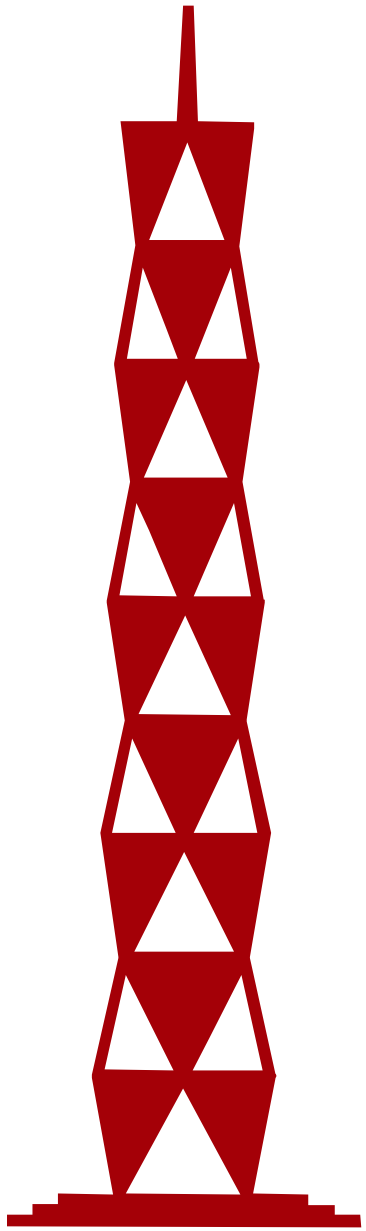


SI100B Introduction to Information Science and Technology (Part 3: Electrical Engineering)

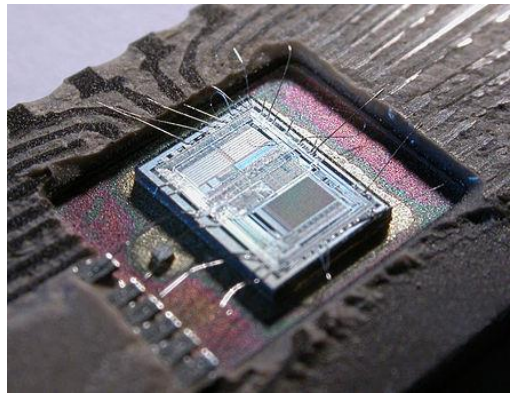
Lecture #5 Digital Building Blocks & Computer Organization

Instructor: Junrui Liang (梁俊睿)

Nov. 25th, 2020

The Theme Story

Devices (1)



Circuits (3/4)

Systems ~~(2)~~ (1)

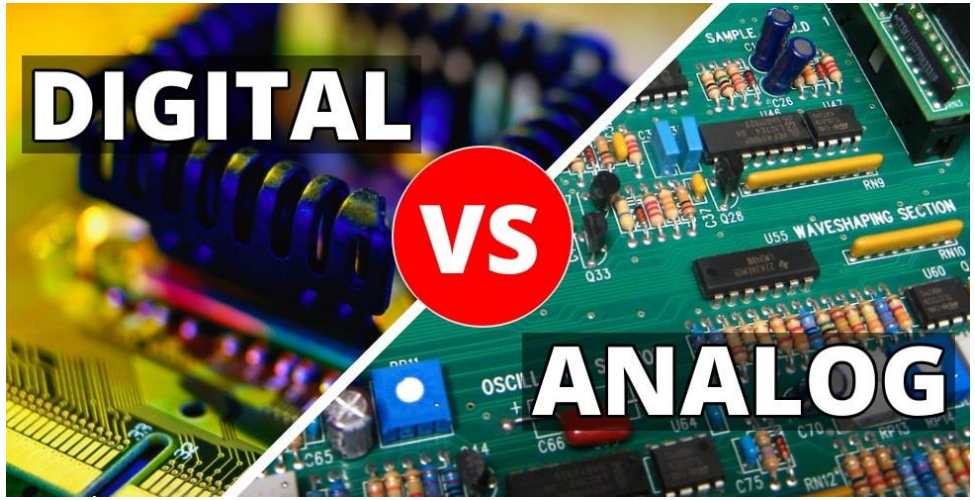
(Pictures are from the Internet)

Study Purpose of Lecture #5

- 哲学（bao'an）三问
  - Who are you?
  - Where are you from?
  - Where are you going?

To answer those questions throughout your life

- In this lecture, we ask
  - What are the fundamental digital building blocks 数字组成模块?
  - How we can build a computer by using those building blocks?
  - How does a computer run under software instructions 软件指令?



