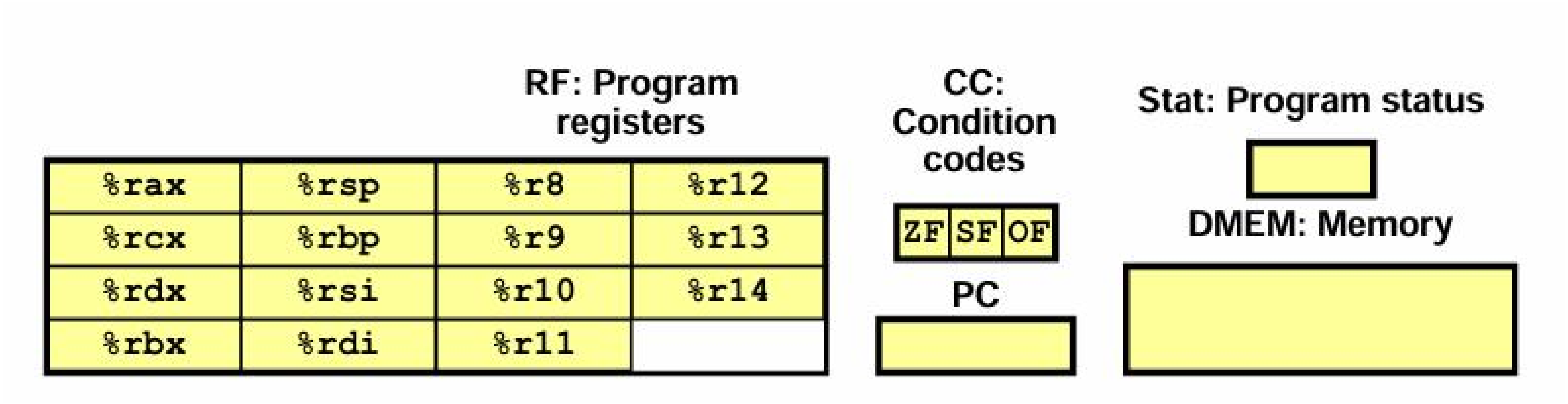
# 4.1-4.2:
# Instruction-Set Architecture & Logic design

周百川　2024.10.16

## **01** Instruction-Set Architecture(ISA)

Y86-64:
Similar to x86-64 but simpler;
Between CISC and RISC

# Programmer-Visible State



Registers:15 registers,no **`%r15`**

Conditional Codes:**`Zero/Negative/Overflow`**

Stat:**`AOK/HLT/ADR/INS`**

Memory:a big array

# Instructions

- **`halt`**: 0 0,suspend instruction execution and set Stat to HLT
- **`nop`**: 1 0,no operation

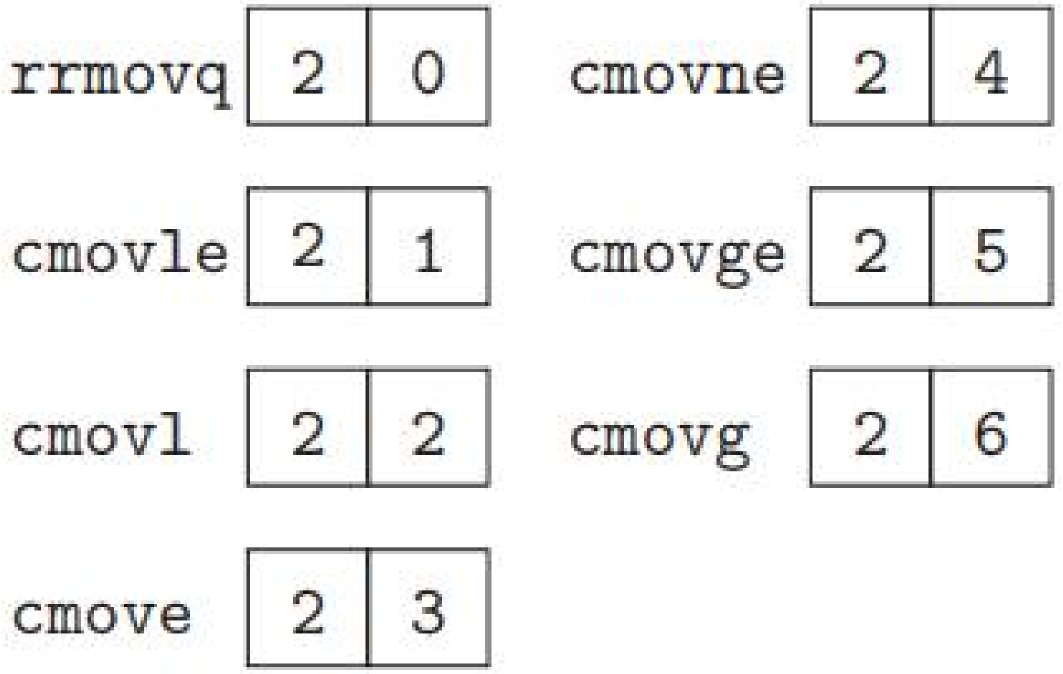
halt
0
0
nop
1
0

# Instructions

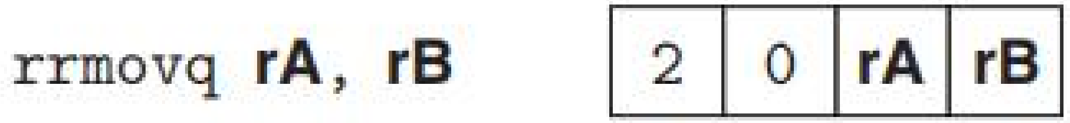
- Moves

  Code:2 X(0-6) rA rB

**`rrmovq`** is a ‘unconditional move’



| | | | | | |
|---|---|---|---|---|---|
| rrmovq | 2 | 0 | cmovne | 2 | 4 |
| cmovle | 2 | 1 | cmovge | 2 | 5 |
| cmovl | 2 | 2 | cmovg | 2 | 6 |
| cmove | 2 | 3 | | | |



