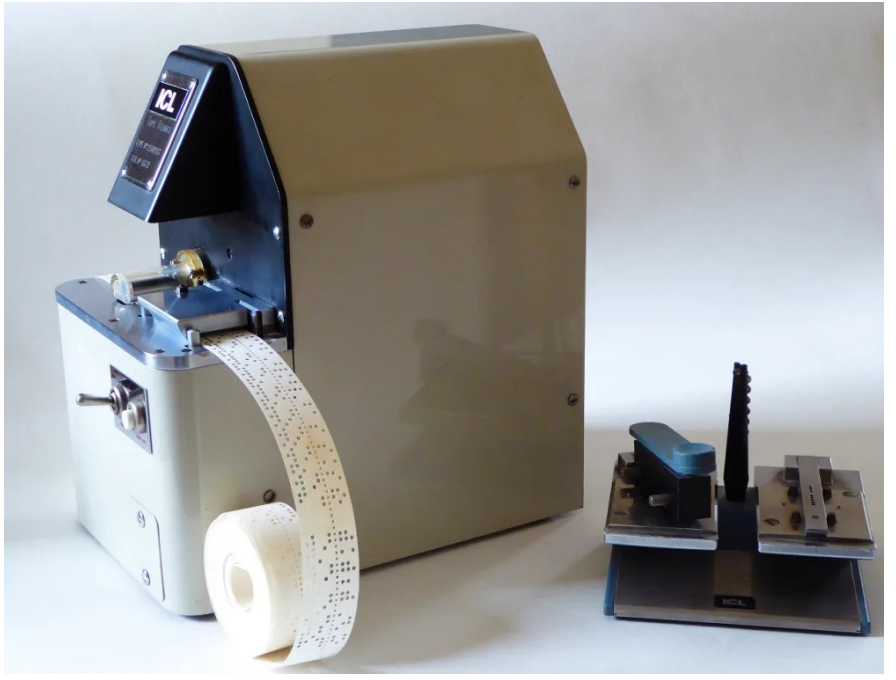
Lab Review and Questions

- >> << | & ^ ?