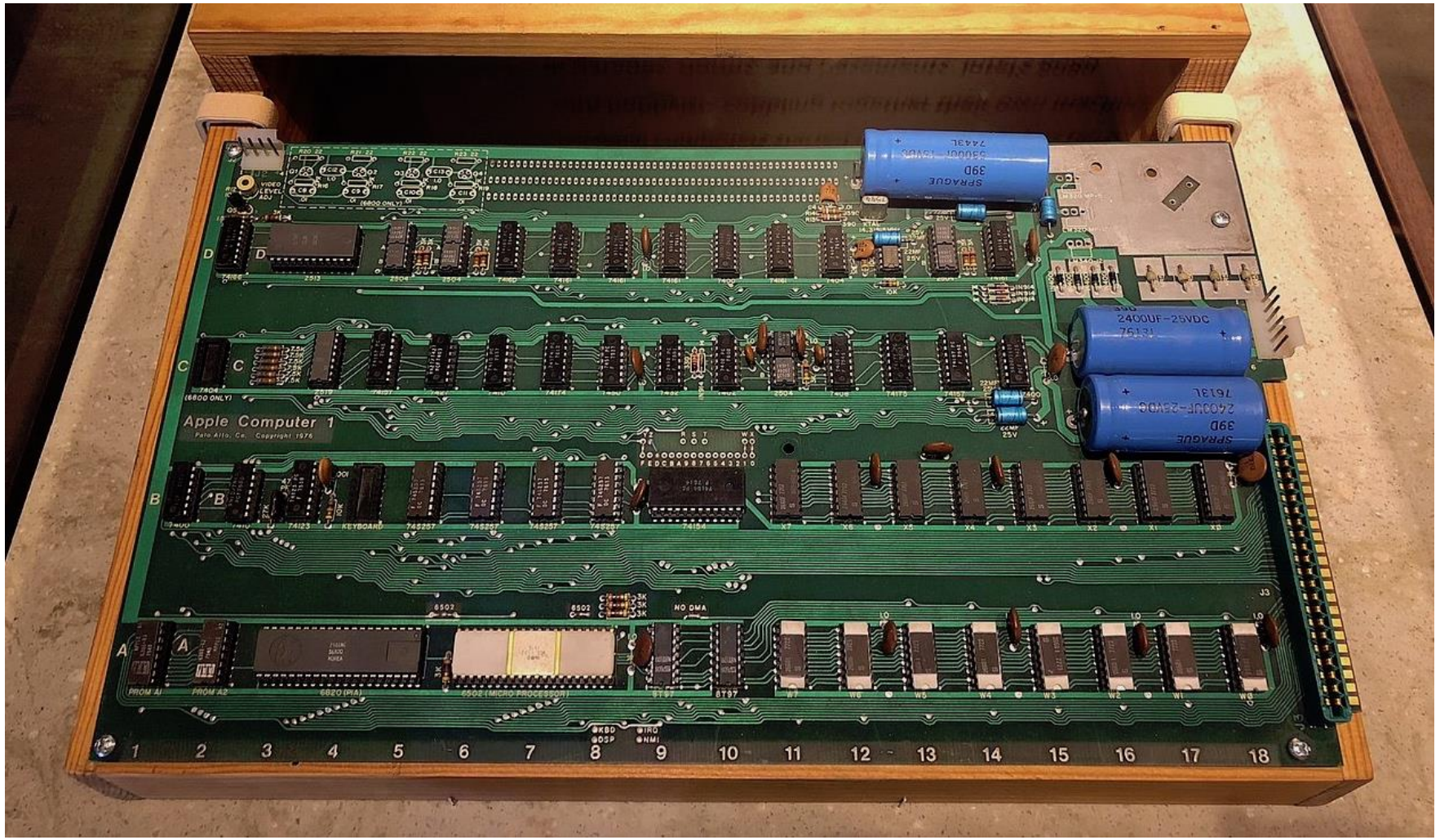
(Pictures are from the Internet)

Lecture Outline

- Encoder and decoder 编码器与解码器
- Arithmetic circuits 算术电路
  - Adder 加法器
  - Subtractor 减法器
  - Comparator 比较器
  - Arithmetic Logic Unit (ALU) 算术逻辑单元
  - Multiplier 乘法器
- Memory arrays 存储器阵列
- Computer architecture 计算机体系结构
- Assembly language 汇编语言
- Instruction cycle 指令周期

