# lab post-mortem / HCL

# Changelog

17 September: broken counter circuit: correct wire voltage diagrams to match values at time 0/1/2/3 better

# last time

RISC v CISC

typical RISC properties

- separate memory access instructions, few ways to specify operands
- more registers (make up for separate memory access)
- simpler instructions generally
- hope: simple instructions are fast

Y86-64: kinda RISC-like variation on x86-64

Y86-64 encoding/decoding

# the lab generally

yes, took a lot more time than my estimate

overall asssessment: difficulty of parts of lab mostly okay (assuming issues with clarity of benchmark materials fixed), but too much stuff

# lab part 1: yas

common misconception re: `yas`

basically produces an executable, not object file

yes, the extension `.yo` is confusing!

so, no symbol table+relocations — everything ready to load into memory

use in lab: to get correct machine code

...but *you* needed to handle relocations+symbols

# lab part 1: relocations/symbol table entry

symbol table entries to remember where labels are:

```
foo:  /* <- label defintion, becomes symbol table entry */
```

---

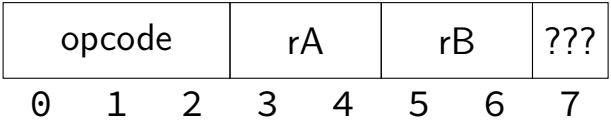
relocations to fill in addresses:

```
    call foo  /* <- label use, needs relocation */
    jmp foo  /* <- label use, needs relocation */
    irmovq $foo, %rax  /* <- label use, needs relocation */
```

# lab part 2: register availability

8-bit instructions, 3-bit opcode

instruction encoding with two operands:




e.g if opcode for add is `001`, then `add %r2, %r3` might become `001 10 11 0`

2-bit register numbers → 4 total registers

(maybe we should consider adding a bit to the opcode?)

# lab part 2: register availability (alt)

8-bit instructions, 3-bit opcode

instruction encoding with three operands:


e.g. `add %r1, %r1, %r0` might become
`001 1 1 0 00`

1 bit register numbers → 2 total registers

(probably, we should reconsider this instruciton size)

# lab part 3a: more instructions

question about: will compiler use more/less instructions with less registers

example of more instructions **(not what lab asked for)**:

(function accumulating into a local variable, then returning it)

```
/* with less registers */
...
movq $0, 8(%rsp)
... lots of computation ...
pushq %r8
movq 8(%rsp), %r8
subq %r8, 128(%rdi)
popq %r8
... lots of computation ...
addq %rax, 8(%rsp)
movq 8(%rsp), %rax
subq $16, %rsp
ret
```

```
/* with more registers */
...
movq $0, %r15
... lots of computation ...
/* omitted instruction */
/* omitted instruction */
subq %r15, 128(%rdi)
/* omitted instruction */
... lots of computation ...
addq %r15, %rax
/* omitted instrution */
subq $8, %rsp
ret
```

# lab part 3a: less instructions

question about: will compiler use more/less instructions with less registers

example of less instructions (but probably slower/longer machine code!)

(function accumulating into a global variable)

```
/* with less registers */
/* omitted instruction */
... lots of computation ...
addq %rax, global_variable
... lots of computation ...
addq %rax, global_variable
... lots of computation ...
addq %rax, global_variable
/* omitted instruction */
ret
```

```
/* with more registers */
movq global_variable, %r15
... lots of computation ...
addq %rax, %r15
... lots of computation ...
addq %rax, %r15
... lots of computation ...
addq %rax, %r15
movq %r15, global_variable
ret
```

# lab part 3b (1)

anonymous feedback:

> The expectation that we spend hours pouring over terribly formatted data just to do a long series of basic math operations as part of a hour lab is a waste of our time and teaches nothing.

I have a problem of opening text files in very wide text editors and not thinking about how it would look on smaller screens.

I probably should have added an "instructions that read or write memory" row to the data files

I probably should have written "instructions that don't use memory" instead of "instructions that only use registers"

> I expected (all instructions - instructions using memory)

# lab part 3b (2)

anonymous feedback:

> The expectation that we spend hours pouring over terribly formatted data just to do a long series of basic math operations as part of a hour lab is a waste of our time and teaches nothing.

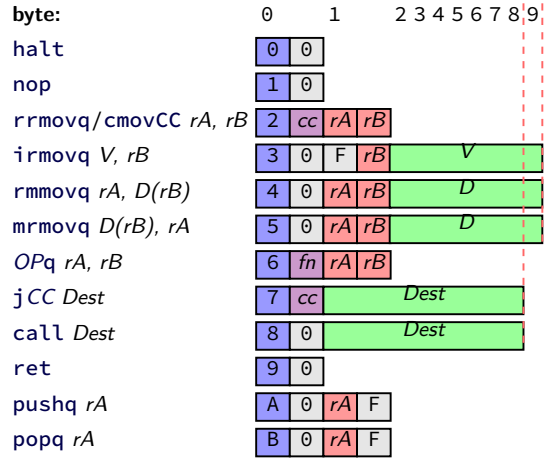
Hope was to help with intuition for answering "does doing X help performance"

Also reveal that giving compilers a lot more registers isn't super helpful

Expected this to be a pretty quick part of the lab

# Y86-64 decoding

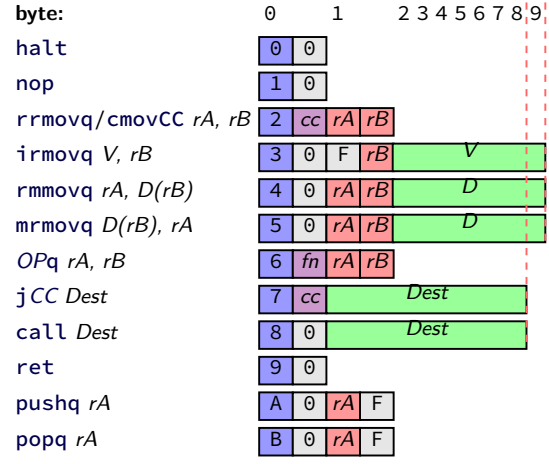
```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00 00
```



# Y86-64 decoding

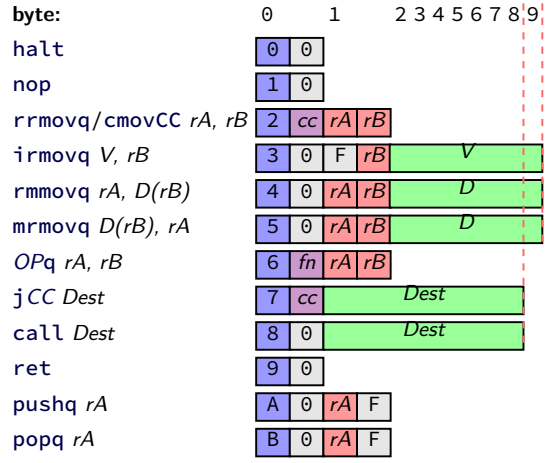
```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

exercise: types of first three instructions?



# Y86-64 decoding