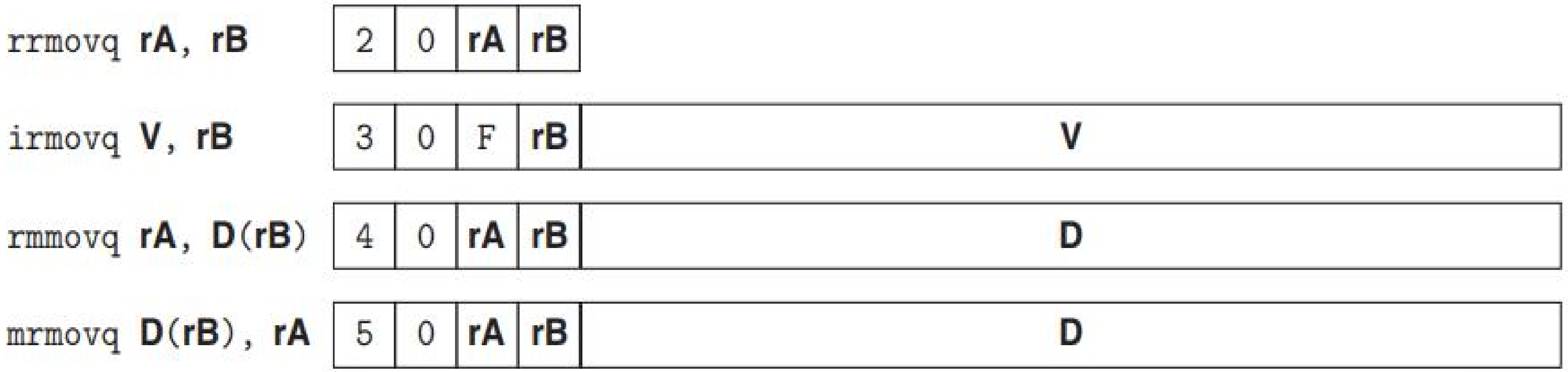
| | | | | |
|---|---|---|---|---|
| rrmovq **rA**, **rB** | 2 | 0 | **rA** | **rB** |

| | | | | |
|---|---|---|---|---|
| cmovXX **rA**, **rB** | 2 | **fn** | **rA** | **rB** |

# Instructions

- Moves

Code:X 0 rA rB (V/D)(8bytes)



register ID:

**0xF** means no register should be accessed

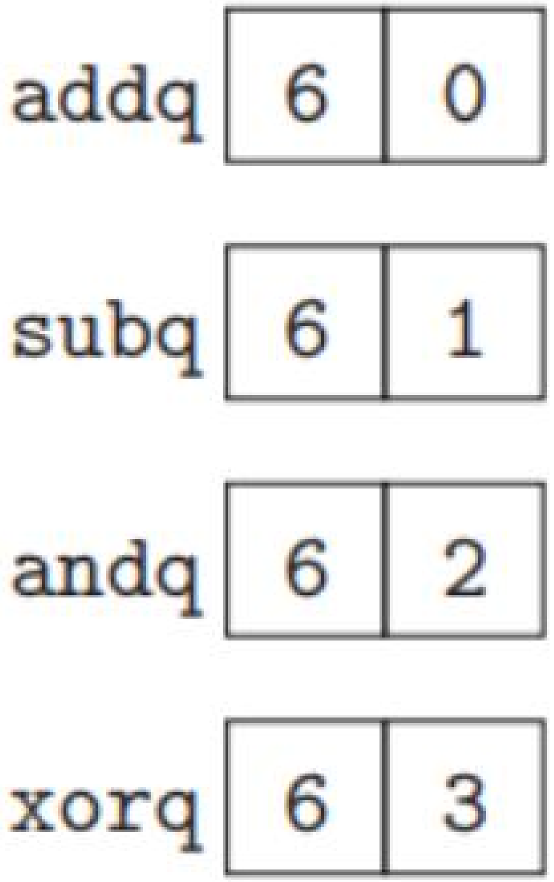
| Number | Register name | Number | Register name |
|---|---|---|---|
| 0 | %rax | 8 | %r8 |
| 1 | %rcx | 9 | %r9 |
| 2 | %rdx | A | %r10 |
| 3 | %rbx | B | %r11 |
| 4 | %rsp | C | %r12 |
| 5 | %rbp | D | %r13 |
| 6 | %rsi | E | %r14 |
| 7 | %rdi | F | No register |

