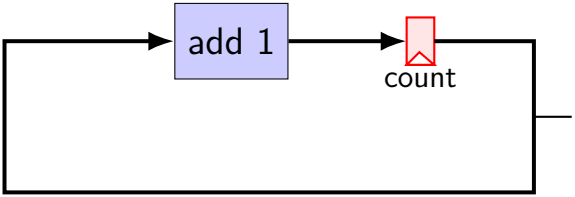
need extra elements to store values

- registers, memory

example: counter circuit (corrected)

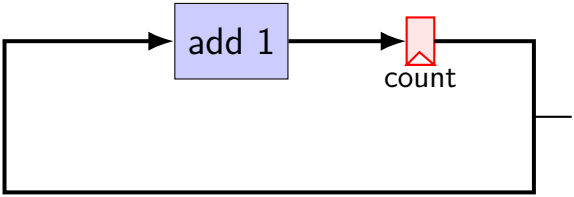


example: counter circuit (corrected)



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

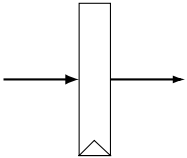
example: counter circuit (corrected)



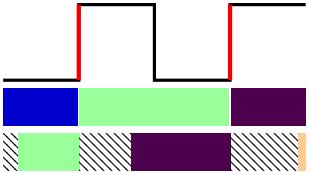
time 0: `000`
time 1: `001`
time 2: `010`
time 3: `011`

add register to store current count
updates based on "clock signal" (not shown)
avoids intermediate updates

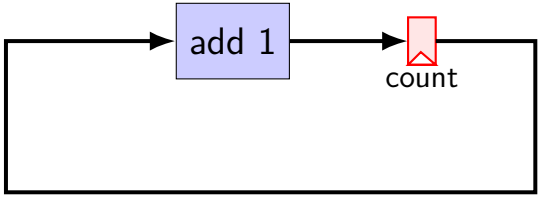
registers



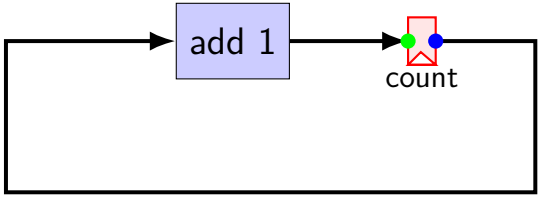
updates every clock cycle



example: counter circuit (real HCLRS)

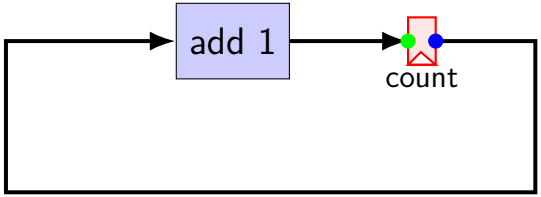


example: counter circuit (real HCLRS)




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

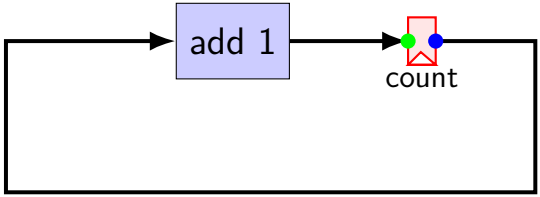
example: counter circuit (real HCLRS)




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

register "bank"
can have multiple (related) registers

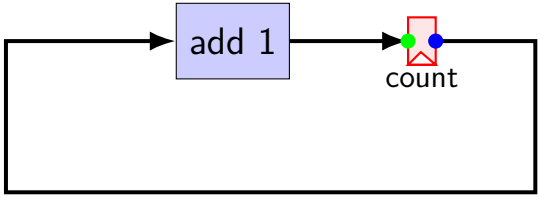
example: counter circuit (real HCLRS)