```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

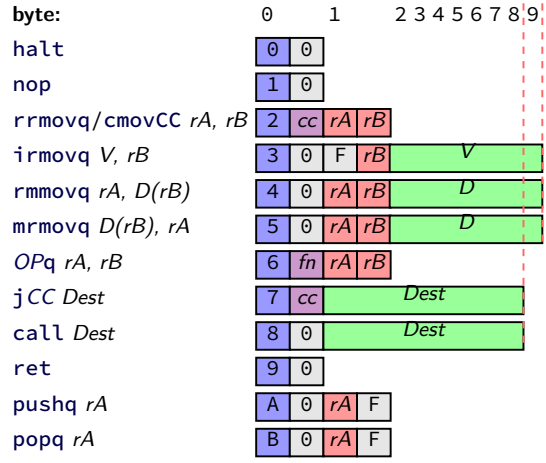


# Y86-64 decoding

```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

`rrmovq %rcx, %rax`

- $0$ as cc: always
- $1$ as reg: `%rcx`
- $0$ as reg: `%rax`

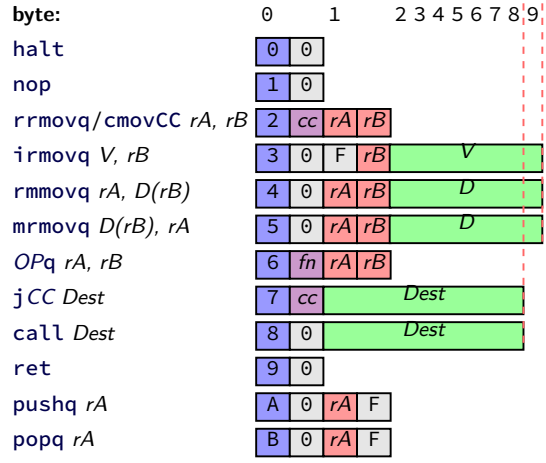


# Y86-64 decoding

```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

```
rrmovq %rcx, %rax
addq   %rdx, %rax
subq   %rbx, %rdi
```

- $0$ as fn: add
- $1$ as fn: sub

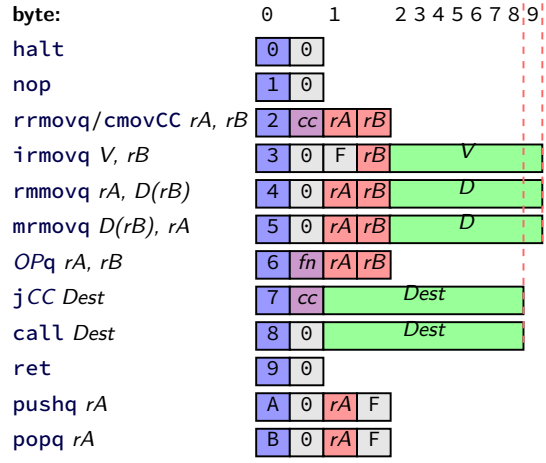


# Y86-64 decoding

```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

```
rrmovq %rcx, %rax
addq   %rdx, %rax
subq   %rbx, %rdi
jl     0x84
```

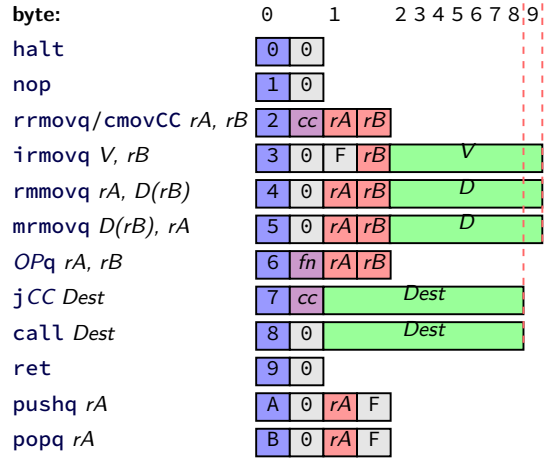
- 2 as cc: `l` (less than)
- hex 84 00… as little endian *Dest: 0x84*



# Y86-64 decoding