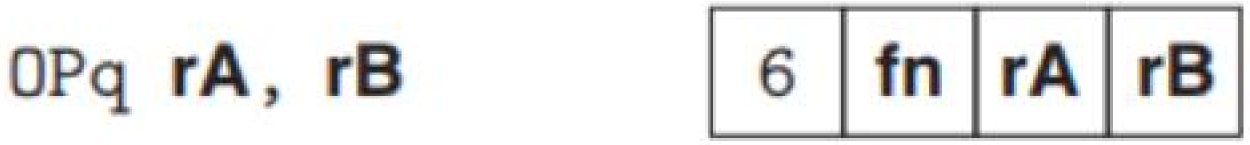
## Instructions

- Operations

Code:6 X(0-3) rA rB

*can only operate registers

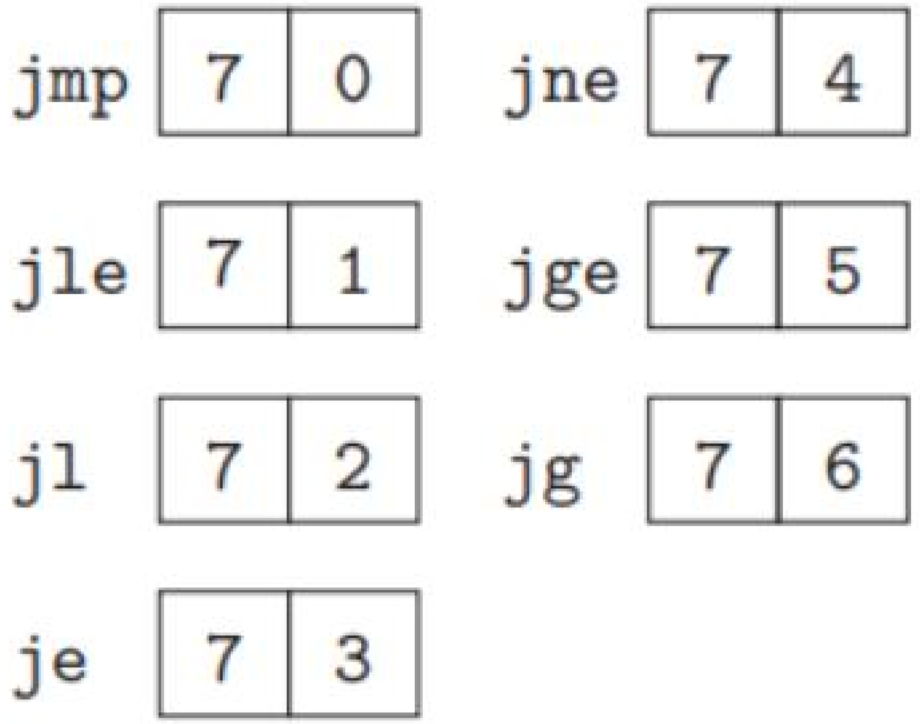




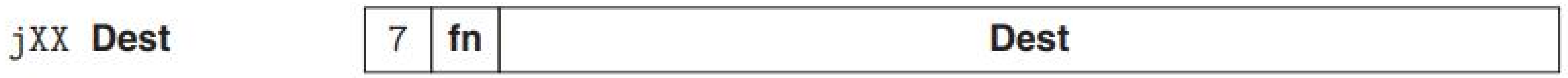
## Instructions

- Branches

Code:7 X(0-6) Dest(8 bytes)



| | | | | | |
|---|---|---|---|---|---|
| jmp | 7 | 0 | jne | 7 | 4 |
| jle | 7 | 1 | jge | 7 | 5 |
| jl | 7 | 2 | jg | 7 | 6 |
| je | 7 | 3 | | | |



| | | | |
|---|---|---|---|
| jXX **Dest** | 7 | **fn** | **Dest** |

## Instructions

**`call`** and **`ret`**,

**`pushq`** and **`popq`**

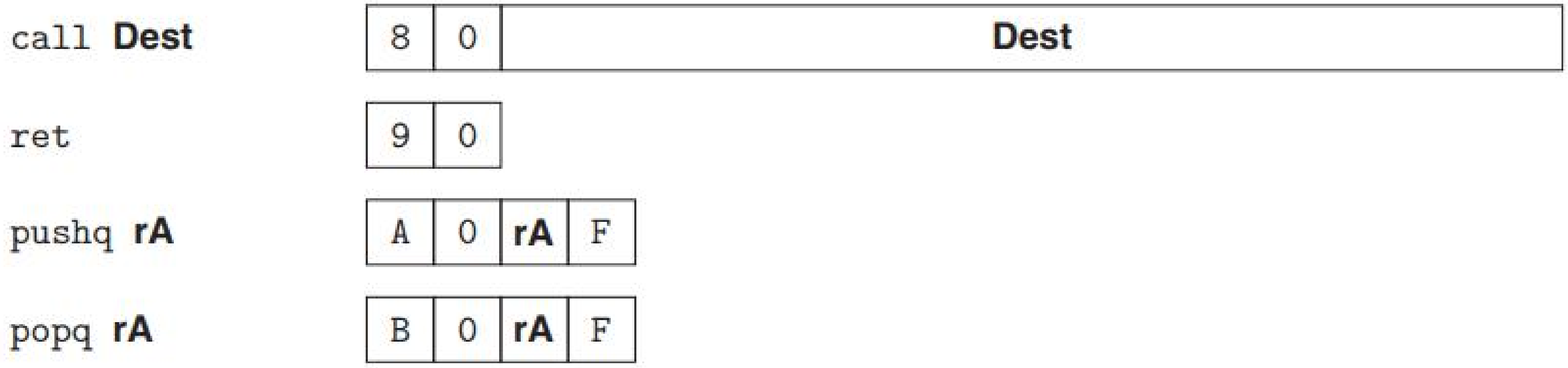
Code:

**`call`**:8 0 Dest(8 bytes)

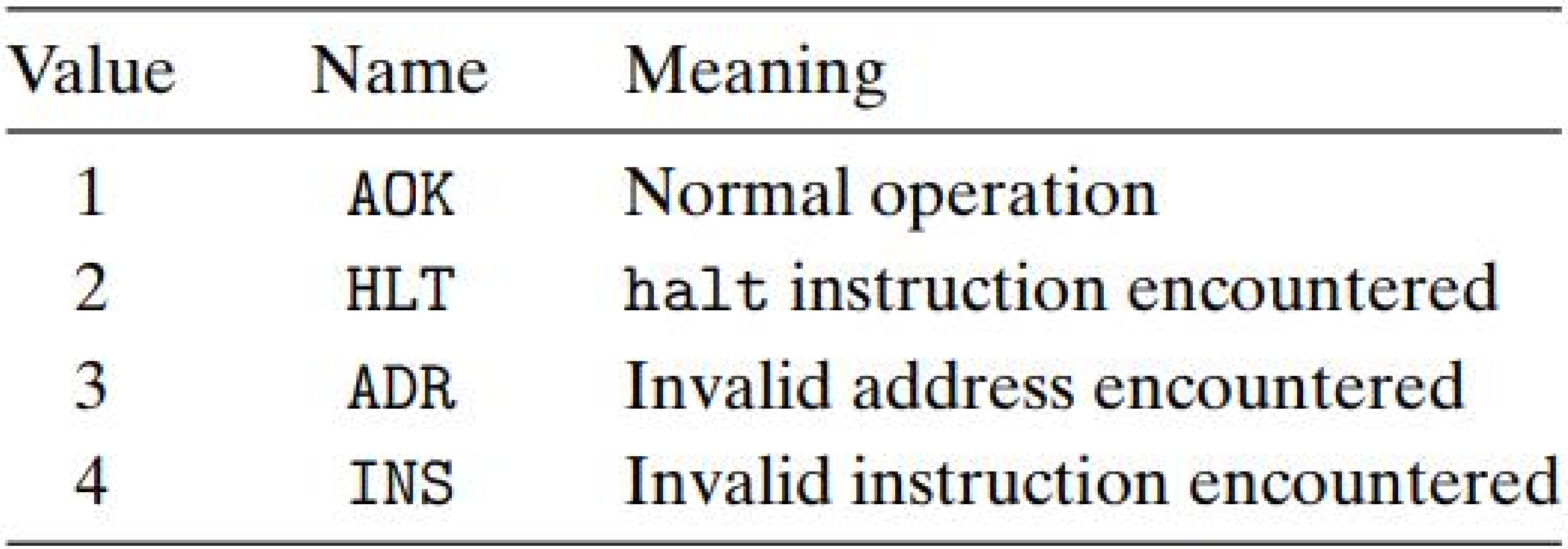
**`ret`**:9 0

**`pushq`**:A 0 rA F

**`popq`**:B 0 rA F



# Exceptions



| Value | Name | Meaning |
|---|---|---|
| 1 | `AOK` | Normal operation |
| 2 | `HLT` | `halt` instruction encountered |
| 3 | `ADR` | Invalid address encountered |
| 4 | `INS` | Invalid instruction encountered |

## **02** Logic Design

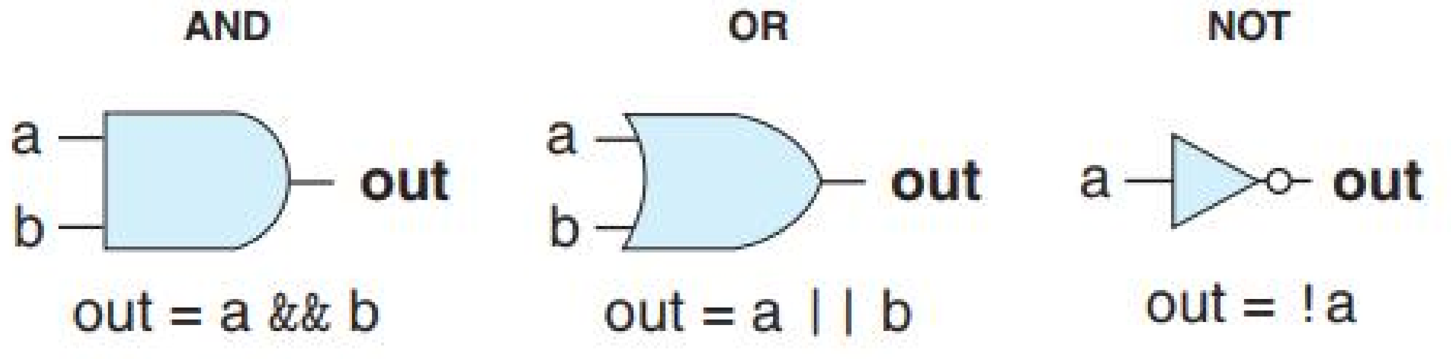
HCL:Hardware Control Language
Can be translate into Verilog and further a working microprocessor