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;
```

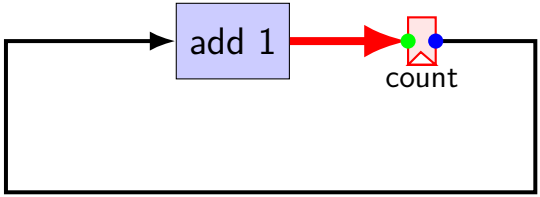
example: counter circuit (real HCLRS)



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

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

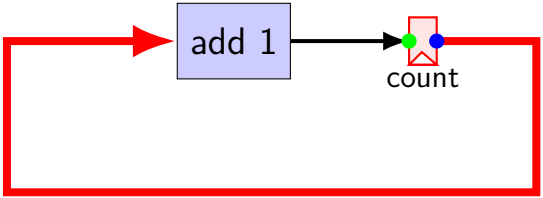
example: counter circuit (real HCLRS)




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

input wire to register

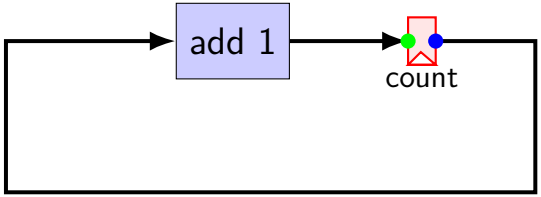
example: counter circuit (real HCLRS)




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

output wire of register

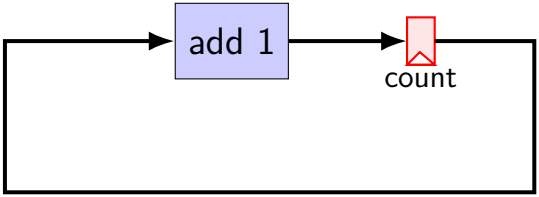
example: counter circuit (real HCLRS)



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

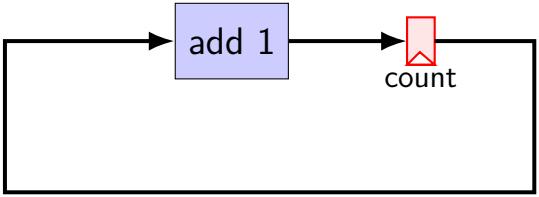
```
register xY {
    count : 3 = 0b000;
}
x_count = Y_count + 0b001;
```

example: counter circuit




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

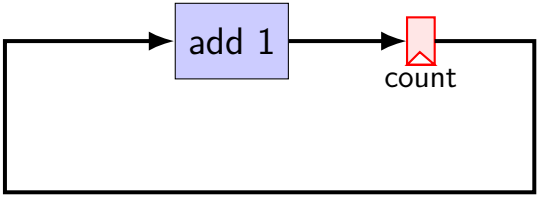
example: counter circuit




```
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
…	…	…

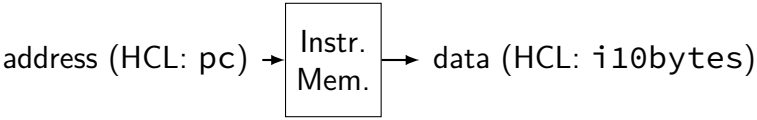
HCL circuit with registers

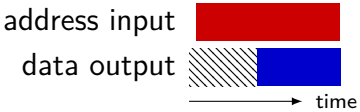
```
register xY {
    a : 4 = 1;  /* <-- initial Y_a */
    b : 4 = 1;  /* <-- initial Y_b */
}
x_b = x_a + Y_a;
x_a = Y_a + Y_b;
```

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:

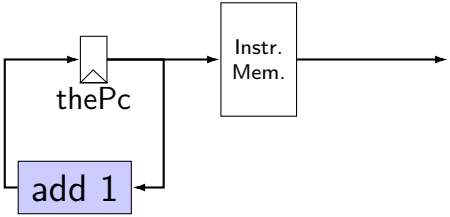
- `STAT_AOK` — keep going
- `STAT_HLT` — stop, normal shtdown
- `STAT_INS` — invalid instruction
- ...(and more errors)

(predefined 3-bit constants)