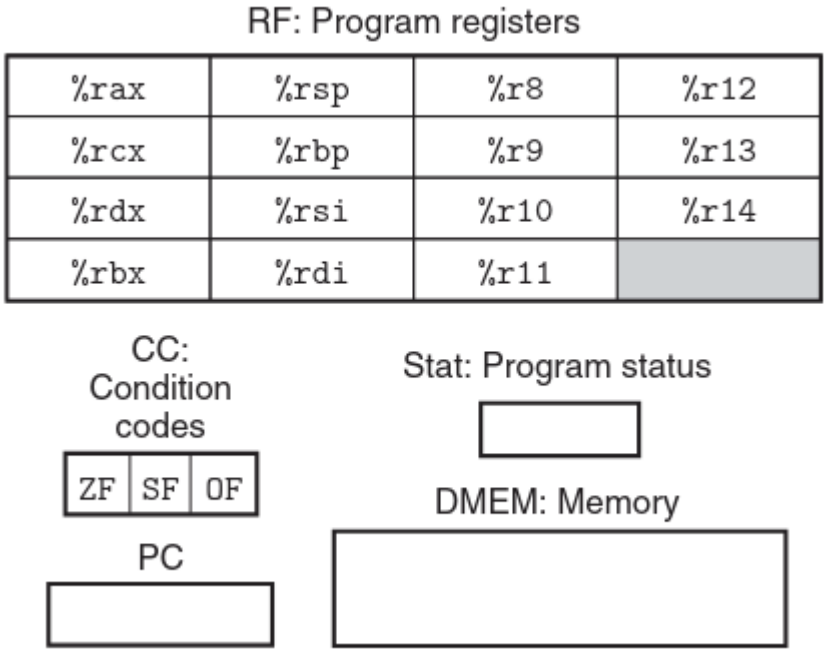
Processor Architecture

- Y86-64 is NOT real! ;-)
- Serialized
  - Tutorial arch for class



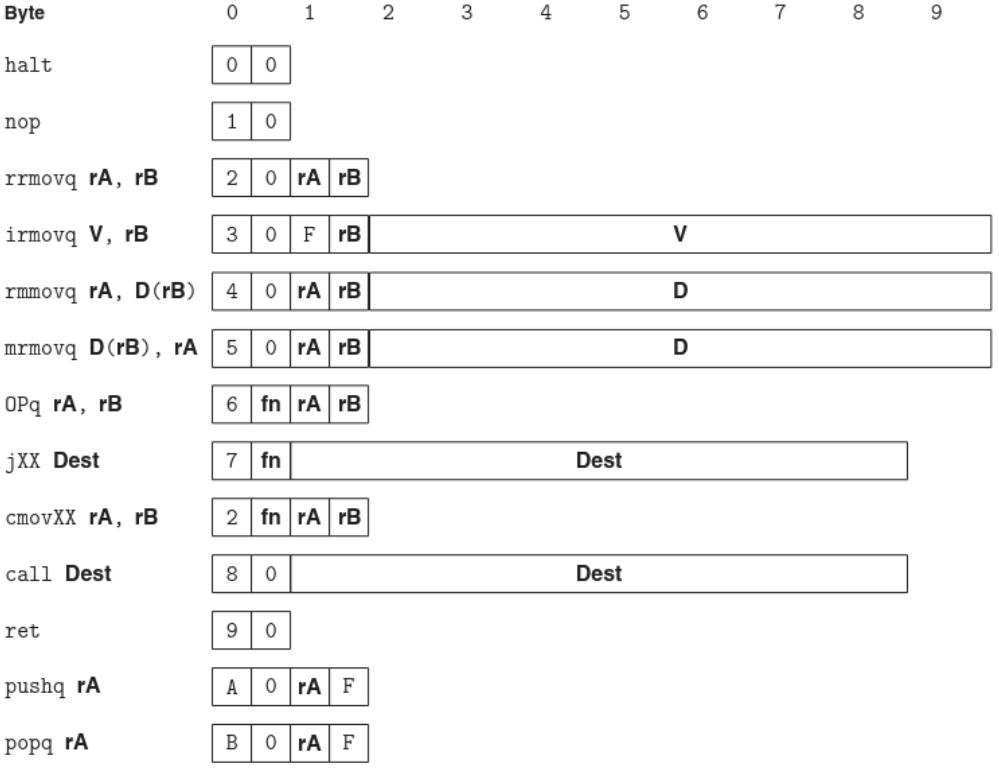
Processor Architecture

- Virtualized state of the system



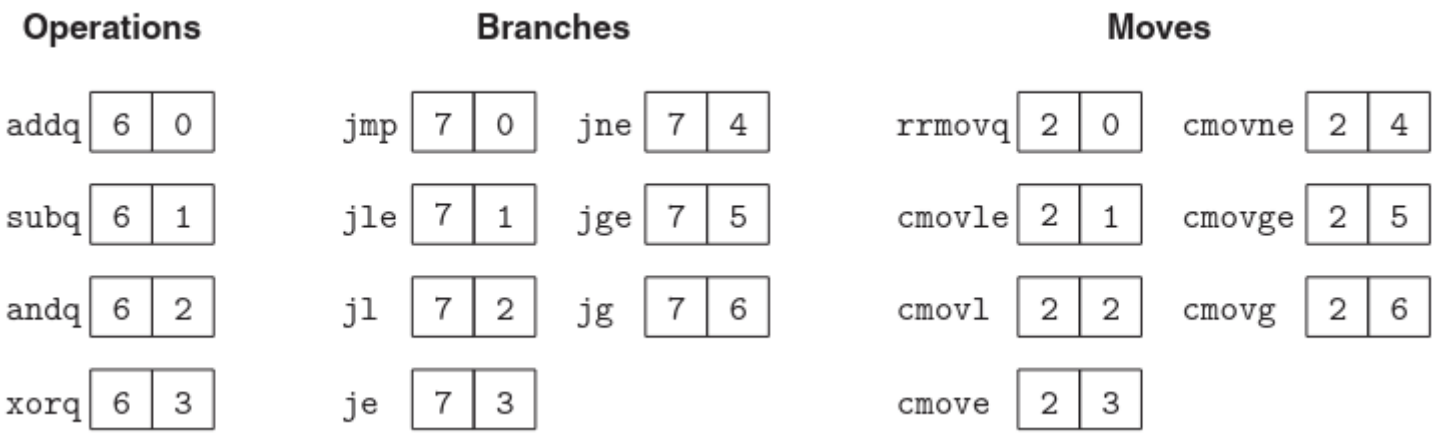
Processor Architecture

- Simplified Instruction Set



Processor Architecture

- Simplified Instruction Set



Processor Architecture

- Simplified Register Layout

Number	Register name	Number	Register name
0	%rax	8	%r8
1	%rcx	9	%r9
2	%rdx	A	%r10
3	%rbx	B	%r11
4	%rsp	C	%r12
5	%rbp	D	%r13
6	%rsi	E	%r14
7	%rdi	F	No register