```
20 10 60 20 61 37 72 84 00 00 00 00 00 00 00
20 12 20 01 70 68 00 00 00 00 00 00 00
```

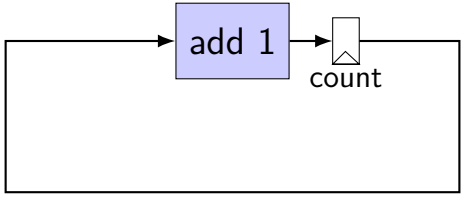
```
rrmovq %rcx, %rax
addq   %rdx, %rax
subq   %rbx, %rdi
jl     0x84
rrmovq %rcx, %rdx
rrmovq %rax, %rcx
jmp    0x68
```



# 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

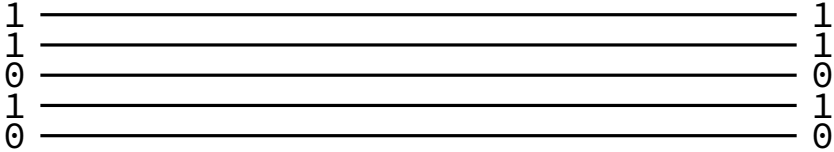
# our tool: HCLRS

built for this course

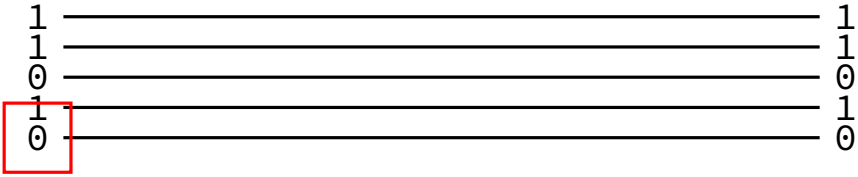
assumes you're making a processor

somewhat different from textbook's HCL

# circuits: wires

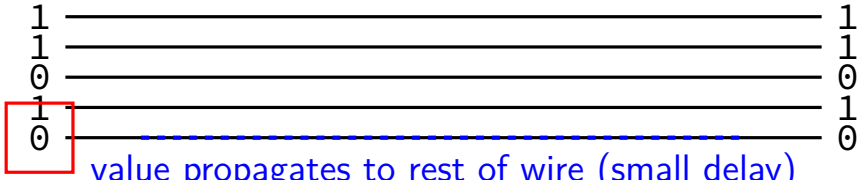


# circuits: wires

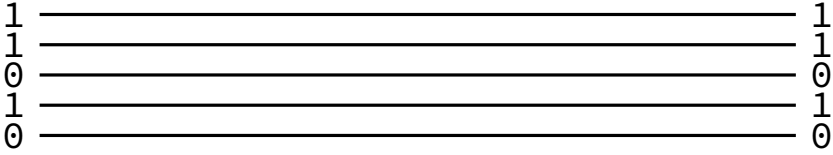



binary value — actually voltage

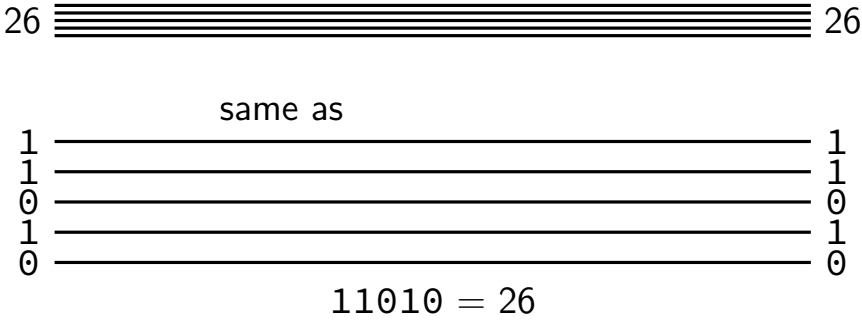
# circuits: wires



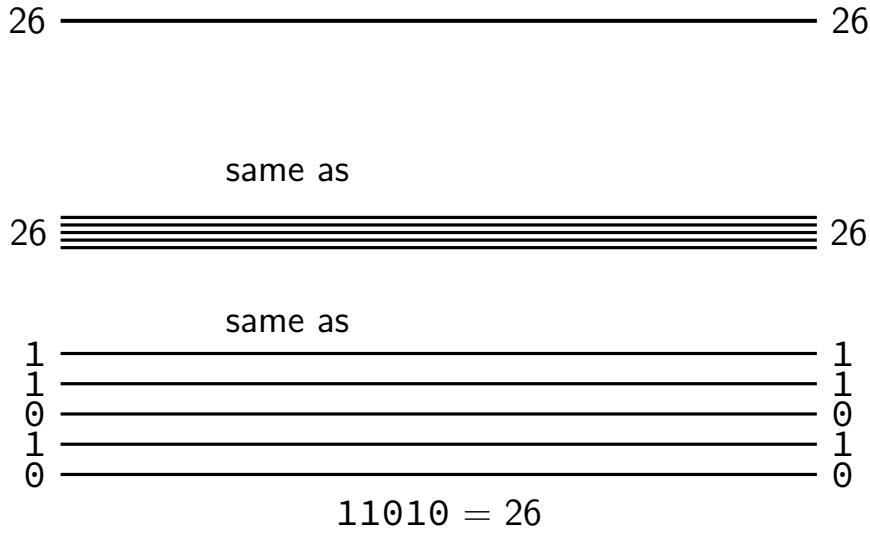
# circuits: wire bundles



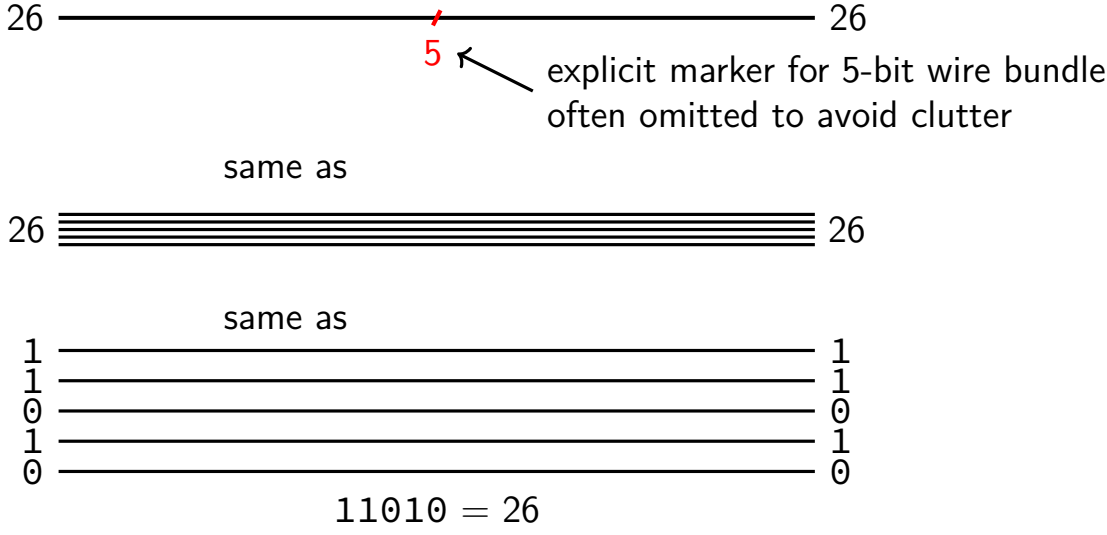
# circuits: wire bundles



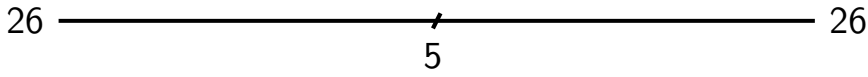
# circuits: wire bundles



# circuits: wire bundles

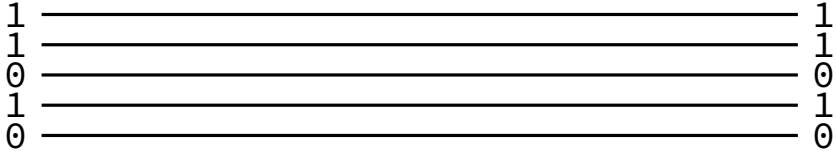


# circuits: wire bundles

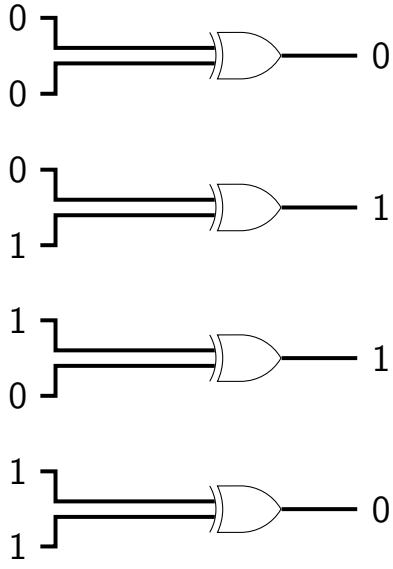



same as


same as



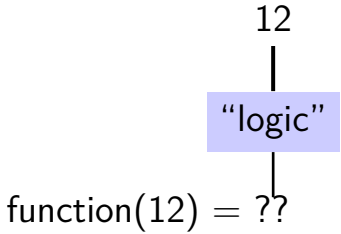
# circuits: gates



# circuits: logic

want to do calculations?

generalize gates:



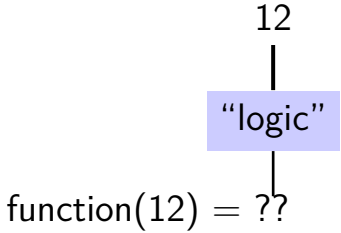
# circuits: logic

want to do calculations?

generalize gates:

output wires contain result of function on input

changes as input changes (with delay)



# circuits: logic

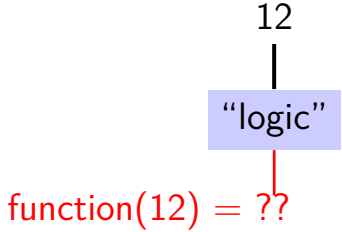
want to do calculations?

generalize gates:

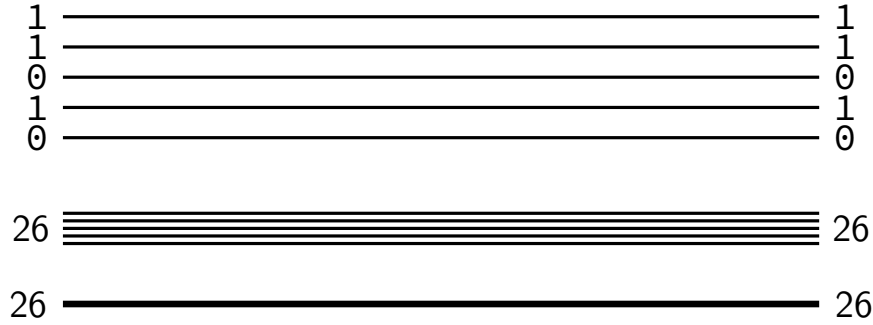
output wires contain result of function on input

changes as input changes (with delay)

need not be same width as output

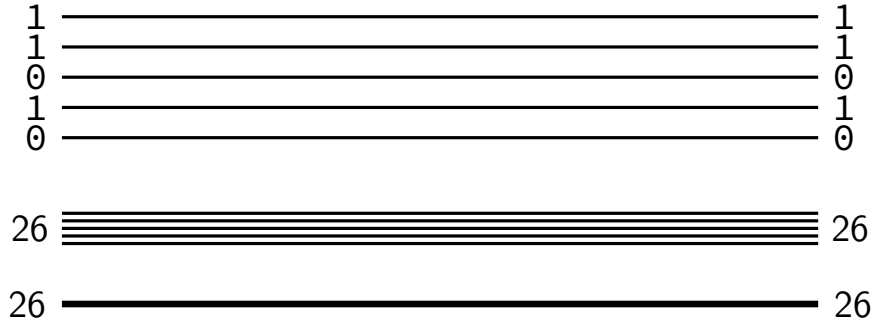


# HCLRS: wire (bundle)s




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

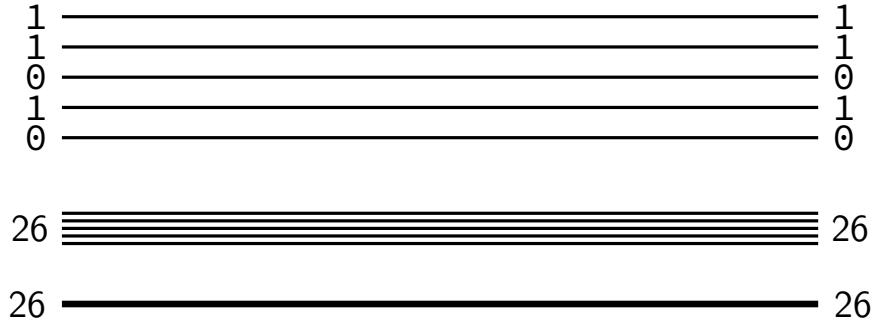
# HCLRS: wire (bundle)s




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

*name*

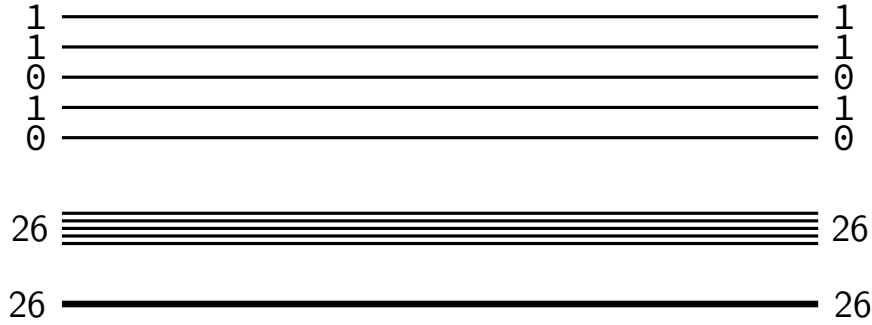
# HCLRS: wire (bundle)s




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

*width* (in bits)

# HCLRS: wire (bundle)s



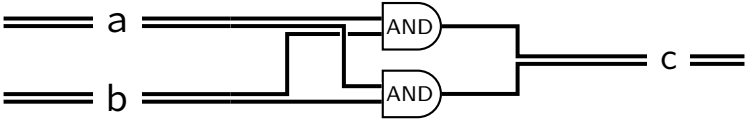