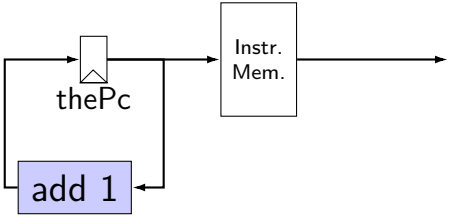
must be set

determines if simulator keeps going

nop CPU

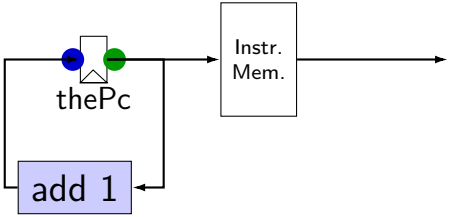


nop CPU



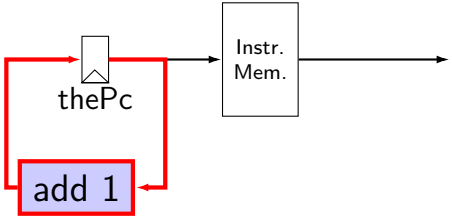

```
register pF {
   thePc : 64 = 0;
}
```

nop CPU



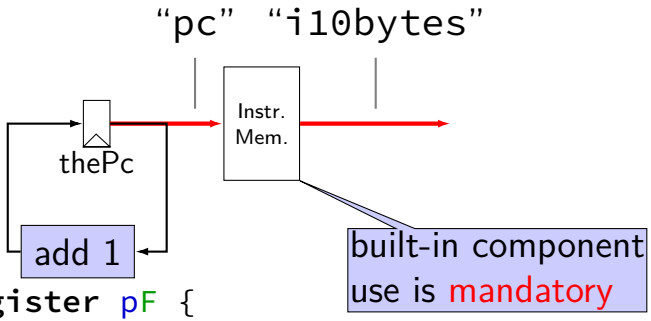

```
register pF {
   thePc : 64 = 0;
}
```

nop CPU



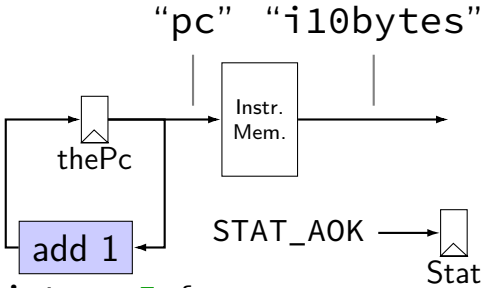

```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
```

nop CPU



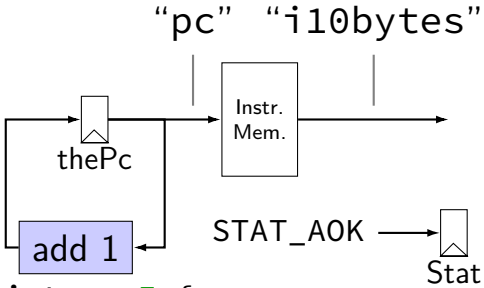

```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```

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                                       |
+----------------------------------------------------------------------+
```

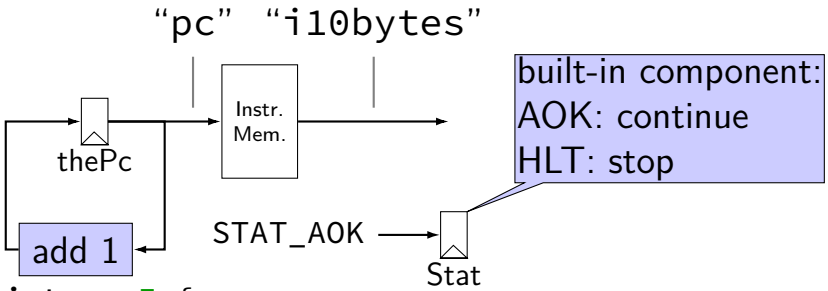
running a simulator (2)

```
$ ./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)

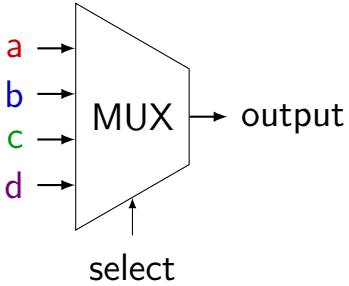
```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:                0   RCX:                0   RDX:                0 |
| RBX:                0   RSP:                0   RBP:                0 |
| RSI:                0   RDI:              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

