HCLs and HCL2D

logistics — exam

there is an exam next Thursday

next Tuesday — finishing up this topic + review

next Wednesday — review with your lab TAs (entirely optional)

selected anonymous feedback (1)

"Could you make the slides available some time in the morning before lecture, so that we can have the option of printing them & taking notes on them in class?"

- mine: **draft** sometimes already posted
- "all lecture notes" link at top of schedule page

on "simpler instructions can take up less space..." question

- e.g. "nop" is less bytes than "mrmovq"
- not RISC-like: variable-length encoding

over other quiz questions being unclear

- need to discuss amongst ourselves

selected anonymous feedback (2)

"It seems that the quizzes were added to the page MUCH later than usual instead of coming up at least one quiz in advance and for whatever reason I thought that you might have given us a bye for one week, didn't realize that it just came late and ended up getting a zero for it. Doubtful you all can do anything about this but please keep uploading the quizzes early."

- we will upload a placeholder "no quiz this week" quiz if there is no quiz for a week
- and yes, try to post early

selected anonymous feedback (3)

"Why do labs and homeworks have nothing to do with what we're learning in lecture? It's incredibly stressful to be handed these assignments when we're not talking about them (nor are we learning anything about C) in lecture."

- can't really give you Y86-64 assignment yet
- mostly delay between lecture and HW
- and/or "things we thought were taught on 2150"
- scheduling constraint: avoided HW before strsep

selected anonymous feedback (4)

on the homework:

"Maybe don't recommend coding in the assignment area because it can "lose contact with the server" and, even though it assures you that your code is saved when the save button is pressed, it loses it when you refresh the page. Nice."

- bug fixed Monday — very sorry about this

selected anonymous feedback (5)

"For the hw, the directions are not clear enough, and the online simulator is miserable to use. I would like to see the test cases that are being run, and the output form that is expected of the functions. It is very hard to understand what to do when we dont know what you are testing the functions for."

- probably should've given refresher on pointers-to-pointers
- still want held-out test cases (no hard-coding answers)
- would like to give AddressSanitizer output (or similar), but logistically hard
- "normal" assignment: no student-accessible autograder

selected anonymous feedback (6)

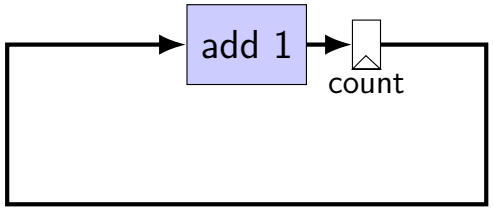
office hour problems late Tuesday

primary problem: TA missed their OH but queue not closed

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

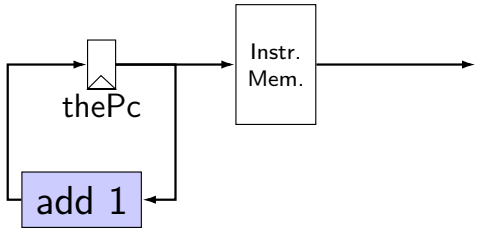
our tool: HCL2D

built for this course

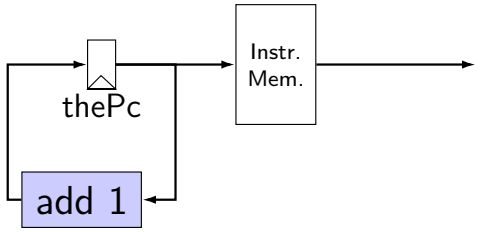
assumes you're making a processor

somewhat different from textbook's HCL

nop CPU

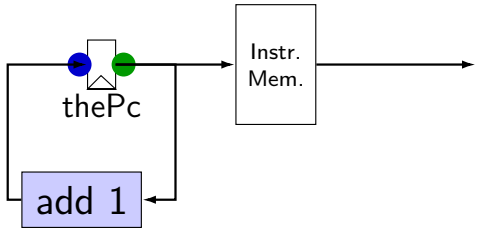


nop CPU



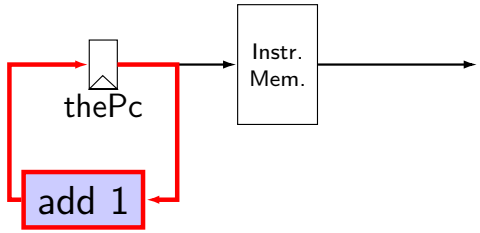

```
register pF {
   thePc : 64 = 0;
}
```