Early computer built with digital IC modules

- Apple I computer

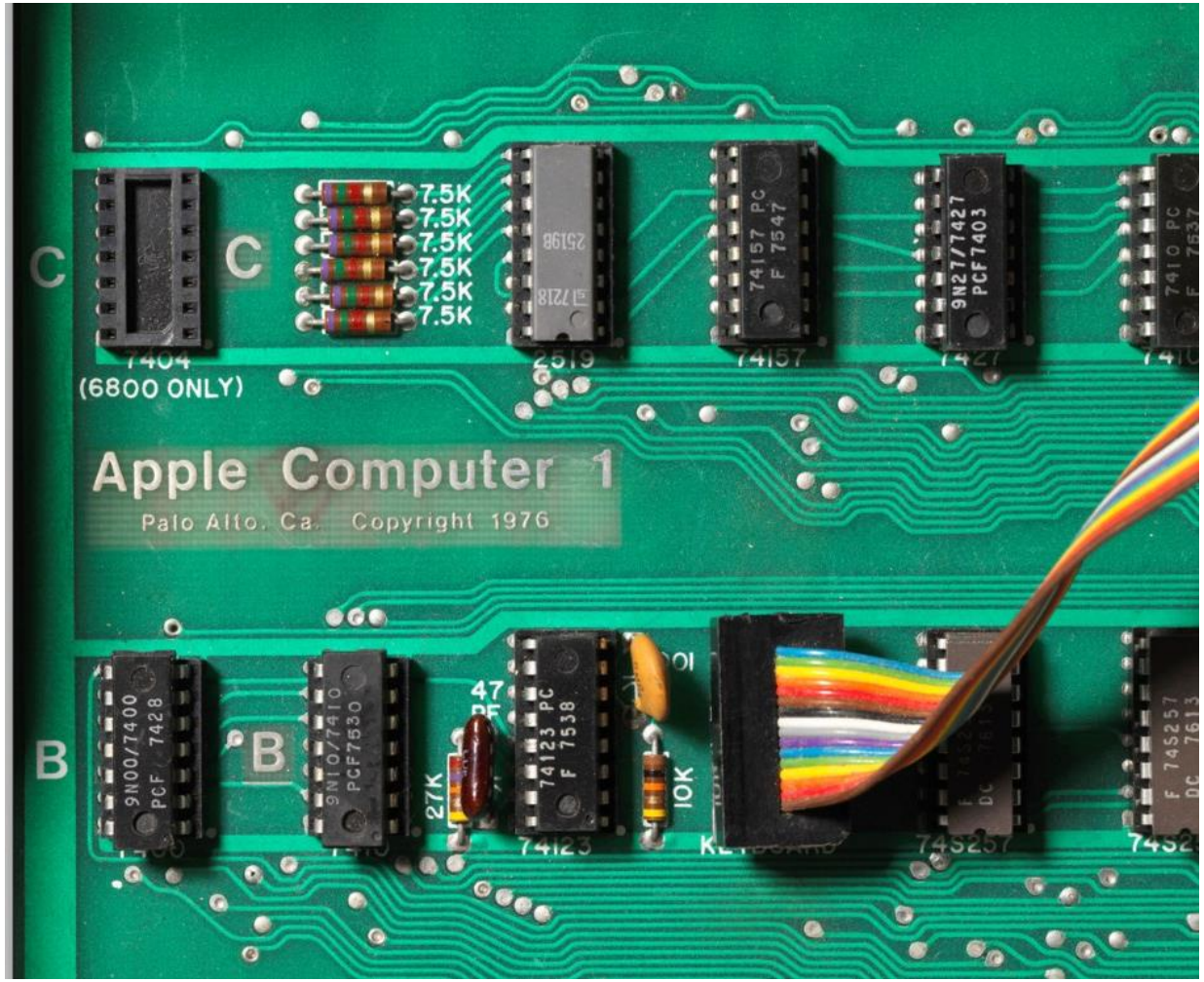


(Pictures are from the Internet)

7400-series integrated circuits

- The 7400 series of integrated circuits (ICs) were one of the most popular logic families of transistor-transistor logic (TTL) logic chips.

  7400系列是历史上使用最为广泛的一系列晶体管－晶体管逻辑（transistor-transistor logic, TTL）集成电路。它最初由德州仪器公司制造，在1960年代和1970年代被用于构架小型和主板计算机。



- List of 7400-series integrated circuits - Wikipedia

(Pictures are from the Internet)

How do we communicate with computers?

Normal people's language
(decimal 十进制)

0 8
1 9
2 10
3 11
4 12
5 13
6 14
7 15

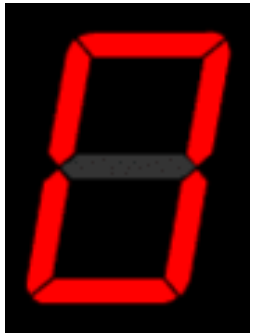
Programmers' language
(hexadecimal 十六进制)

0x0 0x8
0x1 0x9
0x2 0xA
0x3 0xB
0x4 0xC
0x5 0xD
0x6 0xE
0x7 0xF

Machines' language
(binary 二进制)