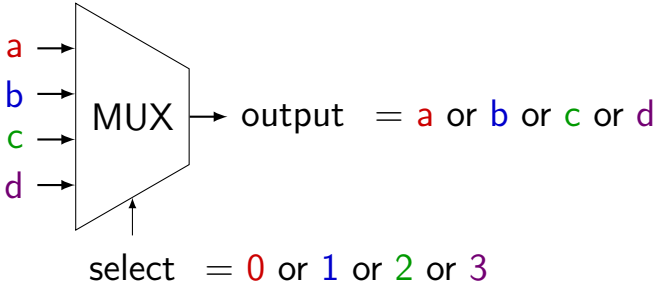



```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
Stat = STAT_AOK;
```

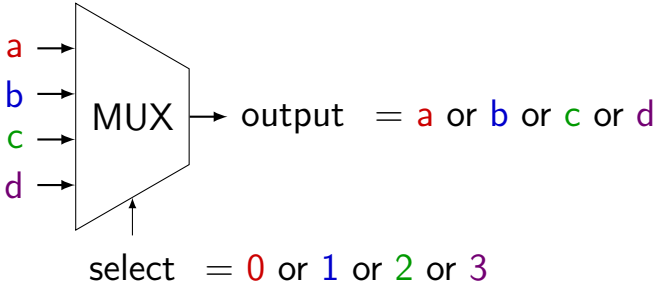
multiplexers



multiplexers



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

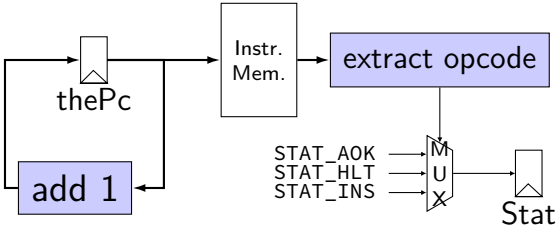
- encoding `10`

halt

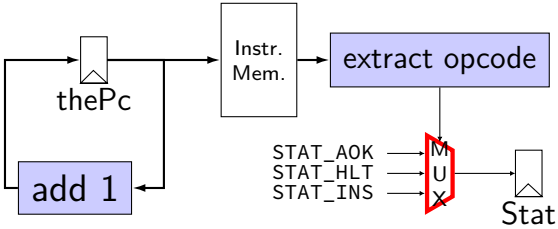
- encoding `00`

our strategy: MUX to decide using opcode

nop/halt CPU

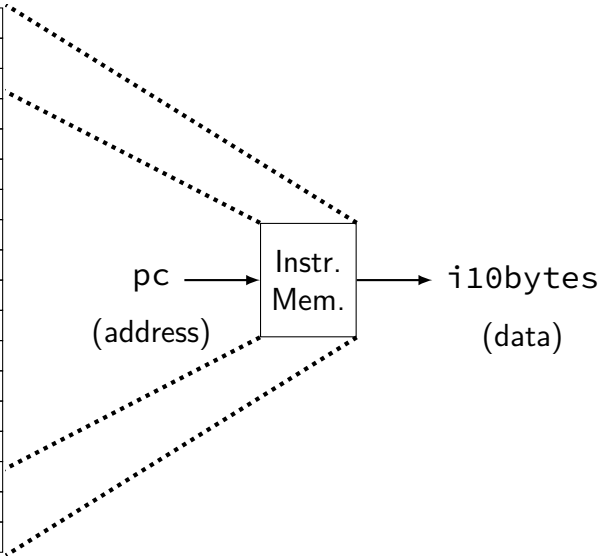

thePc
Instr.
Mem.
extract opcode
add 1
STAT_AOK
STAT_HLT
STAT_INS