nop CPU



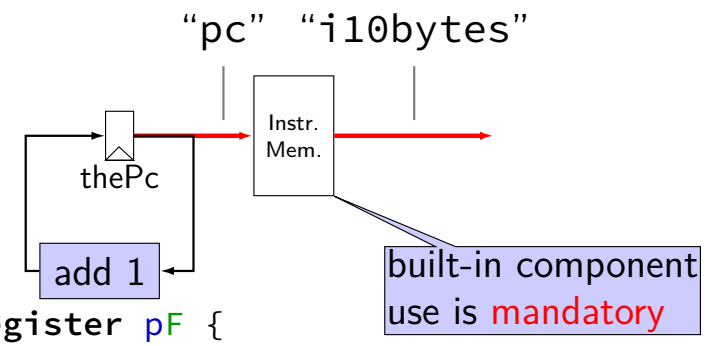

```
register pF {
   thePc : 64 = 0;
}
```


nop CPU



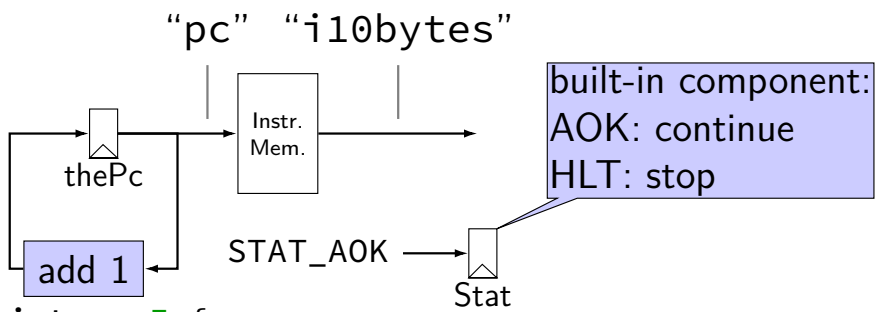

```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
```

nop CPU



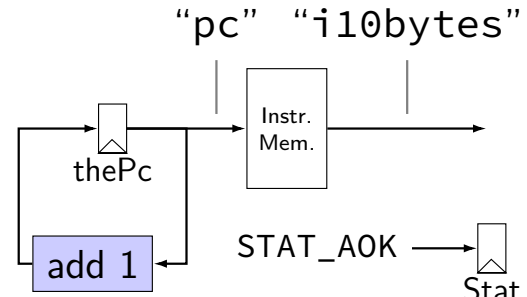

```
register pF {
   thePc : 64 = 0;
}
p_thePc = F_thePc + 1;
pc = F_thePc;
```


nop CPU




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


nop CPU




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


nop CPU: running

need a program in memory

.yo file

`tools/yas` — convert `.ys` to `.yo`

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

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

can build tools by running `make`

nop CPU: creating a program

create assemby file: nops.ys:

```
nop
nop
nop
nop
nop
```

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

nop.yo

more readable/simpler than normal executables:

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

loaded into data and program memory

parts left of | just comments

building a simulator

put HCL2D code in `nop_cpu.hcl`

run `make nop_cpu.exe`

then, `./nop_cpu.exe`:

```
USAGE: ./nop_cpu.exe [options] somefile.yo
Options:
    [a number]       : time out after that many steps (default: 10000)
    -i --interactive : pause every clock cycle
    -q --quiet       : only show final state
    -d --debug       : show every action during simulation
```


running simulator

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

running simulator

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

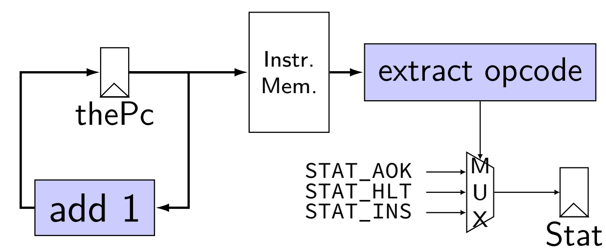

running simulator

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

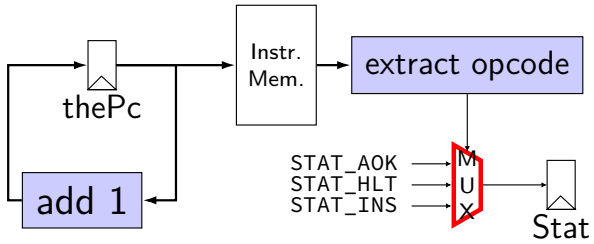

running simulator

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


nop/halt CPU



nop/halt CPU



MUXes in HCL2D

book calls "case expression"