```
wire foo : 5; foo = 0b11010;    OR
wire foo : 5; foo = 26;         OR
wire foo : 5; foo = 0x1a;
```

*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;
```

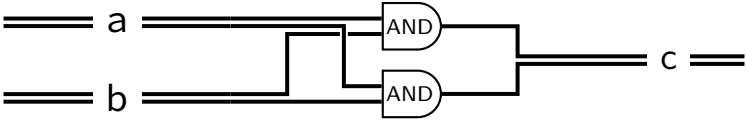


# HCLRS: gates + calcuations (1)

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

**order doesn't matter**

connected or not

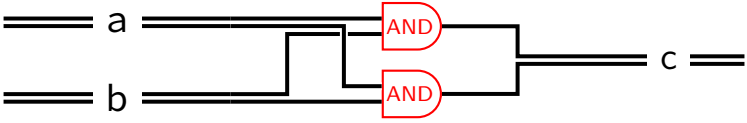


# HCLRS: gates + calcuations (1)

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

C-like expressions supported

`0b10 & 0b11 = 0b10`

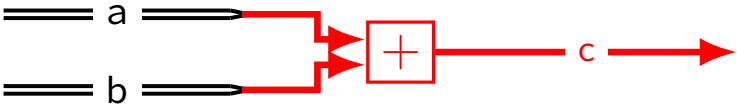


# HCLRS: gates + calcuations (2)

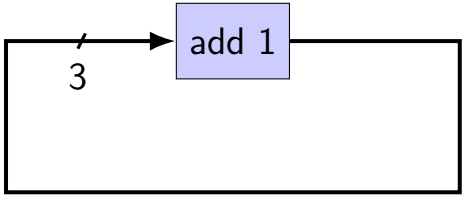
```
wire a : 2; wire b : 2; wire c : 2;
c = b + a; /* was b & a */
a = 0b10;
b = 0b11;
```

more than bitwise operators supported

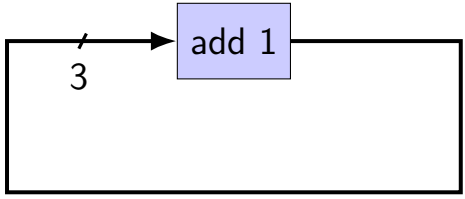
$0b10 + 0b11 = 0b101 \rightarrow 0b01$ (extra bits lost)