MUX
Stat

nop/halt CPU


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

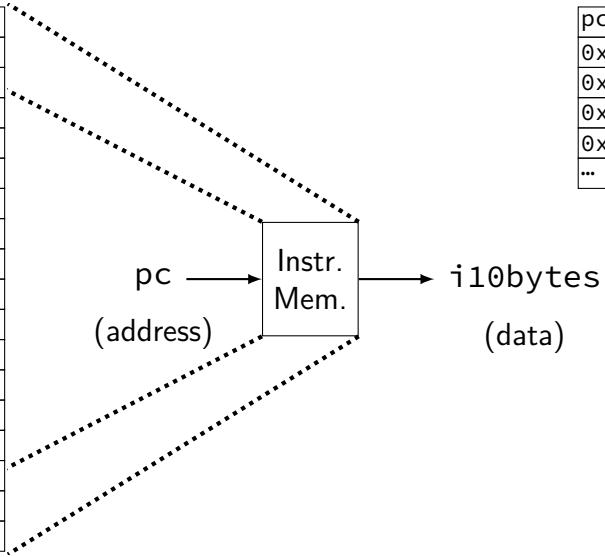
what is i10bytes?

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?

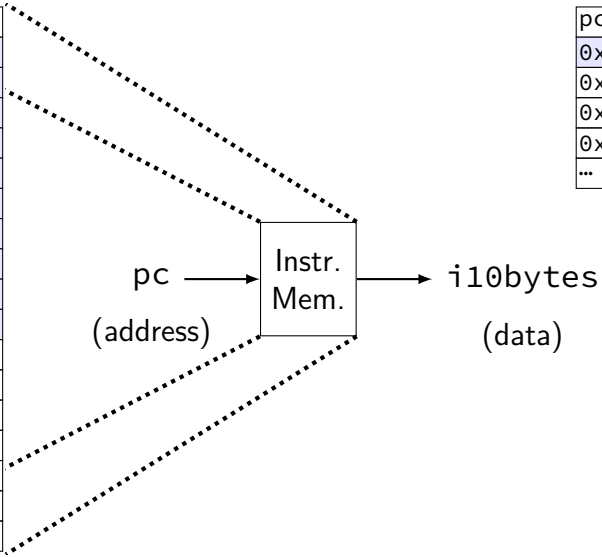
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



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

what is i10bytes?

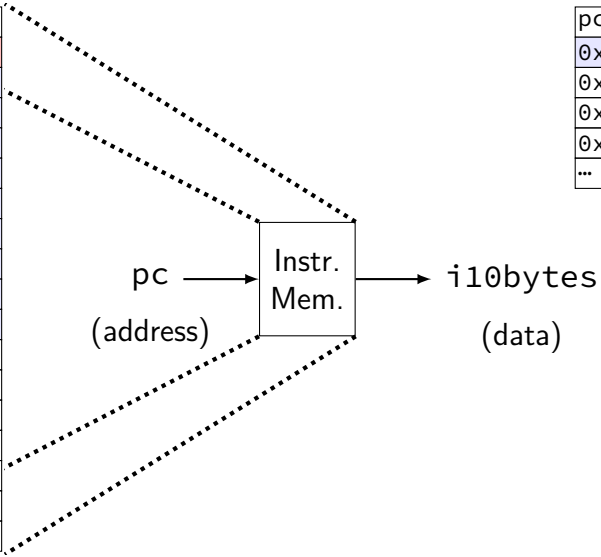
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



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

what is i10bytes?