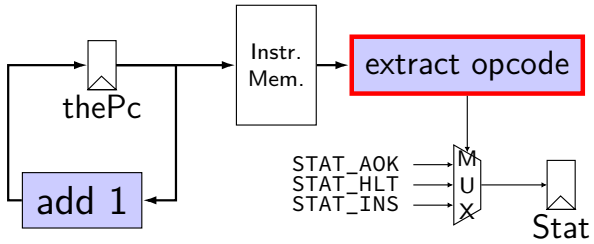
conditions evaluated (as if) in order

first match is output: `result = [`

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

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

nop/halt CPU



subsetting bits in HCL2D

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

least significant bit is bit 0

bit numbers and instructions

value from instruction memory in `i10bytes`

80-bit integer, little-endian order:

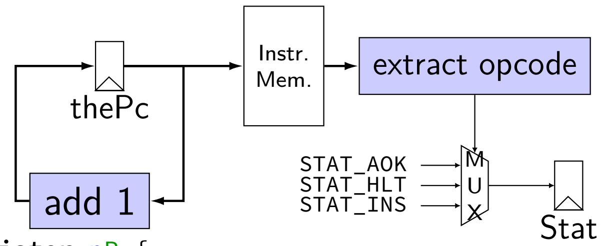
first byte is least significant byte

first bit is least significant bit of first byte

byte 0: (MSB) `opcode` `fn` (LSB)
(byte 1: (MSB) `rA` `rB` (LSB))

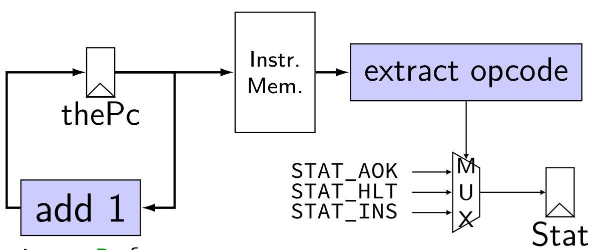
therefore: bits 4–8 (most significant bits of 1st byte)

nop/halt CPU



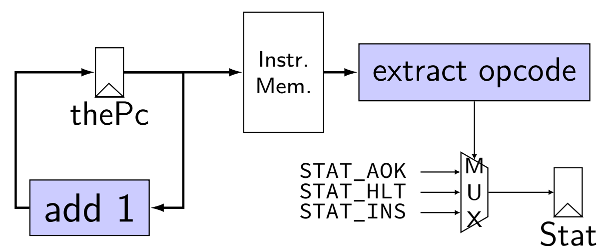

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; (default case)
];
```


nop/halt CPU



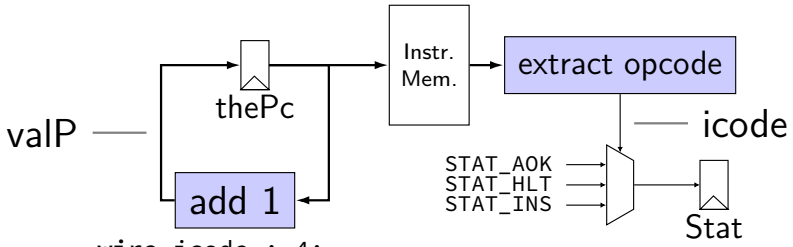

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; (default case)
];
```