# example: (broken) counter circuit (1)

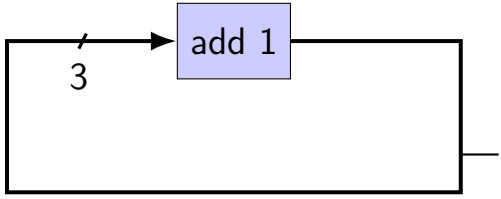


# example: (broken) counter circuit (1)




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

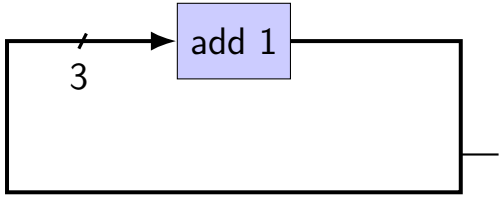
# example: (broken) counter circuit (1)




time 0: `000`

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

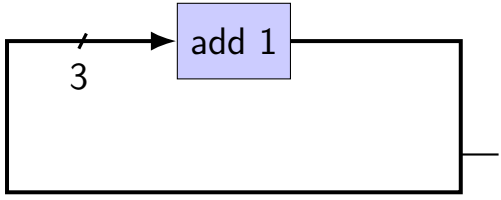
# example: (broken) counter circuit (1)




time 0: `000` ← set how???

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

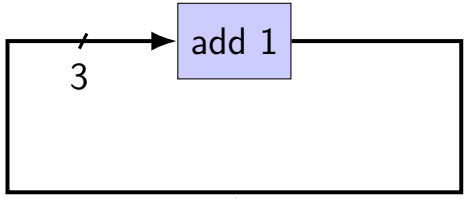
# example: (broken) counter circuit (1)




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

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

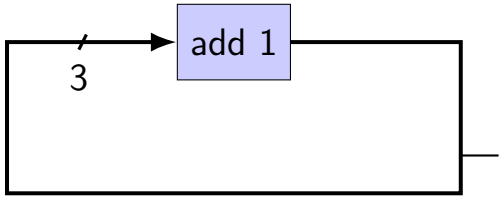
# example: (broken) counter circuit (2)




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

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

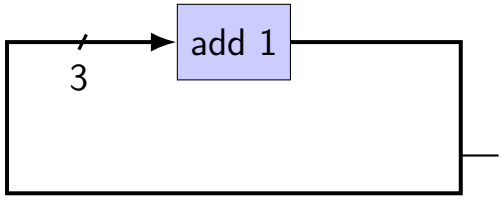
# example: (broken) counter circuit (3)




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

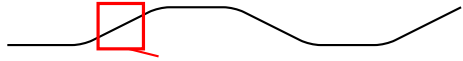
```
wire x : 3;
x = x + 1;
```

# example: (broken) counter circuit (3)



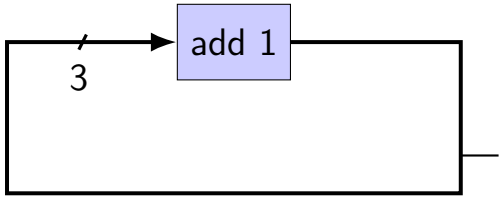

time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

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



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

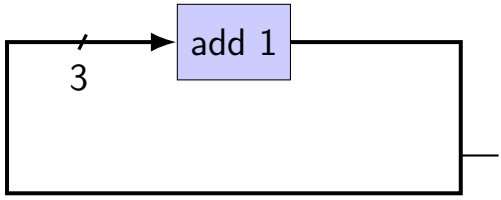
# example: (broken) counter circuit (3)




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

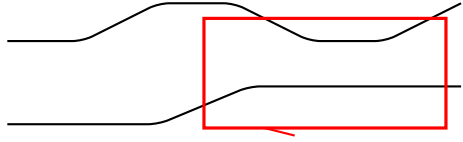
```
wire x : 3;
x = x + 1;
```

# example: (broken) counter circuit (3)



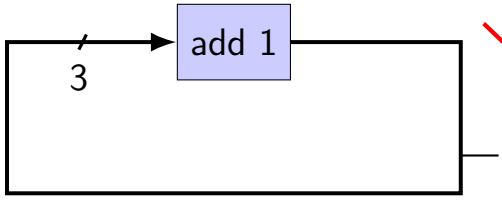

time 0: `000`
time 1: `001?`
time 2: `010?`
time 3: `011?`

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



problem 2: changes not in sync?

# example: (broken) counter circuit (4)




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

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

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

# circuits: state

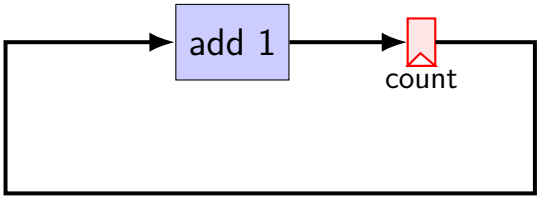
logic performs calculations all the time

never stores values!

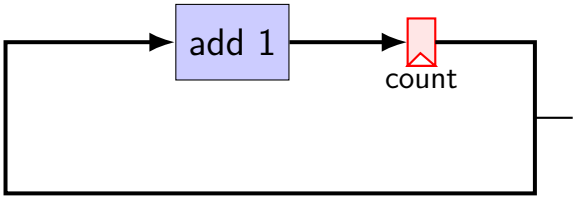
need extra elements to store values

- registers, memory

# example: counter circuit (corrected)

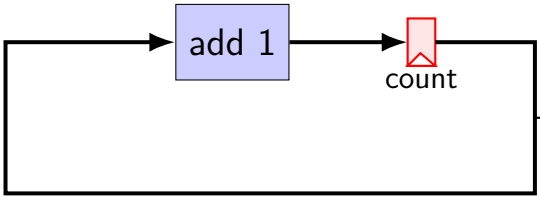


# example: counter circuit (corrected)




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

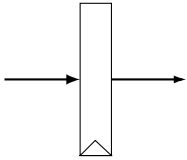
# example: counter circuit (corrected)