## Logic Gates

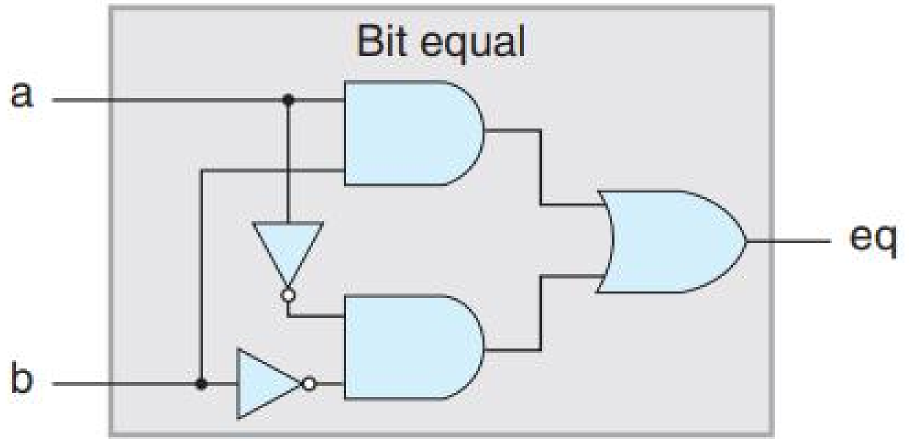
AND: `&&`

OR: `||`

NOT: `!`

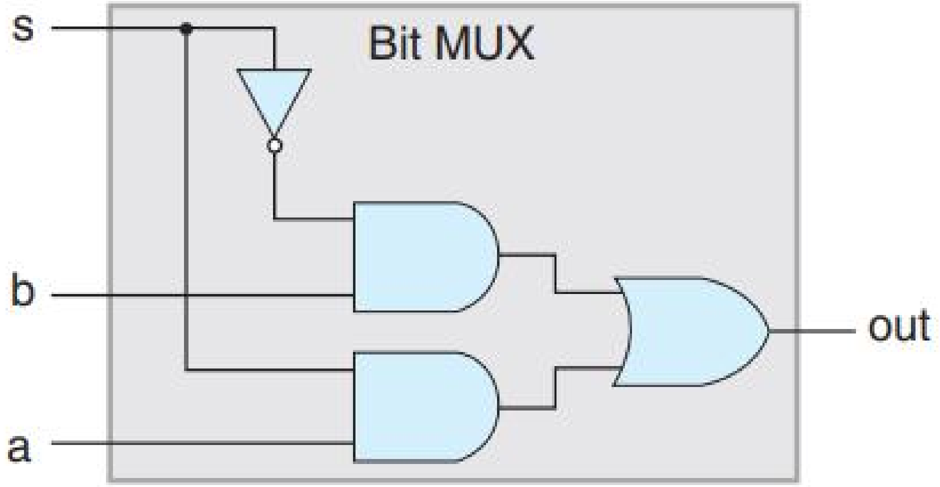


# Combinational Circuits



bit equal

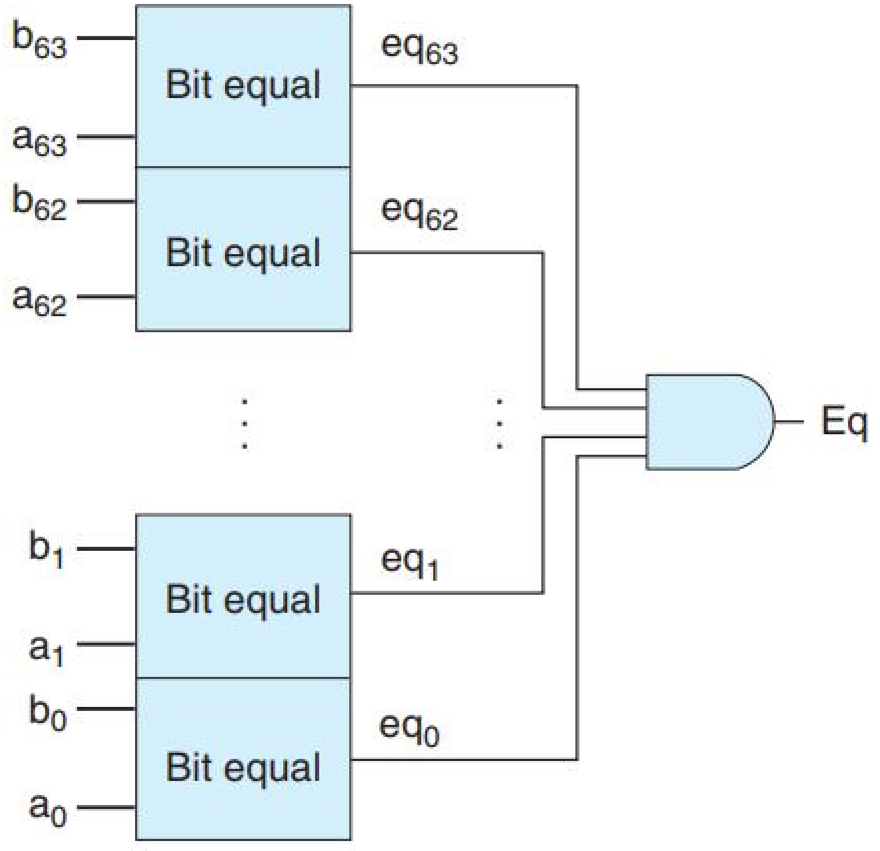
```
bool eq = (a && b) || (!a && !b)
```



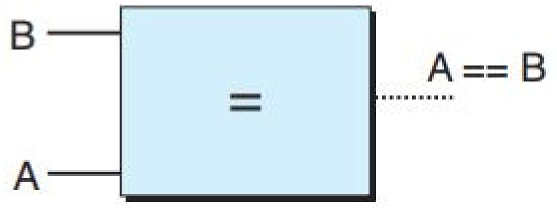
bit MUX

```
bool out = (s && a) || (!s && b)
```