nop/halt CPU



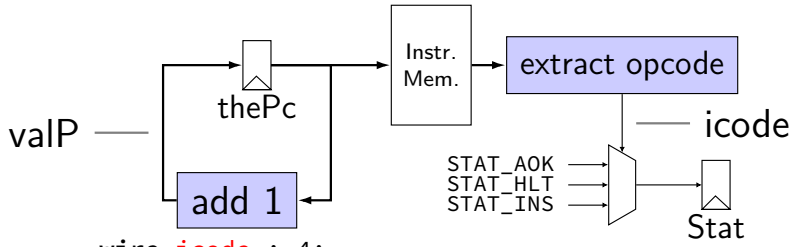

```
register pP {
    thePc : 64 = 0;
}
p_thePc = P_thePc + 1;
pc = P_thePc;
Stat = [
    i10bytes[4..8] == NOP : STAT_AOK;
    i10bytes[4..8] == HALT : STAT_HLT;
    1 : STAT_INS; (default case)
];
```

nop/halt CPU (w/ named wires)



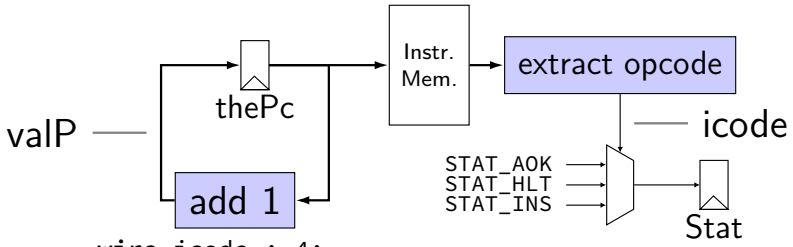

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePc + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```


nop/halt CPU (w/ named wires)



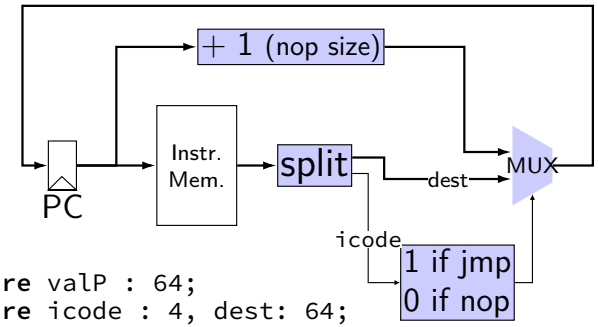

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePc + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```


nop/halt CPU (w/ named wires)




```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePc + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

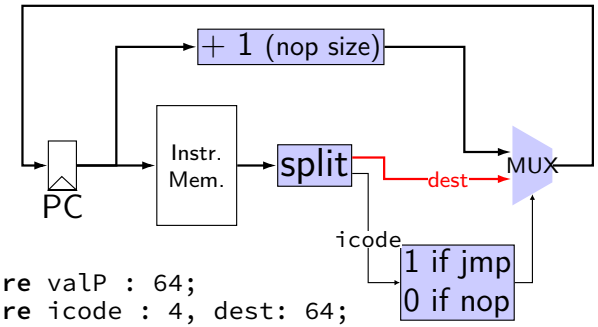

nop/jmp CPU




```
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;
];
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;
];
p_thePc = valP;
pc = P_thePc;
```

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


running nop/jmp/halt

nopjmp.ys:

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

...assemble with `yas`

nopjmp.yo

nopjmp.yo:

```
0x000: 10                       |      nop
0x001: 7013000000000000000      |      jmp C
0x00a: 701c00000000000000       | B:   jmp D
0x013: 700a00000000000000       | C:   jmp B
0x01c: 10                       | D:   nop
0x01d: 10                       |      nop
0x01e: 00                       |      halt
```


nopjmp.yo

nopjmp.yo:

```
0x000: 10                       |      nop
0x001: 7013000000000000000      |      jmp C
0x00a: 701c00000000000000       | B:   jmp D
0x013: 700a00000000000000       | C:   jmp B
0x01c: 10                       | D:   nop
0x01d: 10                       |      nop
0x01e: 00                       |      halt
```

running nopjmp.yo

```
$ ./nopjmp_cpu.exe nopjmp.yo
...
...
+--------------------- (end of halted state) ---------------------------+
Cycles run: 7
Time used: 238
```


comparing to yis

```
$ ./nopjmp_cpu.exe nopjmp.yo
...
...
+--------------------- (end of halted state) ---------------------------+
Cycles run: 7
Time used: 238