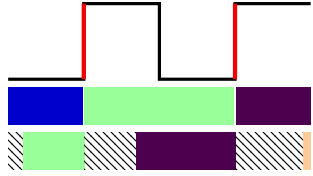

time 0: `000`
time 1: `001`
time 2: `010`
time 3: `011`

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

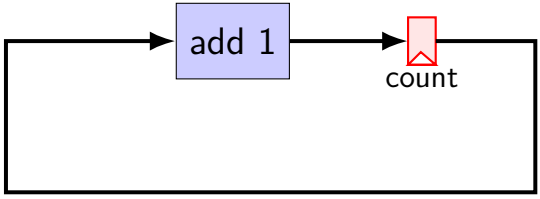
# registers



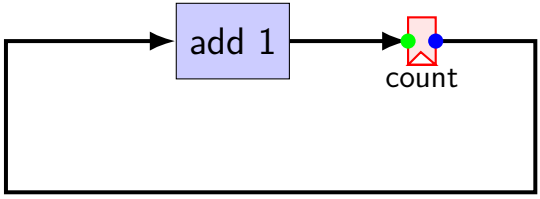
updates every clock cycle



# example: counter circuit (real HCLRS)

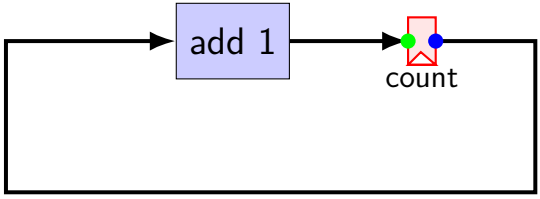


# example: counter circuit (real HCLRS)




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

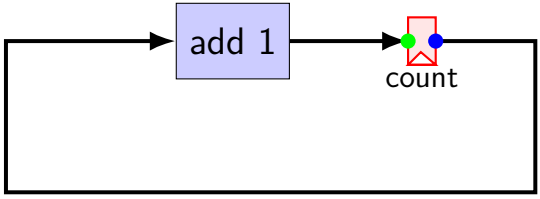
# example: counter circuit (real HCLRS)




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

register "bank"
can have multiple (related) registers

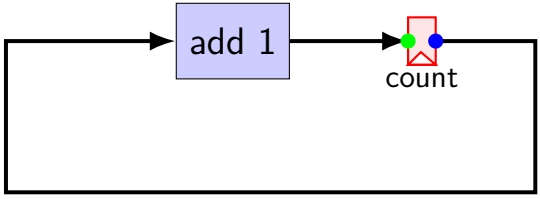
# example: counter circuit (real HCLRS)




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;
```

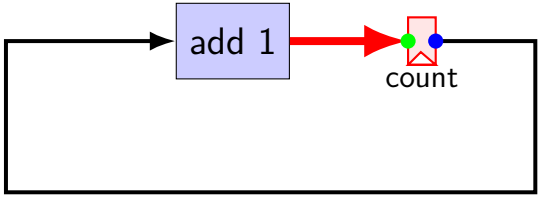
# example: counter circuit (real HCLRS)




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

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

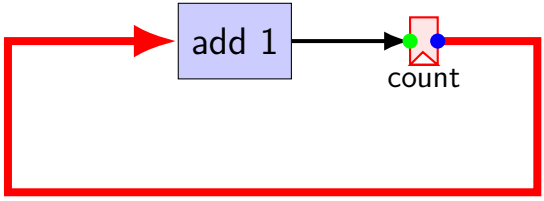
# example: counter circuit (real HCLRS)




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

input wire to register

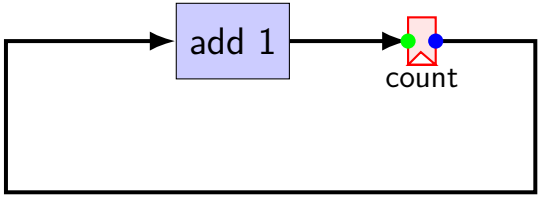
# example: counter circuit (real HCLRS)




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

output wire of register

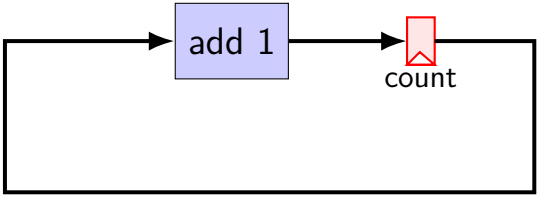
# example: counter circuit (real HCLRS)




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

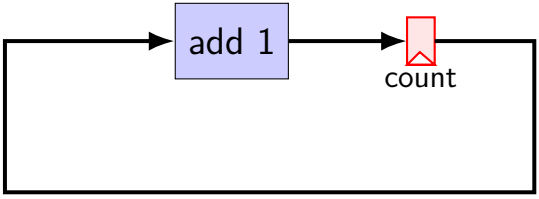
```
register  xY  {
     count  : 3 =  0b000 ;
}
 x_count  =  Y_count  + 0b001;
```

# example: counter circuit




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

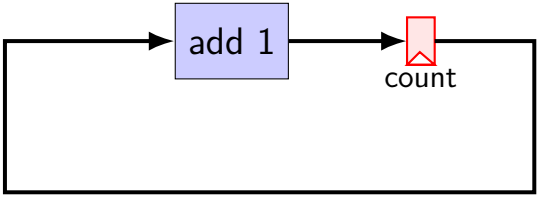
# example: counter circuit




```
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 |
| ... | ... | ... |

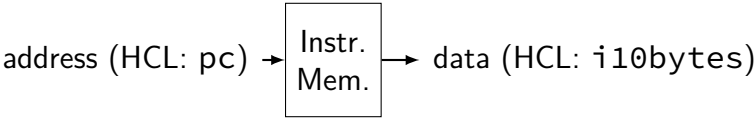
# HCL circuit with registers

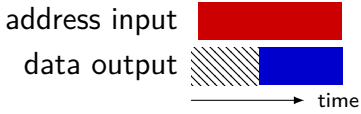
```
register xY {
    a : 4 = 1;  /* <-- initial Y_a */
    b : 4 = 1;  /* <-- initial Y_b */
}
x_b = x_a + Y_a;
x_a = Y_a + Y_b;
```

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:

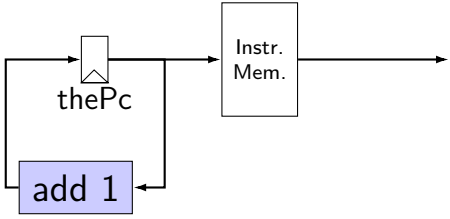
- `STAT_AOK` — keep going
- `STAT_HLT` — stop, normal shtdown
- `STAT_INS` — invalid instruction
- ...(and more errors)

(predefined 3-bit constants)

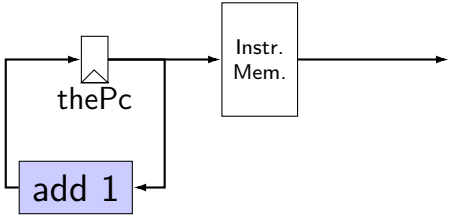
must be set

determines if simulator keeps going

# nop CPU

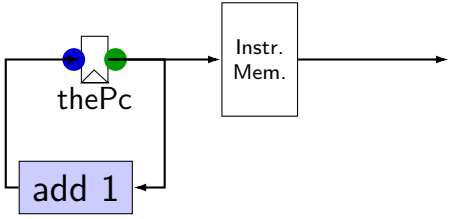


# nop CPU



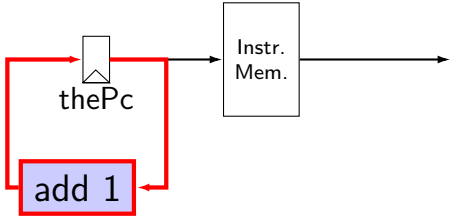

```
register pF {
    thePc : 64 = 0;
}
```