0000 1000
0001 1001
0010 1010
0011 1011
0100 1100
0101 1101
0110 1110
0111 1111

Normal people's understanding



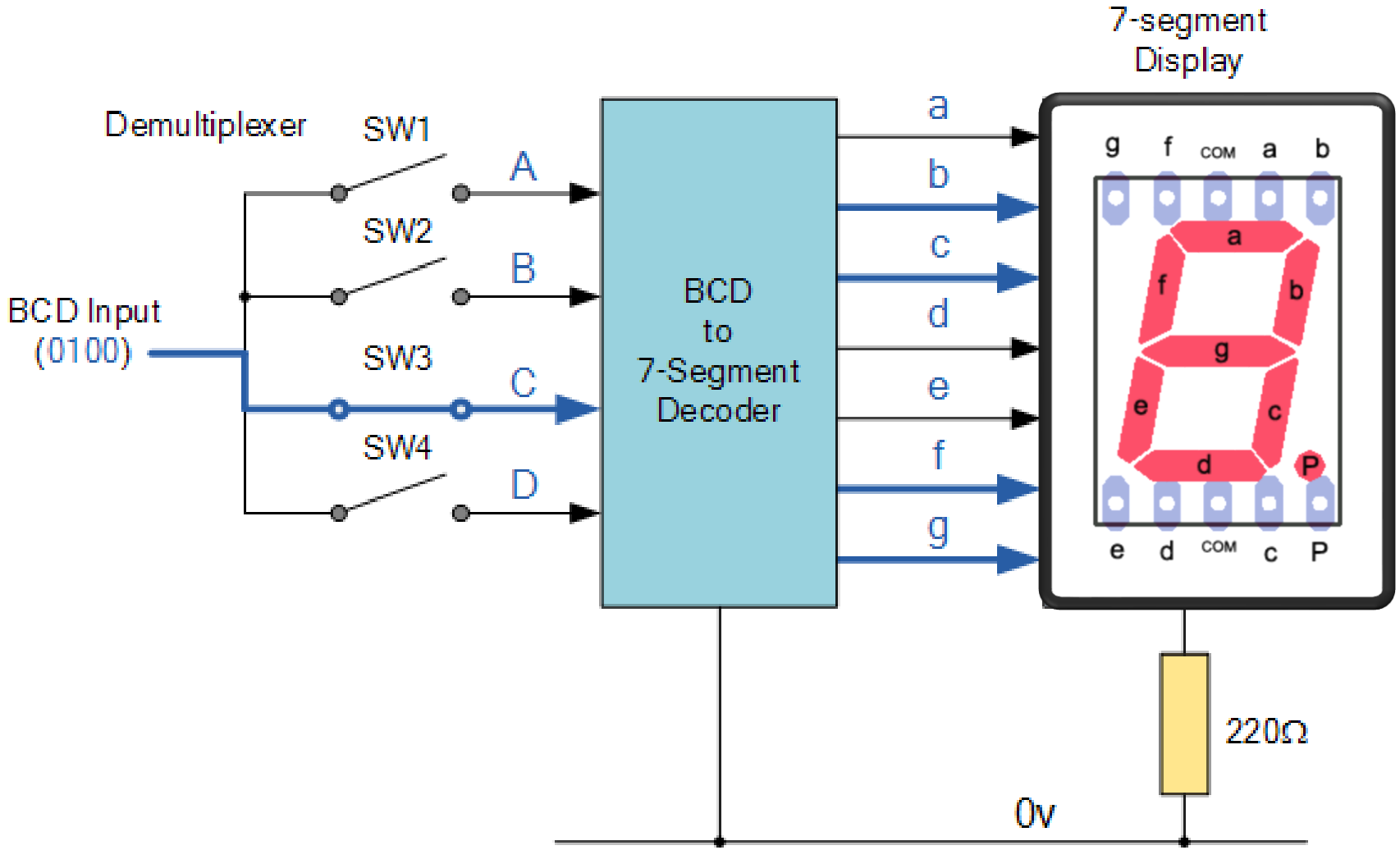
the simplest seven-segment display

(Pictures are from the Internet)

Seven segment decoder

(BCD: binary-coded decimal)

0 → 0000
1 → 0001
2 → 0010
3 → 0011
4 → 0100
5 → 0101
6 → 0110
7 → 0111
8 → 1000
9 → 1001



(Pictures are from the Internet)

BCD to seven-segment decoder

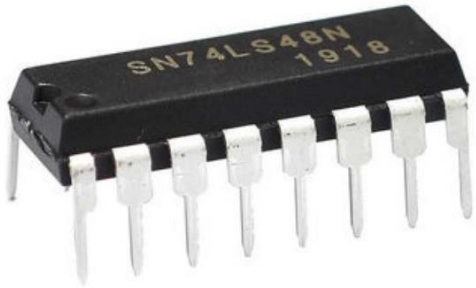
- Truth table

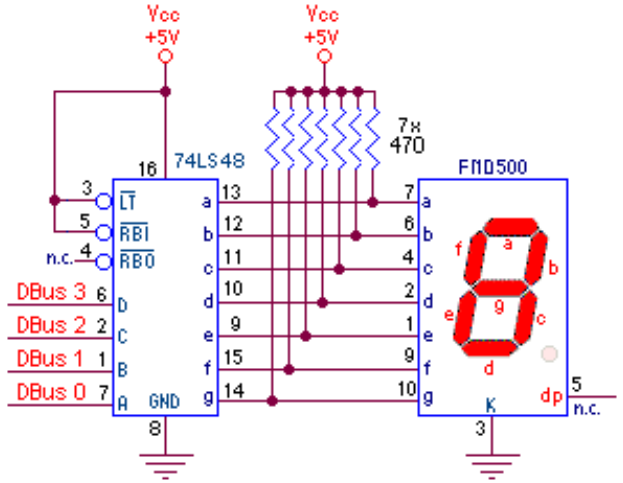
DESIMAL	D	C	B	A	a	b	c	d	e	f	g	7-LED
0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	1	1	0	0	1	1	1	1	1
2	0	0	1	0	0	0	1	0	0	1	0	2
3	0	0	1	1	0	0	0	0	1	1	0	3
4	0	1	0	0	1	0	0	1	1	0	0	4
5	0	1	0	1	0	1	0	0	1	0	0	5
6	0	1	1	0	1	1	0	0	0	0	0	6
7	0	1	1	1	0	0	0	1	1	1	1	7
8	1	0	0	0	0	0	0	0	0	0	0	8
9	1	0	0	1	0	0	0	1	1	0	0	9
10	1	0	1	0	1	1	1	0	0	1	0	c
11	1	0	1	1	1	1	0	0	1	1	0	3
12	1	1	0	0	1	0	1	1	1	0	0	9
13	1	1	0	1	0	1	1	0	1	0	0	6
14	1	1	1	0	1	1	1	0	0	0	0	6
15	1	1	1	1	1	1	1	1	1	1	1	8