$ ./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:
```


debugging mode

```
$ ./nopjmp_cpu.exe -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       |
+-----------------------------------------------------------------------+
set pc to 0x0
pc = 0x0; loaded [10 : nop]
set icode to 0x1
set valP to 0x1
set p_thePc to 0x1
set Stat to 0x1
.------------------- between cycles    1 and    2 ----------------------+
...
```


debugging mode

```
$ ./nopjmp_cpu.exe -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       |
+-----------------------------------------------------------------------+
set pc to 0x0
pc = 0x0; loaded [10 : nop]
set icode to 0x1
set valP to 0x1
set p_thePc to 0x1
set Stat to 0x1
.------------------- between cycles    1 and    2 ----------------------+
...
```

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)
set pc to 0x0
pc = 0x0; loaded [10 : nop]
set icode to 0x1
set valP to 0x1
set p_thePc to 0x1
set Stat to 0x1
+------------------- between cycles    1 and    2 ----------------------+
...
```


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)
set pc to 0x0
pc = 0x0; loaded [10 : nop]
set icode to 0x1
set valP to 0x1
set p_thePc to 0x1
set Stat to 0x1
+------------------- between cycles    1 and    2 ----------------------+
...
```


quiet mode

```
$ ./nopjmp_cpu.exe -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
Time used: 238
```


things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

register banks

```
register xY {
   foo : width1 = defaultValue1;
   bar : width2 = defaultValue2;
}
```

two letters: input (X) / Output (Y)

- input signals: `x_foo`, `x_bar`
- output signals: `Y_foo`, `Y_bar`

each value has width in bits

each value has initial value — *mandatory*

some other signals — `stall`, `bubble`

- later in semester

register banks

```
register xY {
   foo : width1 = defaultValue1;
   bar : width2 = defaultValue2;
}
```

two letters: input (X) / Output (Y)

- input signals: `x_foo`, `x_bar`
- output signals: `Y_foo`, `Y_bar`

each value has width in bits

each value has initial value — *mandatory*

some other signals — `stall`, `bubble`

- later in semester

register banks

```
register xY {
   foo : width1 = defaultValue1;
   bar : width2 = defaultValue2;
}
```

two letters: input (X) / Output (Y)

- input signals: `x_foo`, `x_bar`
- output signals: `Y_foo`, `Y_bar`

each value has width in bits

each value has initial value — *mandatory*

some other signals — `stall`, `bubble`

- later in semester

things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

wires

```
wire wireName : wireWidth;
wireName = ...;
... = wireName;
... = wireName;
```

things that can accept/produce a signal

- some created implicitly – e.g. by creating register
- some builtin — supplied components (like instruction memory)

assignment — connecting wires

wires and order

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
p_thePc = valP;
pc = P_thePc;
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
valP = P_thePC + 1;
icode = i10bytes[4..8];
```

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePC + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```


wires and order

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
p_thePc = valP;
pc = P_thePc;
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
valP = P_thePC + 1;
icode = i10bytes[4..8];
```

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePC + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

wires and order

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
p_thePc = valP;
pc = P_thePc;
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
valP = P_thePC + 1;
icode = i10bytes[4..8];
```

```
wire icode : 4;
wire valP : 64;
register pP {
    thePc : 64 = 0;
}
valP = P_thePC + 1;
p_thePc = valP;
pc = P_thePc;
icode = i10bytes[4..8];
Stat = [
    icode == NOP : STAT_AOK;
    icode == HALT : STAT_HLT;
    1 : STAT_INS;
];
```

order doesn't matter

wire is connected or not connected

wires and width

```
wire bigValueOne: 64;
wire bigValueTwo: 64;
wire smallValue: 32;
bigValueOne = smallValue; /* ERROR */
smallValue = bigValueTwo; /* ERROR */
…
wire bigValueOne: 64;
wire bigValueTwo: 64;
wire smallValue: 32;

smallValue = bigValueTwo[0..32]; /* OKAY */
```


things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

Stat register

how do we stop the machine?

hard-wired mechanism — `Stat` register

possible values:

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

must be set

determines if simulator keeps going

things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

program memory

input wire: `pc`

output wire: `i10bytes`

- 80-bits wide (10 bytes)
- bit 0 — least significant bit of first byte
- (width of largest instruction)

program memory

input wire: `pc`

output wire: `i10bytes`

- 80-bits wide (10 bytes)
- bit 0 — least significant bit of first byte
- (width of largest instruction)

what about less than 10 byte instructions?