# nop CPU



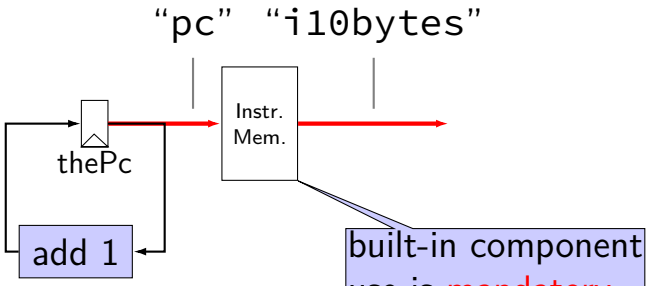

```
register pF {
    thePc : 64 = 0;
}
```

# nop CPU



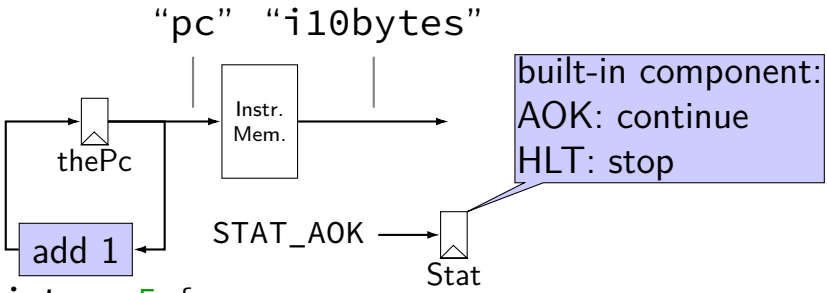

```
register pF {
    thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
```

# nop CPU



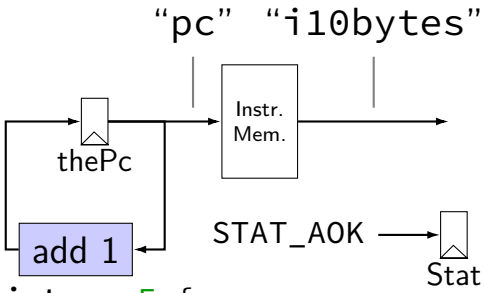

```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```

# 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)

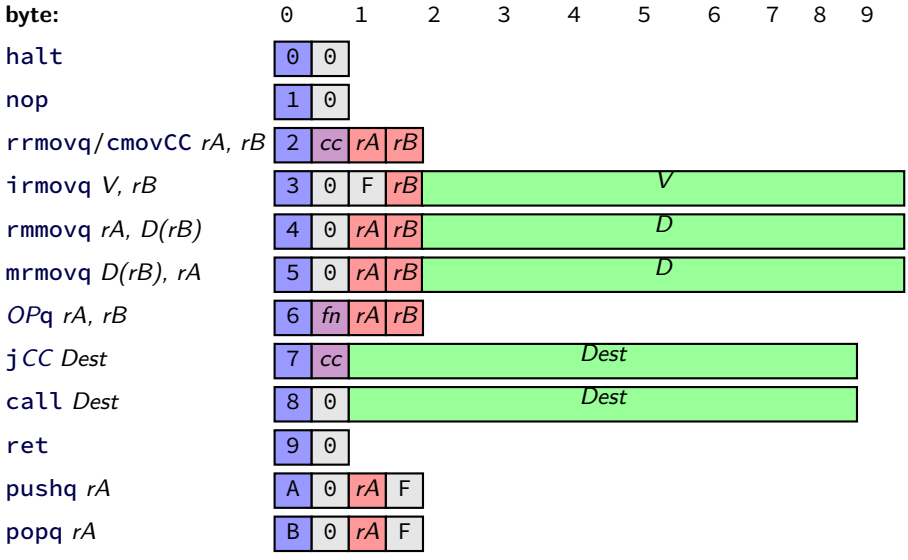
```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:                0   RCX:                0   RDX:                0 |
| RBX:                0   RSP:                0   RBP:                0 |
| RSI:                0   RDI:              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