# Combinational Circuits



(a) Bit-level implementation

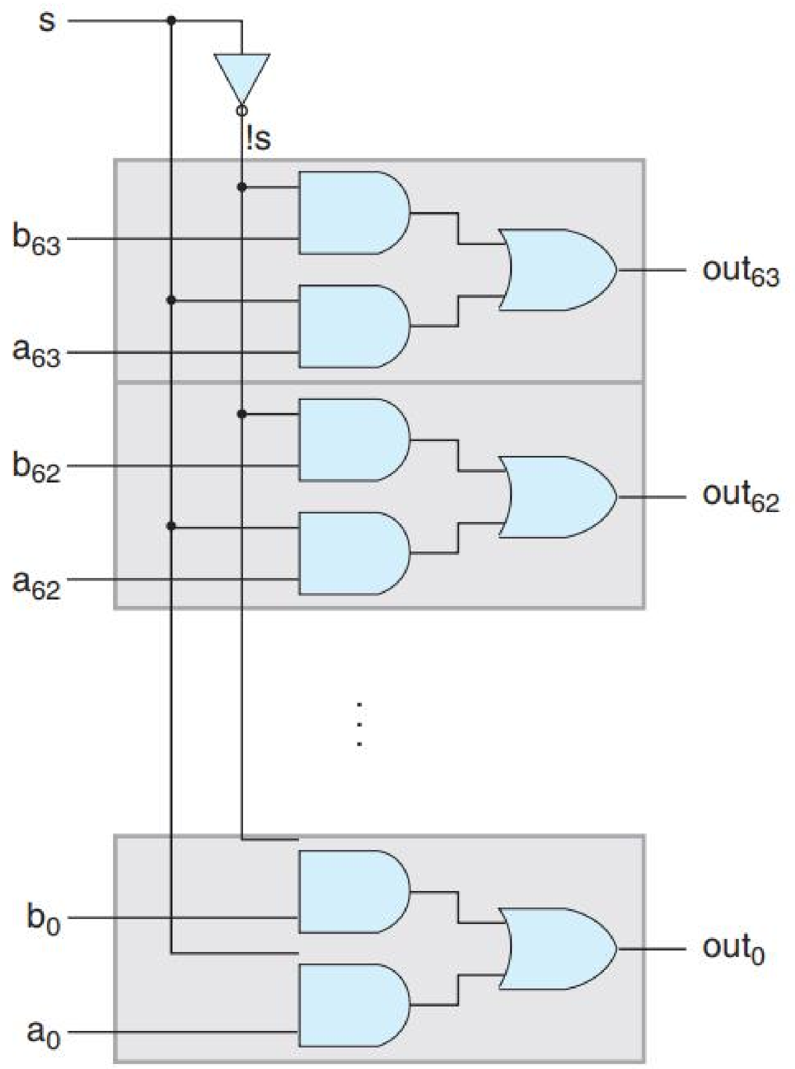


(b) Word-level abstraction

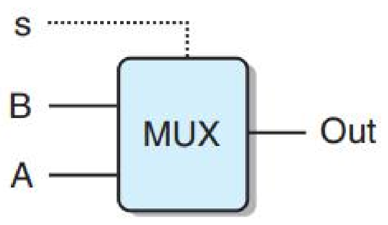
word equal

```
bool Eq = (A == B)
```

## Combinational Circuits



(a) Bit-level implementation



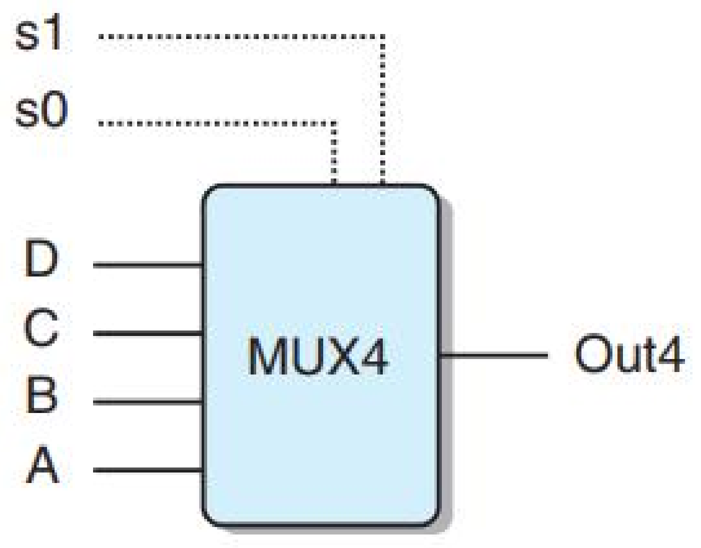
```
int Out = [
  s : A;
  1 : B;
];
```

(b) Word-level abstraction

word MUX

```
word Out = [
     s: A;
     1: B;
];
```

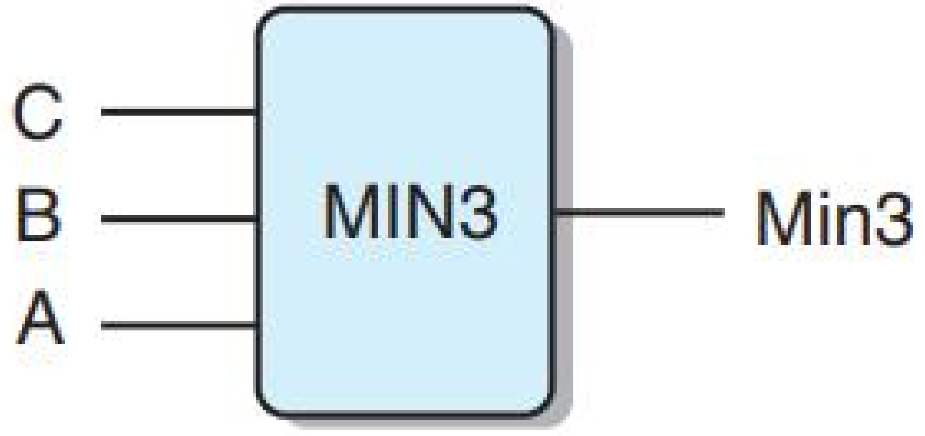
# Combinational Circuits



Four-way MUX

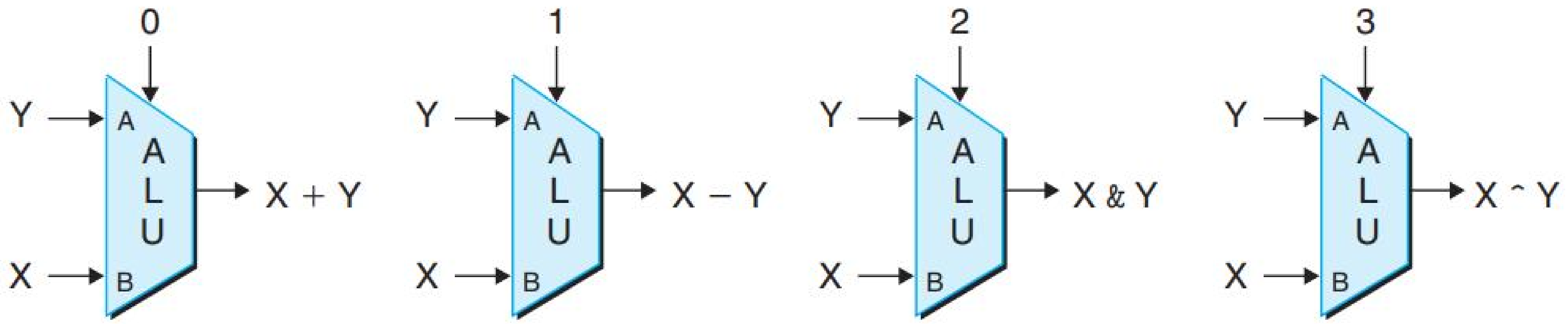
```
word Out4 = [
        !s1 && !s0 : A; # 00
        !s1        : B; # 01
        !s0        : C; # 10
        1          : D; # 11
];
```