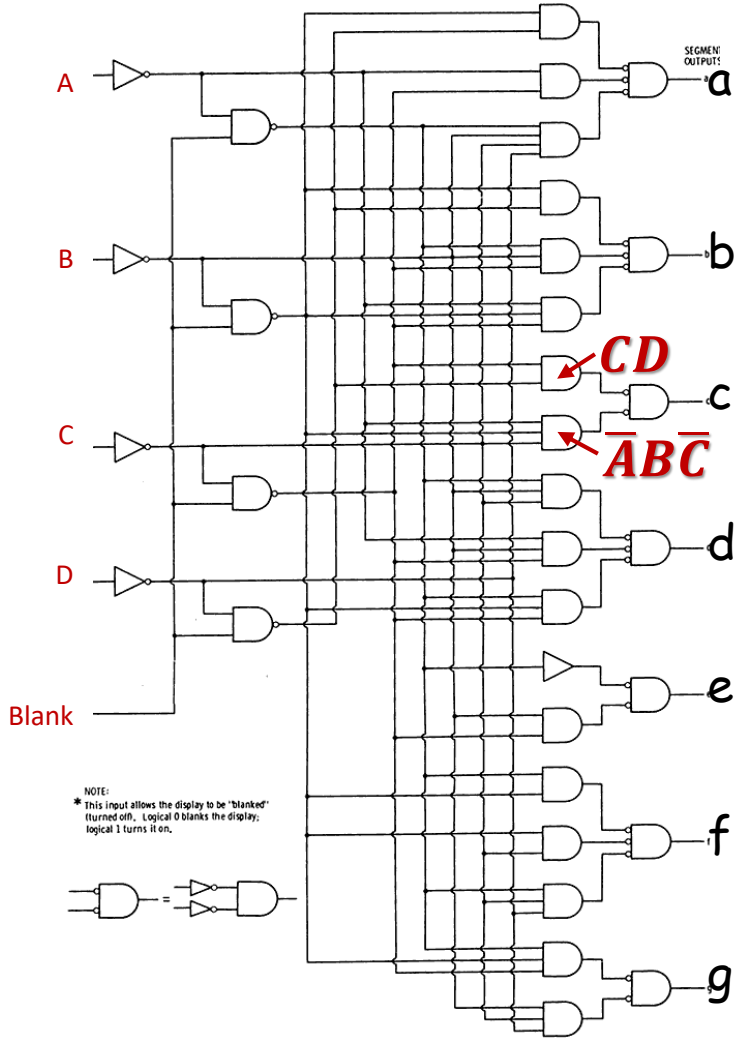
Example:

$$c = \overline{CD} + \overline{A}B\overline{C}$$
$$= \overline{\overline{CD}\ \overline{\overline{A}B\overline{C}}}$$

- 74LS48 (IC) a BCD to 7-Segment Decoder







How to add binary numbers?

- Decimal addition

$$\begin{array}{r} {\scriptstyle 1} \\ 23 \\ +\,28 \\ \hline 51 \end{array}$$

- Binary addition

$$\begin{array}{r} {\scriptstyle 1\;1} \\ 10111 \\ +\,11100 \\ \hline 110011 \end{array}$$

$$51_{10}=110011_2$$

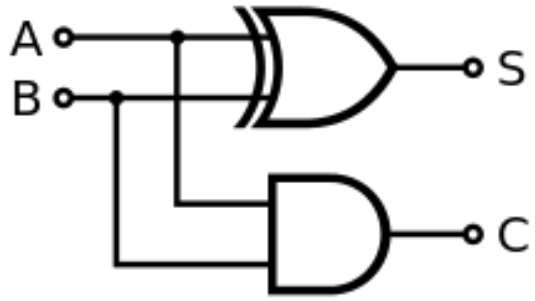
Half adder 半加器

Inputs		Outputs	
A	B	C_{out}	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0

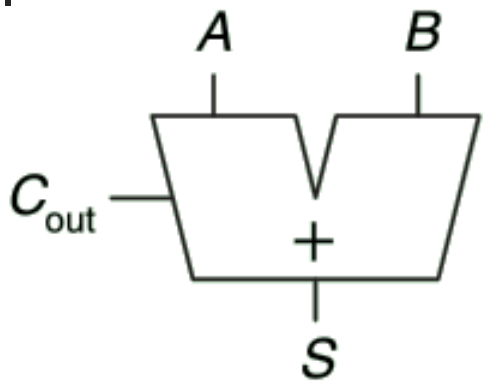
The sum: $\boldsymbol{S = A \oplus B}$

The carry (out): $\boldsymbol{C = AB}$

Logic diagram



Symbol



Full adder 全加器

Inputs			Outputs	
A	B	C_{in}	C_{out}	S
0	0	0	0	0
1	0	0	0	1
0	1	0	0	1
1	1	0	1	0
0	0	1	0	1
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1