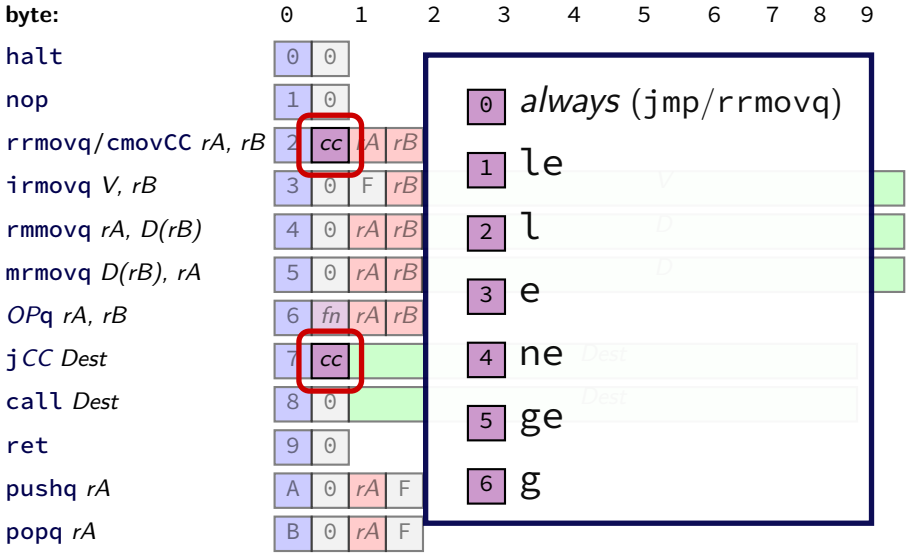
```
$ ./hclrs nop_cpu.hcl nops.yo
+------------------- between cycles    0 and    1 ----------------------+
| RAX:                0   RCX:                0   RDX:                0 |
| RBX:                0   RSP:                0   RBP:                0 |
| RSI:                0   RDI:              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

# backup slides

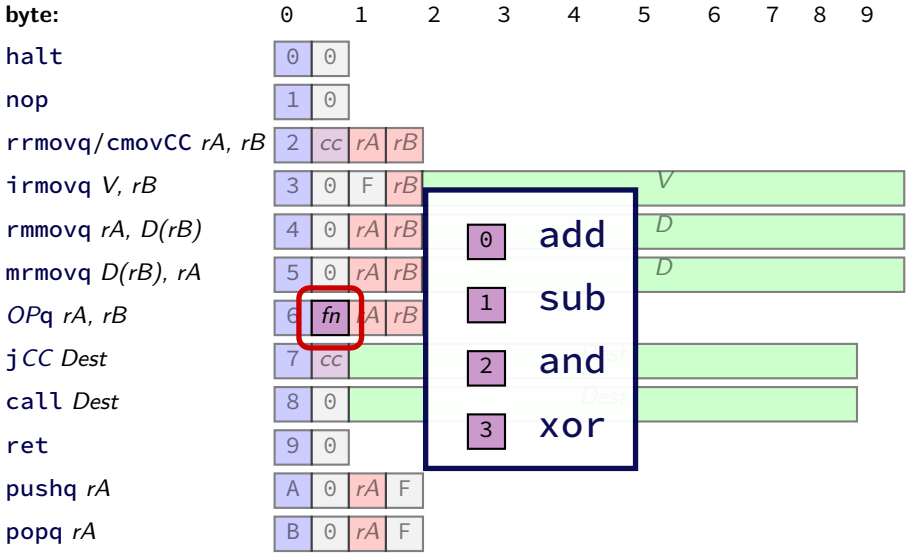
# Y86-64 instruction formats



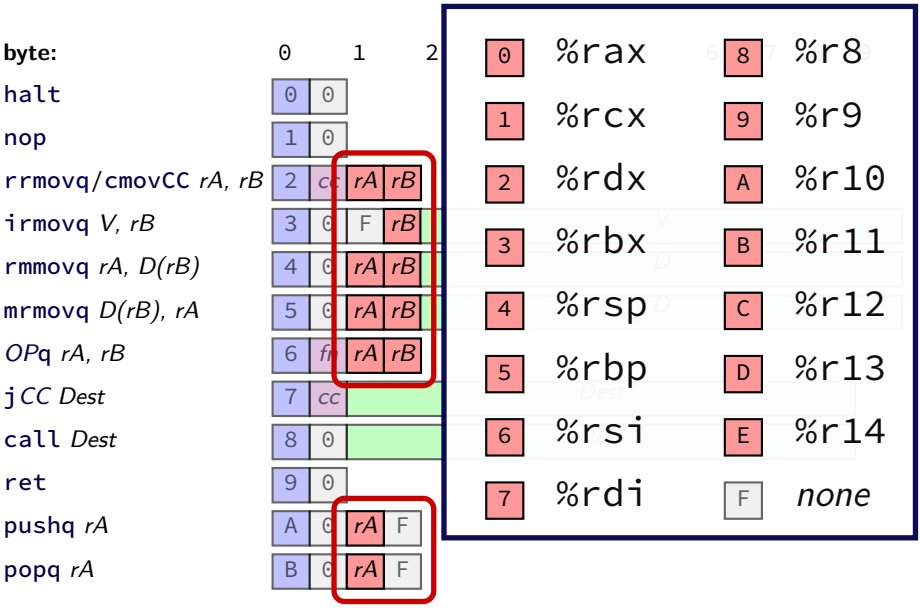
# Secondary opcodes: `cmovcc`/`jcc`



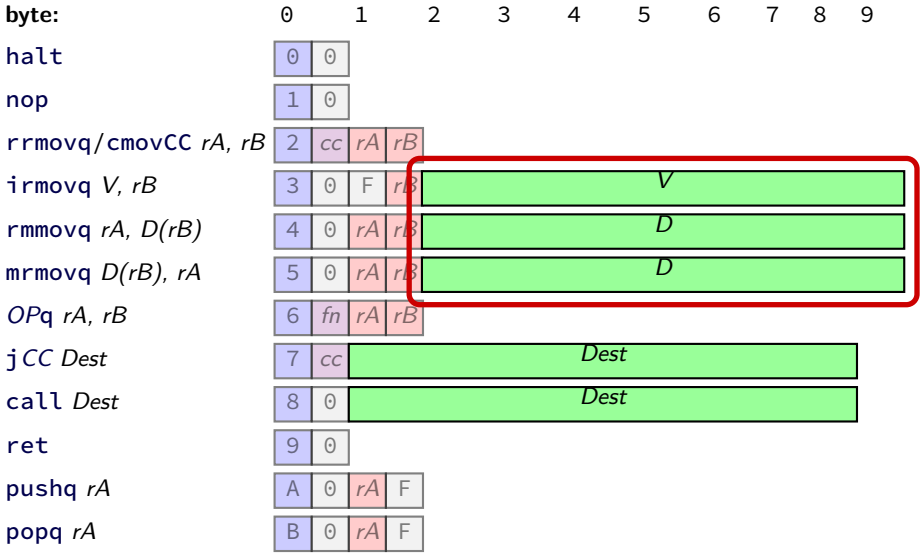
# Secondary opcodes: *OP*q



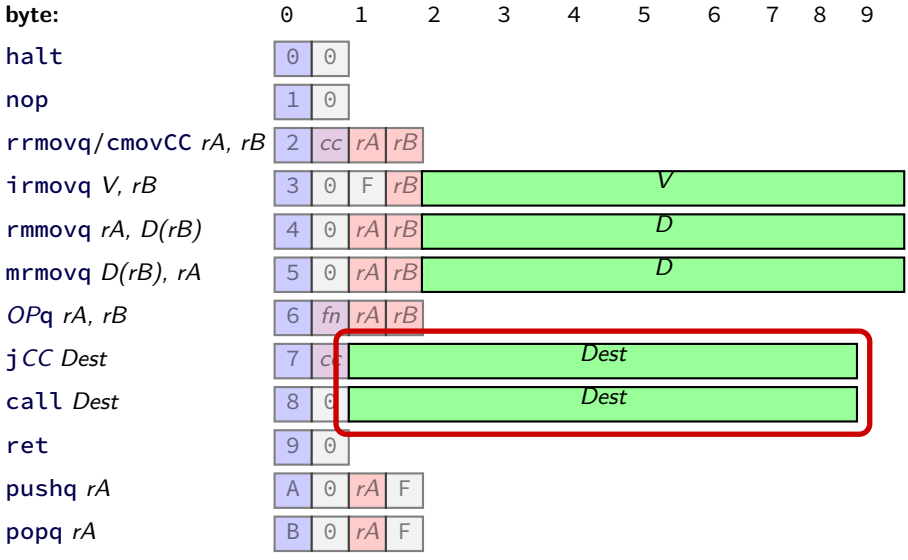
# Registers: *rA, rB*



# Immediates: *V, D, Dest*



# Immediates: *V, D, Dest*



# differences from book

**w**ire not **b**ool or **i**nt

book uses names like `valC` — not required!

- author's environment limited adding new wires

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

implement your own ALU

# differences from book

**w**ire not **b**ool or **i**nt

book uses names like `valC` — not required!

author's environment limited adding new wires

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

implement your own ALU

# differences from book

**w**ire not **b**ool or **i**nt

book uses names like `valC` — not required!

author's environment limited adding new wires

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

implement your own ALU

# 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 `0_name`

fixed functionality

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