## Combinational Circuits

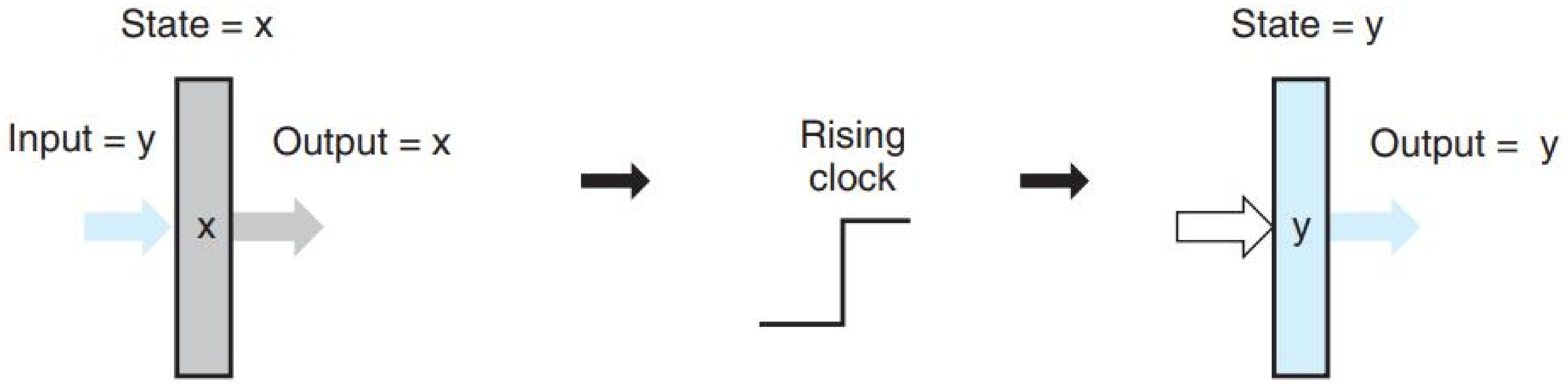


```
word Min3 = [
        A <= B && A <= C : A;
        B <= A && B <= C : B;
        1                : C;
];
```