Y86-64 ASM

- Different Implementations

```
1   long sum(long *start, long count)
2   {
3       long sum = 0;
4       while (count) {
5           sum += *start;
6           start++;
7           count--;
8       }
9       return sum;
10  }
```

Y86-64 ASM

- Different Implementations

x86-64 code

```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     movl    $0, %eax          sum = 0
3     jmp     .L2               Goto test
4   .L3:                      loop:
5     addq    (%rdi), %rax      Add *start to sum
6     addq    $8, %rdi          start++
7     subq    $1, %rsi          count--
8   .L2:                      test:
9     testq   %rsi, %rsi        Test sum
10    jne     .L3               If !=0, goto loop
11    rep; ret                  Return
```

```
1   long sum(long *start, long count)
2   {
3       long sum = 0;
4       while (count) {
5           sum += *start;
6           start++;
7           count--;
8       }
9       return sum;
10  }
```

Y86-64 ASM

- Different Implementations

Y86-64 code

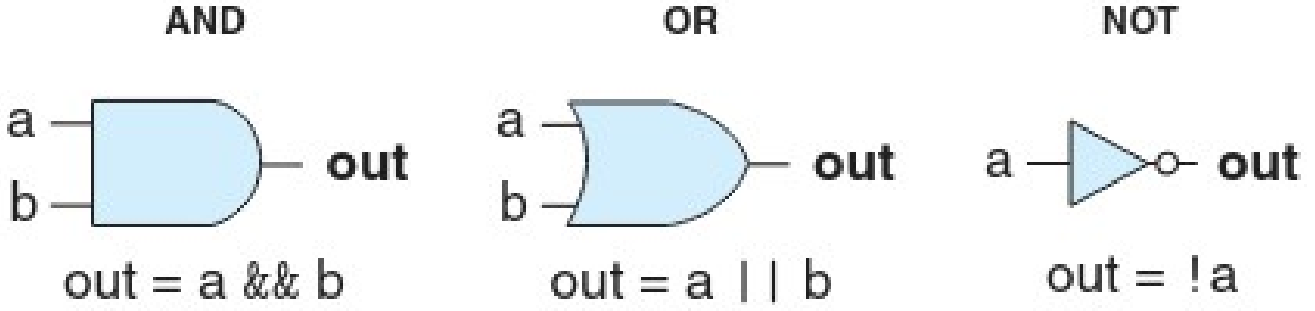
```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     irmovq $8,%r8          Constant 8
3     irmovq $1,%r9          Constant 1
4     xorq %rax,%rax         sum = 0
5     andq %rsi,%rsi         Set CC
6     jmp     test           Goto test
7   loop:
8     mrmovq (%rdi),%r10     Get *start
9     addq %r10,%rax         Add to sum
10    addq %r8,%rdi          start++
11    subq %r9,%rsi          count--.  Set CC
12  test:
13    jne    loop            Stop when 0
14    ret                    Return
```

x86-64 code

```
    long sum(long *start, long count)
    start in %rdi, count in %rsi
1   sum:
2     movl    $0, %eax         sum = 0
3     jmp     .L2              Goto test
4   .L3:                     loop:
5     addq    (%rdi), %rax     Add *start to sum
6     addq    $8, %rdi         start++
7     subq    $1, %rsi         count--
8   .L2:                     test:
9     testq   %rsi, %rsi       Test sum
10    jne     .L3              If !=0, goto loop
11    rep; ret                 Return
```