$$S = (A \oplus B)\overline{C}_{in} + (\overline{A \oplus B})C_{in}$$

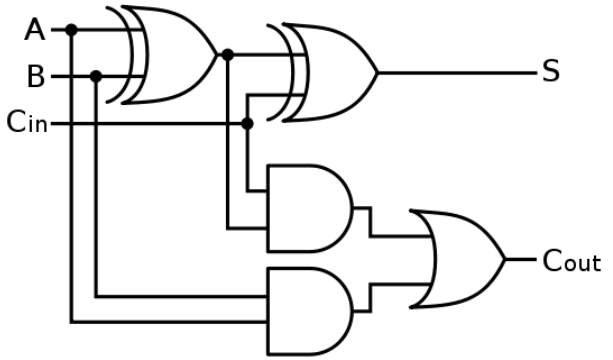
$$= A \oplus B \oplus C_{in}$$

$$C_{out} = AB\overline{C}_{in} + A\overline{B}C_{in} + \overline{A}BC_{in} + ABC_{in}$$

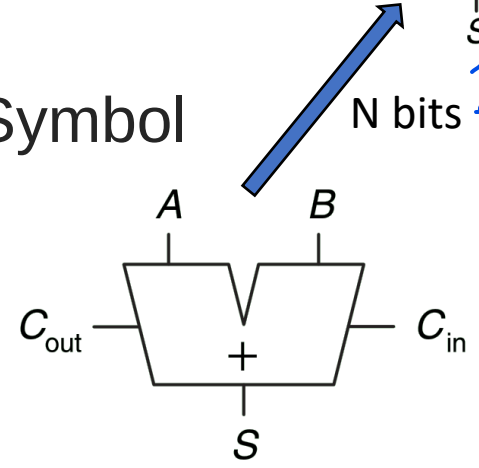
$$= AB + (A + B)C_{in}$$

or $AB + (A \oplus B)C_{in}$

Logic diagram



Symbol



How does a computer read negative number

- Unsigned and signed binary numbers
  - 8 bit unsigned number
    0 to 255, i.e., 0 ~ 0xFF
  - 8 bit signed number
    -128 to 127, i.e., ?? to 0x7F

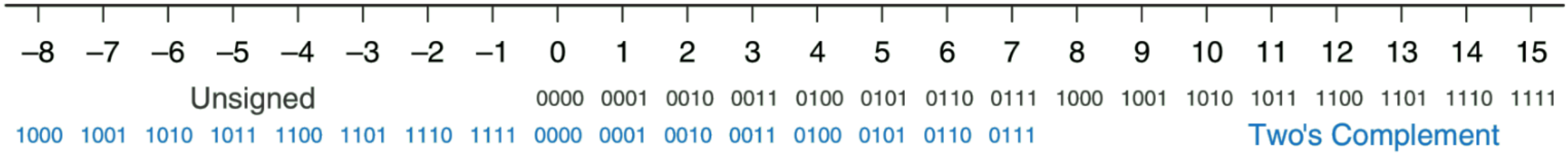
How does the computer understands the minus sign "−"?

- Two's complement numbers (2的补数)
  1. Invert all bits
  2. Add "1" to the last significant bit

Example for four bits number:

$$5_{10} = 0101_2$$
$$(-5_{10}) = 1010_2 + 1$$
$$= 1011_2$$

补数（complement）是对于给定的进位制，相加后能使自然数 a 的位数增加 1 的最小的数。



Binary subtraction

- Decimal subtraction

$$\begin{array}{r} 15 \\ -\ \ 5 \\ \hline 10 \end{array}$$

$$\begin{array}{r} 15 \\ +\ (-5) \\ \hline 10 \end{array}$$

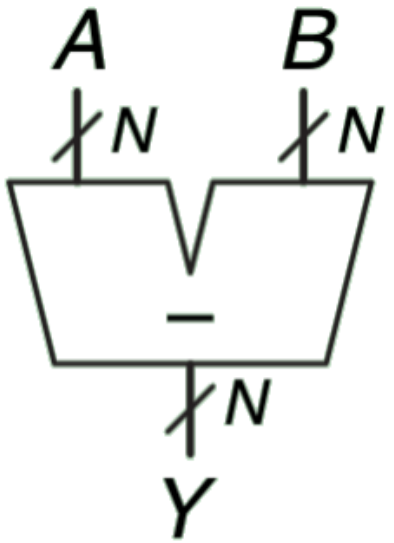
- Binary subtraction

$$\begin{array}{r} \scriptstyle 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1\ \ 1 \quad \text{(carry)} \\ 00001111 \\ +\ 11111011 \\ \hline 00001010 \end{array}$$