# Arithmetic/logic unit (ALU)

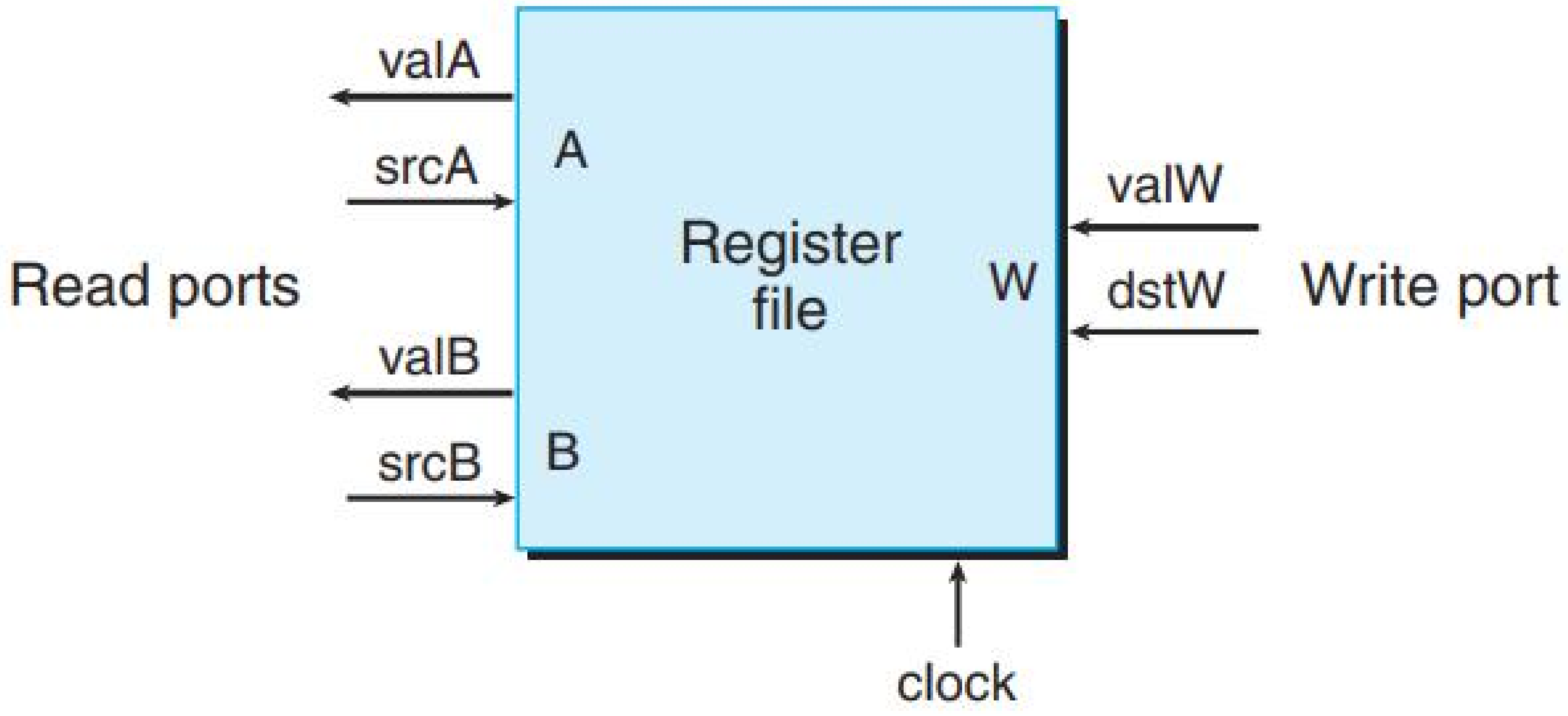


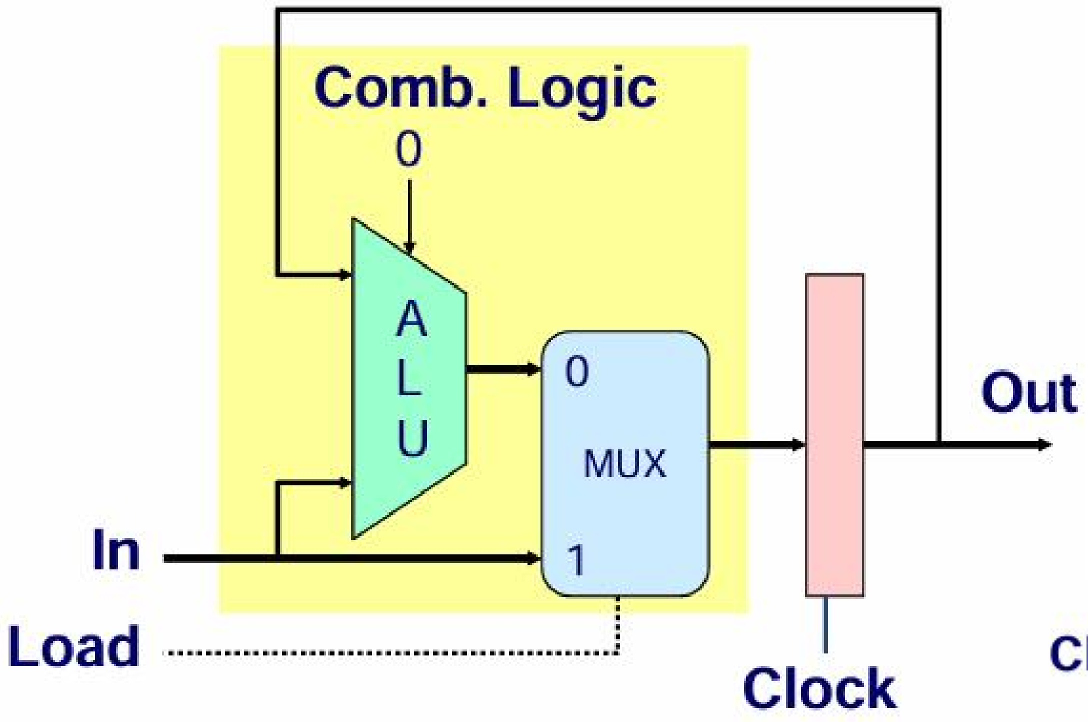
# Register



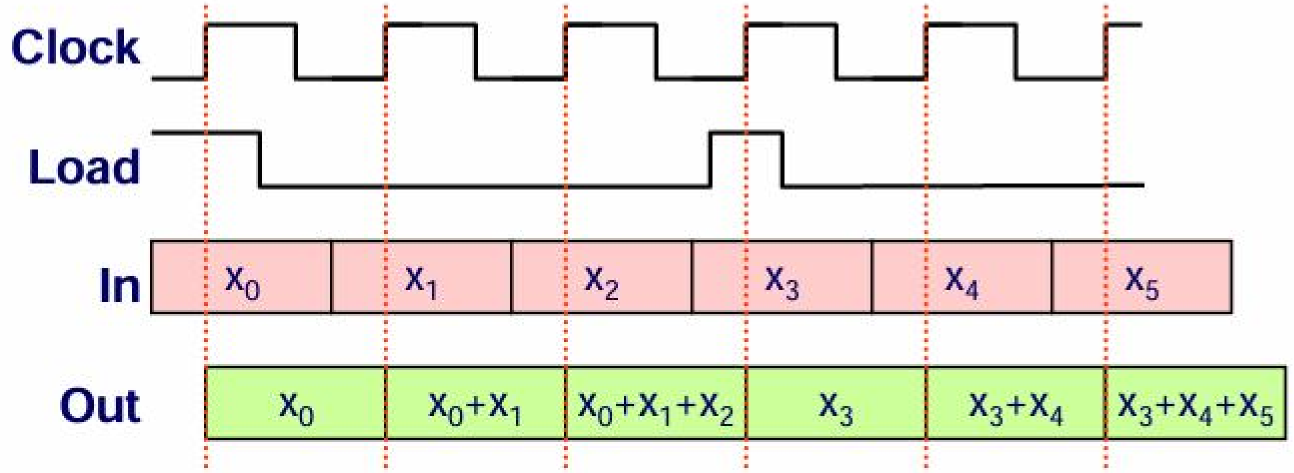
hardware registers != program registers

## Register file





- **Accumulator circuit**
- **Load or accumulate on each cycle**


Clock
Load
In
$x_0$
$x_1$
$x_2$
$x_3$
$x_4$
$x_5$
Out
$x_0$
$x_0+x_1$
$x_0+x_1+x_2$
$x_3$
$x_3+x_4$
$x_3+x_4+x_5$