- just don't use the extra bits

things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

register file

four register number inputs (4-bit):

- sources: `reg_srcA`, `reg_srcB`
- destinations: `reg_dstM` `reg_dstE`

no write or no read? register number `0xF` (`REG_NONE`)

two register value inputs (64-bit):

- `reg_inputE`, `reg_inputM`

two register output values (64-bit):

- `reg_outputA`, `reg_outputB`

example using register file: add CPU

```
wire rA : 4, rB : 4, icode : 4, ifunc: 4;
register pP {
    thePC : 64 = 0;
}
/* PC update: */
pc = P_thePC; p_thePC = P_thePC + 2;
/* Decode: */
icode = i10bytes[4..8]; ifunc = i10bytes[0..4];
rA = i10bytes[12..16]; rB = i10bytes[8..12];
reg_srcA = rA;
reg_srcB = rB;
/* Execute + Writeback: */
reg_inputE = reg_outputA + reg_outputB;
reg_dstE = rB;
/* Status maintainence: */
Stat = ...
```


example using register file: add CPU

```
wire rA : 4, rB : 4, icode : 4, ifunc: 4;
register pP {
    thePC : 64 = 0;
}
/* PC update: */
pc = P_thePC; p_thePC = P_thePC + 2;
/* Decode: */
icode = i10bytes[4..8]; ifunc = i10bytes[0..4];
rA = i10bytes[12..16]; rB = i10bytes[8..12];
reg_srcA = rA;
reg_srcB = rB;
/* Execute + Writeback: */
reg_inputE = reg_outputA + reg_outputB;
reg_dstE = rB;
/* Status maintainence: */
Stat = ...
```

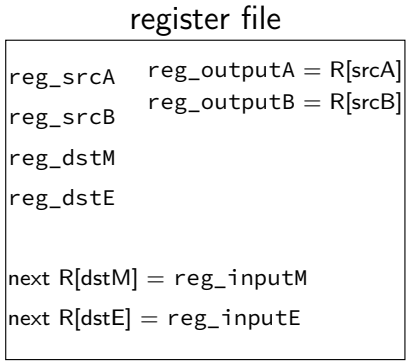
47

example using register file: add CPU

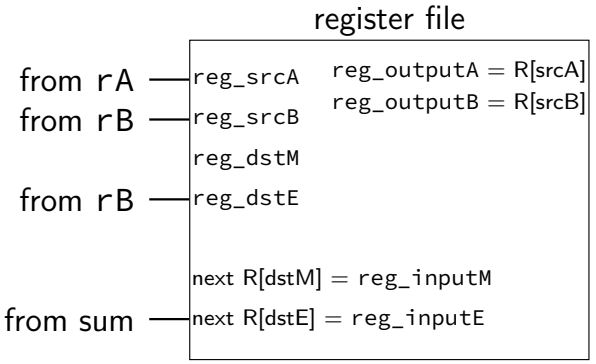
```
wire rA : 4, rB : 4, icode : 4, ifunc: 4;
register pP {
    thePC : 64 = 0;
}
/* PC update: */
pc = P_thePC; p_thePC = P_thePC + 2;
/* Decode: */
icode = i10bytes[4..8]; ifunc = i10bytes[0..4];
rA = i10bytes[12..16]; rB = i10bytes[8..12];
reg_srcA = rA;
reg_srcB = rB;
/* Execute + Writeback: */
reg_inputE = reg_outputA + reg_outputB;
reg_dstE = rB;
/* Status maintainence: */
Stat = ...
```