$(10_{10} = 1010_2)$

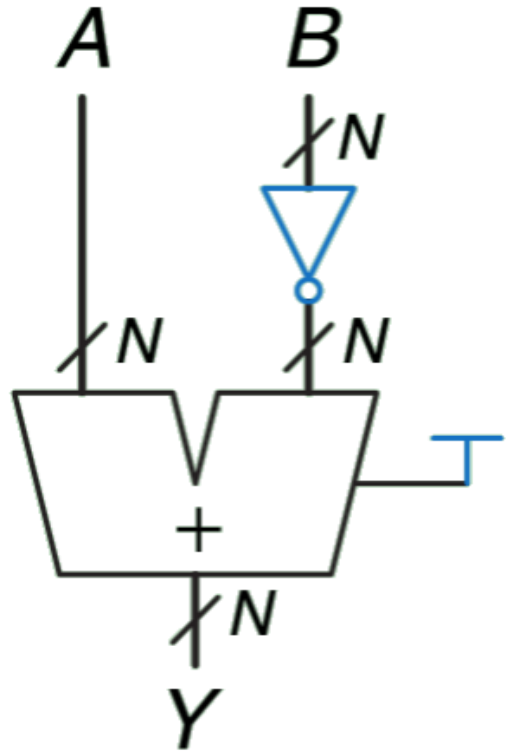
Subtractor

- Symbol



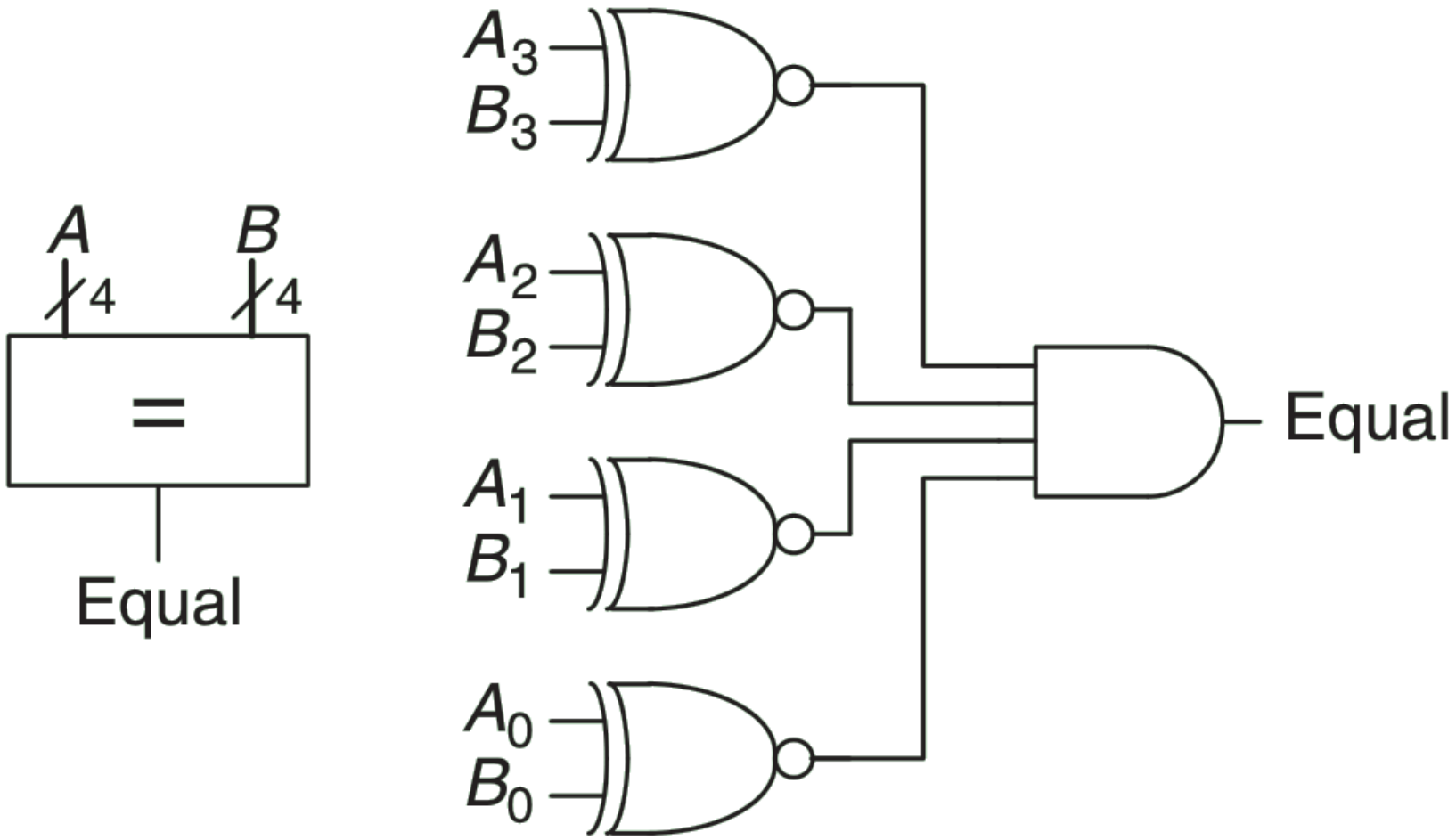
$$Y = A - B = A + \bar{B} + 1$$

- Implementation
  - Based on an full adder



Equality comparator 比较器（仅比较相等与否）

- Symbol



Magnitude comparator 大小比较器

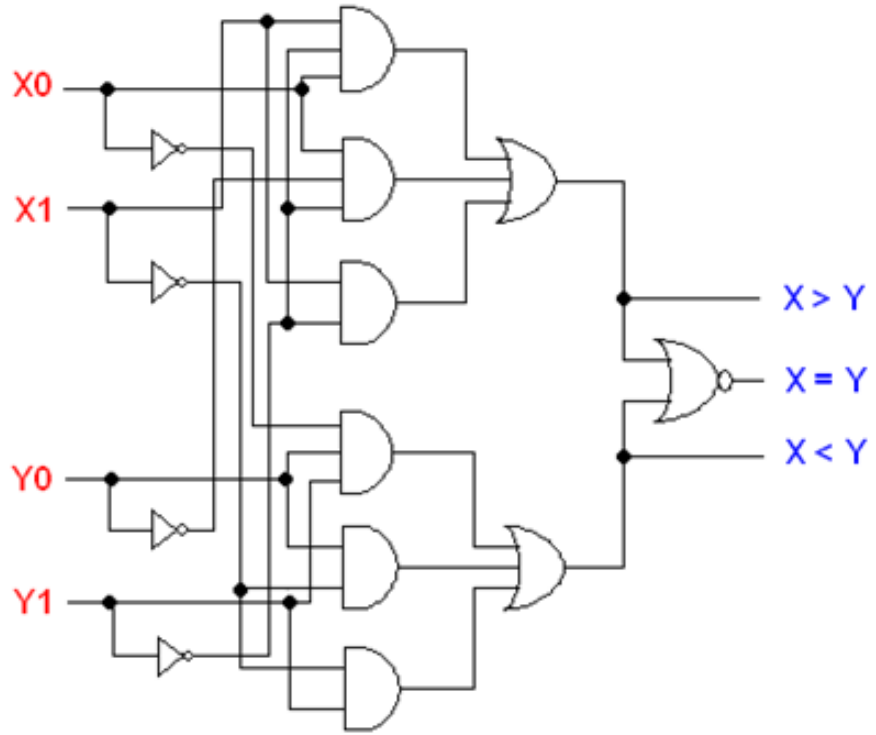
- Truth table of 2-bit magnitude comparator

x_1	x_0	y_1	y_0	x<y	x=y	x>y
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	1	1	0	0