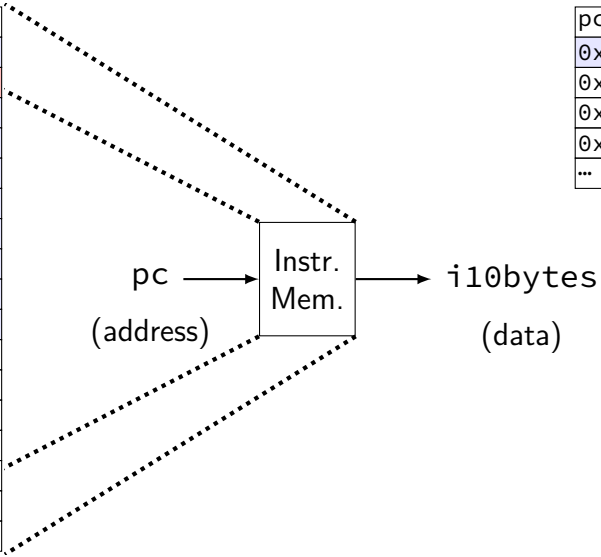
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



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

what is i10bytes?

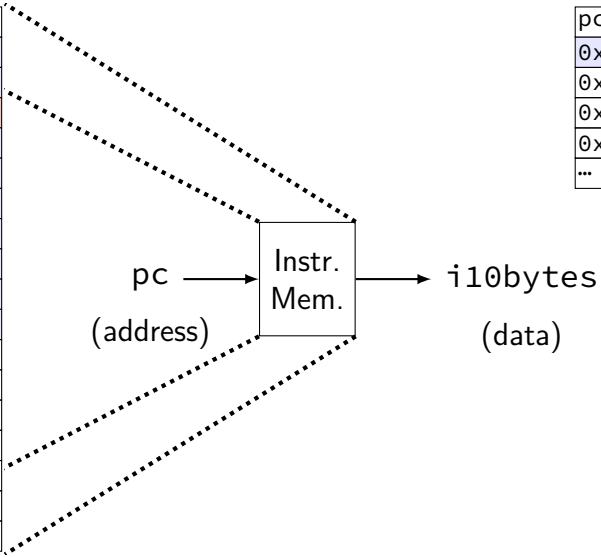
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x01000000000002161126 0
0x001	0x00010000000000216112
0x002	0x00000100000000002161
0x003	0x00000001000000000021
…	…

what is i10bytes?

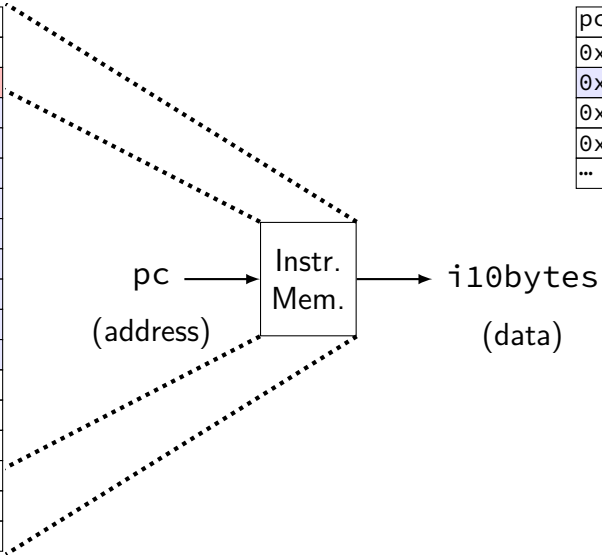
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



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

what is i10bytes?

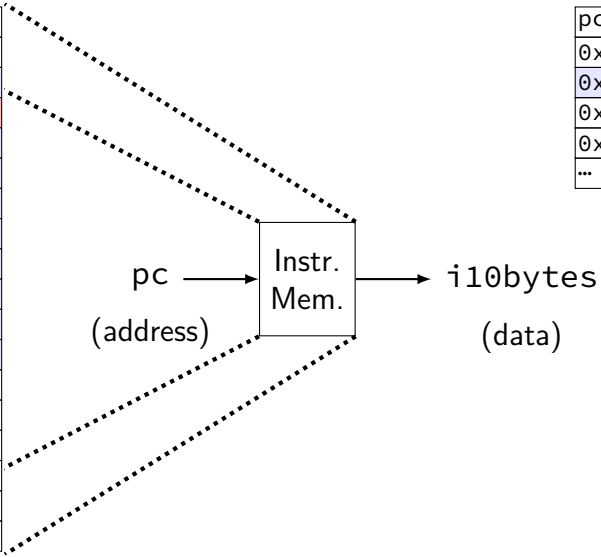
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



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

what is i10bytes?