47

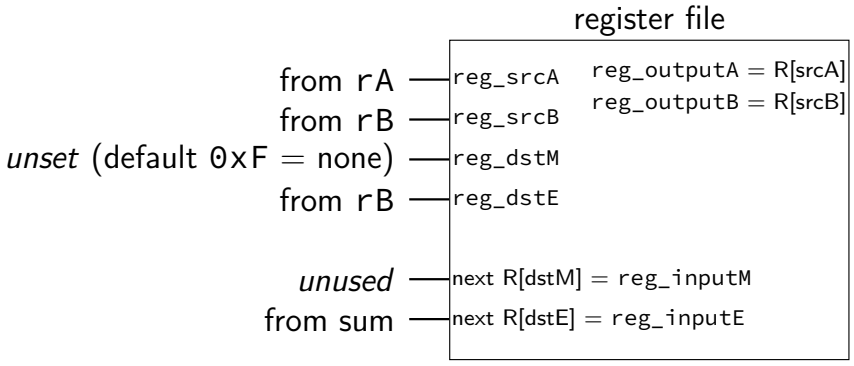
register file picture



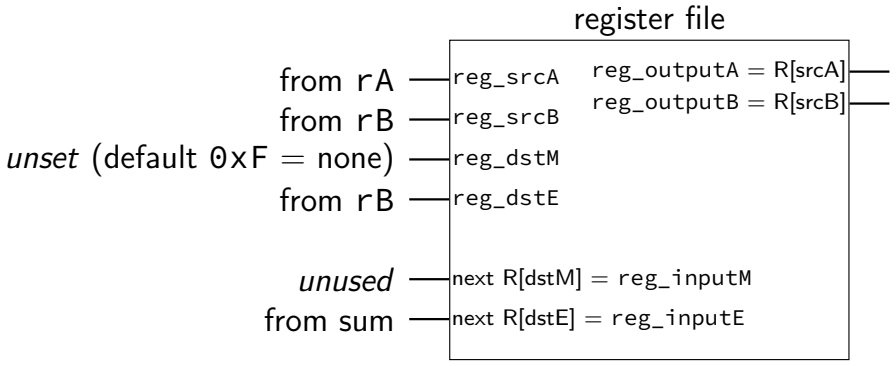
register file picture



register file picture



register file picture



things in HCL2D

register banks

wires

things for our processor:

- `Stat` register
- instruction memory
- the register file
- data memory

data memory

input address: `mem_addr`

input value: `mem_input`

output value: `mem_output`

read/write enable: `mem_readbit`, `mem_writebit`

reading from data memory

```
mem_addr = 0x12345678;
mem_readbit = 1;
mem_writebit = 0;
... = mem_output;
```

`mem_output` has value in same cycle

reading from data memory

```
mem_addr = 0x12345678;
mem_readbit = 1;
mem_writebit = 0;
... = mem_output;
```

`mem_output` has value in same cycle

reading from data memory

```
mem_addr = 0x12345678;
mem_readbit = 1;
mem_writebit = 0;
... = mem_output;
```

mem_output has value in same cycle

writing to data memory

```
mem_addr = 0x12345678;
mem_input = ...;
mem_readbit = 0;
mem_writebit = 1;
```

memory updated for next cycle

writing to data memory

```
mem_addr = 0x12345678;
mem_input = ...;
mem_readbit = 0;
mem_writebit = 1;
```

memory updated for next cycle

writing to data memory

```
mem_addr = 0x12345678;
mem_input = ...;
mem_readbit = 0;
mem_writebit = 1;
```

memory updated for next cycle

differences from book

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

book uses names like `valC` — not required!

- author's environment limited adding new wires

implement your own ALU

differences from book

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

book uses names like `valC` — not required!

- author's environment limited adding new wires

implement your own ALU

exercise: implementing ALU?

```
wire aluOp : 2,
     aluValueA : 64,
     aluValueB : 64,
     aluResult : 64;
const ALU_ADD = 0b00,
      ALU_SUB = 0b01,
      ALU_AND = 0b10,
      ALU_XOR = 0b11;
aluResult = [
    aluOp == ALU_ADD : aluValueA + aluValueB;
    aluOp == ALU_SUB : aluValueA - aluValueB;
    aluOp == ALU_AND : aluValueA & aluValueB;
    aluOp == ALU_XOR : aluValueA ^ aluValueB
];
```


on design choices

textbook choices:

- memory always goes to 'M' port of register file
- RSP +/- 8 uses normal ALU, not seperate adders
- ...

do you have to do this? no

you: single cycle/instruction; use supplied register/memory

other logic: make it function correctly

HCL2D summary

declare/assign values to wires

MUXes with

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

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)