HCL

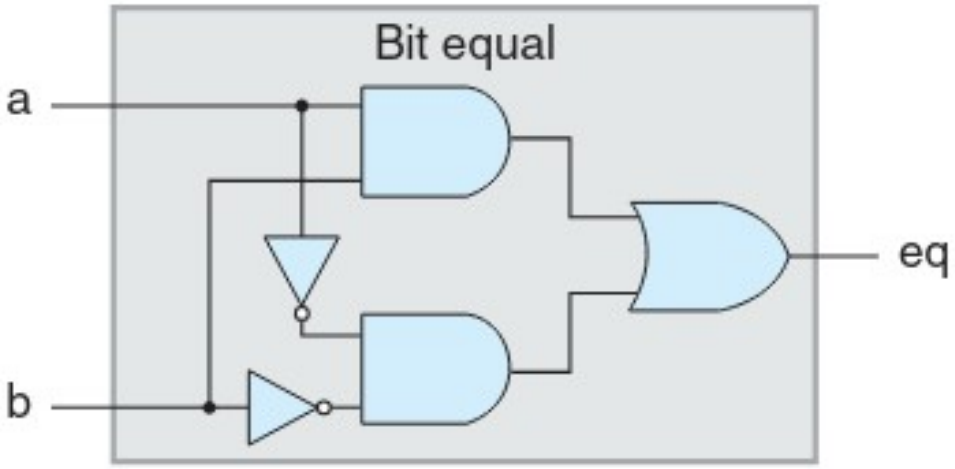
- Logical Translation of Circuits



HCL

- Boolean Logic Reappears
- Truth Table?

0	0	?
0	1	?
1	0	?
1	1	?

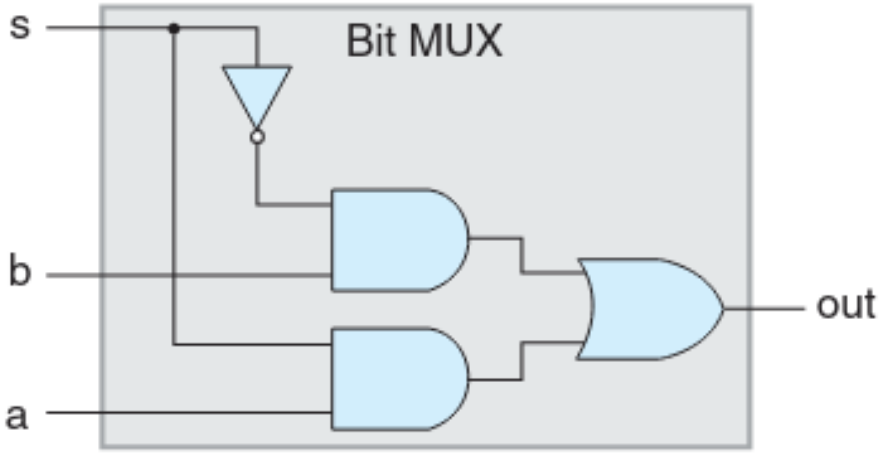


HCL Rules of Circuits

- Inputs can connect to initial input or output of another device.
- Outputs of two or more devices cannot be wired together.
- No connections can form a loop, acyclic graphs only please.

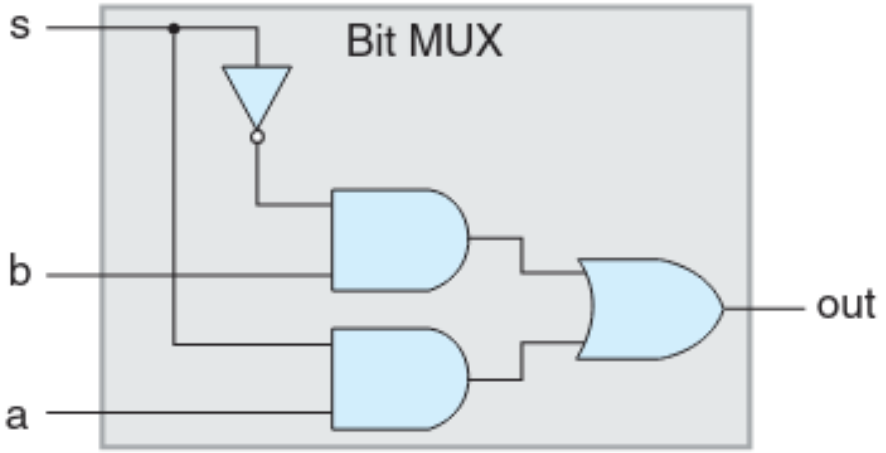
HCL: Multiplexor

- Multiplexor (MUX)
- Sample a/b depend on s
- Usually requires a clock
- out = (s && a) || (!s && b)



HCL: Multiplexor

- Multiplexor (MUX)
- Sample a/b depend on s
- Usually requires a clock
- out = (s && a) || (!s && b)



C vs HCL

- C statements are evaluated once
  - Think of the stream of instructions in a tape reader
- HCL is evaluated each clock cycle
  - Continuous
- Changing input will cause HCL to change output
  - C will not change unless re-excuted.

Makefiles

- GNUMake
  - Let’s hack together
  - Targets and dependencies
  - Clean and Phony