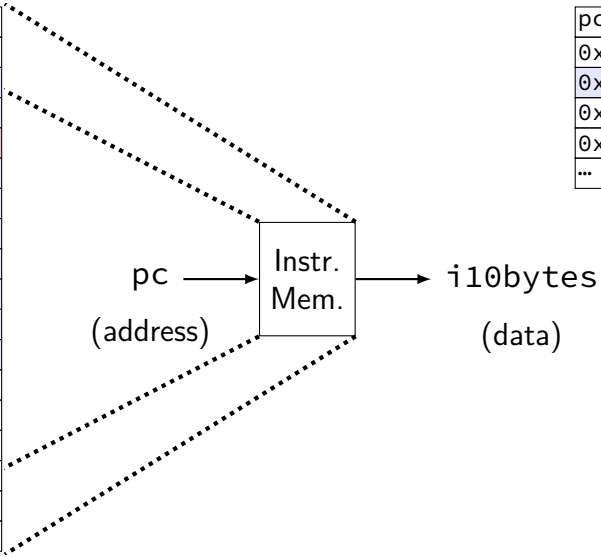
addr.	value
0x000	0x60
0x001	0x12
0x002	0x61
0x003	0x21
0x004	0x00
0x005	0x00
0x006	0x00
0x007	0x00
0x008	0x00
0x009	0x00
0x00a	0x01
0x00b	0x00
0x00c	0x00
0x00d	0x00
0x00e	0x00
0x00f	0x00
…	…



pc	i10bytes
0x000	0x01000000000021611260
0x001	0x00010000000000021612
0x002	0x00000100000000002161
0x003	0x00000001000000000021
…	…

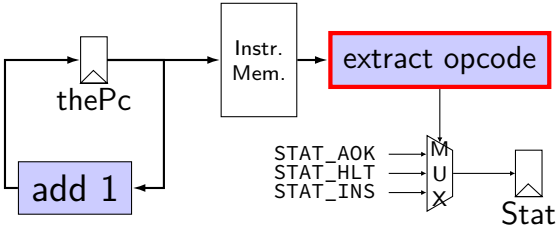
what is i10bytes?

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
…	…

nop/halt CPU


thePc
Instr.
Mem.
extract opcode
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:

... ... 2 F A 0

... ... 0010 1111 1010 0000

most sig. bit (bit 80) (bit 15) (bit 7) least sig. bit (bit 0)

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*								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

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)

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

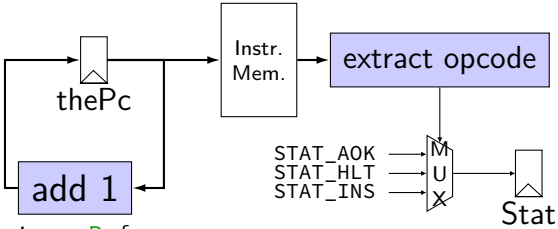
most sig. 4 bits of byte 1: bits 12–16

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

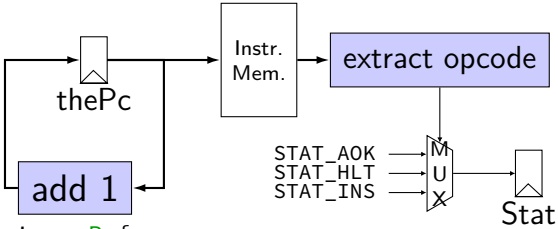
least sig. 4 bits of byte 1: bits 8–12

nop/halt CPU



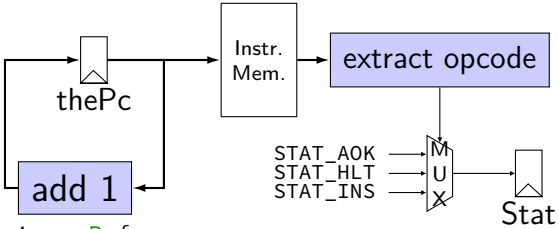

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

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)
];
```

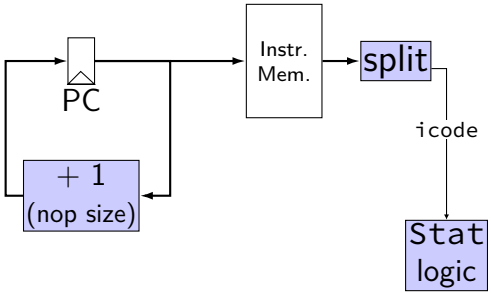
nop/halt CPU



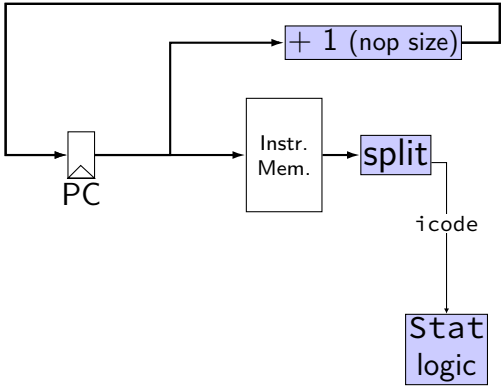

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; // (default case)
];
```

demo

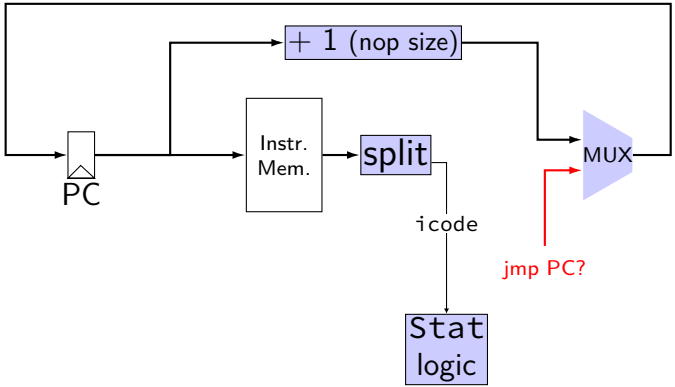
nop/halt → nop/jmp CPU



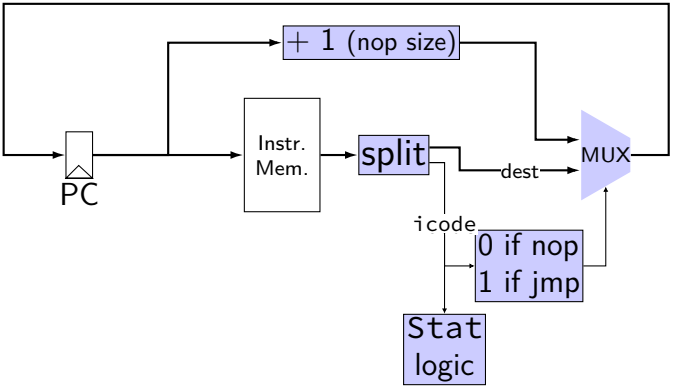
nop/halt → nop/jmp CPU



nop/halt → nop/jmp CPU



nop/halt → nop/jmp CPU



backup slides

comparing to yis

```
$ ./hclrs nopjmp_cpu.hcl nopjmp.yo
...
...
+--------------------- (end of halted state) ---------------------------+
Cycles run: 7
$ ./tools/yis nopjmp.yo
Stopped in 7 steps at PC = 0x1e.  Status 'HLT', CC Z=1 S=0 O=0
Changes to registers:

Changes to memory:
```

HCLRS summary

declare/assign values to wires

MUXes with

```
[ test1: value1; test2: value2; 1: default; ]
```

register banks with **`register i0`**:

next value on `i_name`; current value on `o_name`

fixed functionality

register file (15 registers; 2 read + 2 write)
memories (data + instruction)
Stat register (start/stop/error)