1	0	0	0	0	0	1
1	0	0	1	0	0	1
1	0	1	0	0	1	0
1	0	1	1	1	0	0
1	1	0	0	0	0	1
1	1	0	1	0	0	1
1	1	1	0	0	0	1
1	1	1	1	0	1	0

$$S_{X>Y} = X_1 X_0 \bar{Y}_1 + X_0 \bar{Y}_0 \bar{Y}_1 + X_1 \bar{Y}_1$$

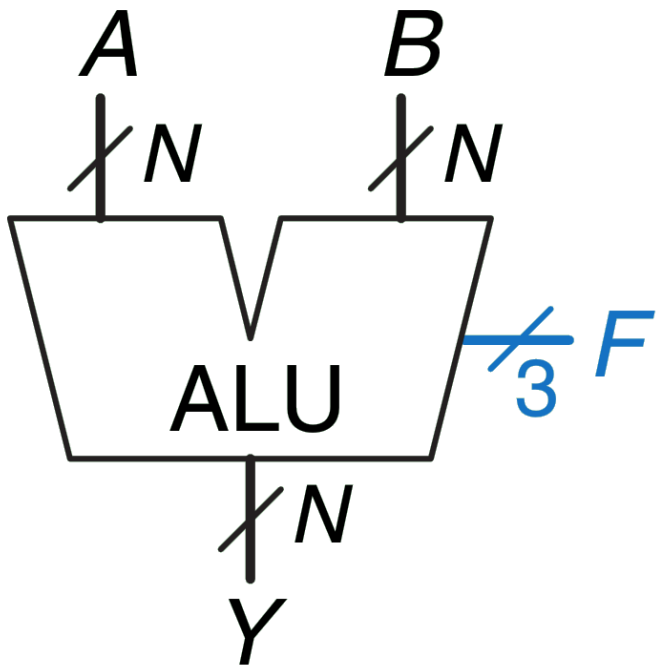
$$S_{X<Y} = \bar{X}_0 Y_0 Y_1 + \bar{X}_1 Y_0 + \bar{X}_1 Y_1$$

$$S_{X=Y} = \overline{S_{X>Y} + S_{X<Y}}$$



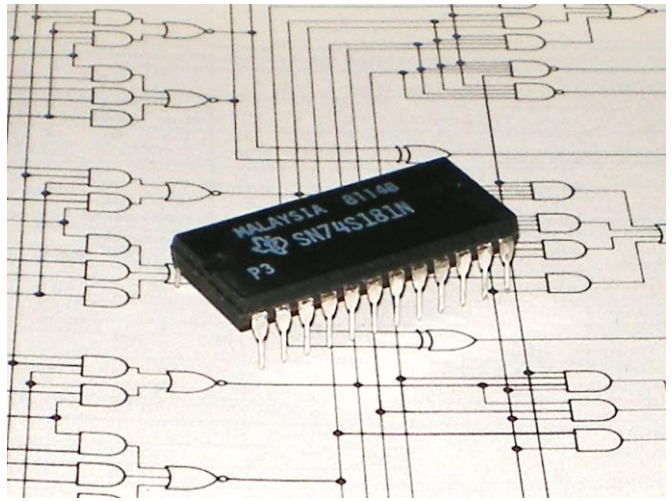
Arithmetic logic unit (ALU) 算术逻辑单元

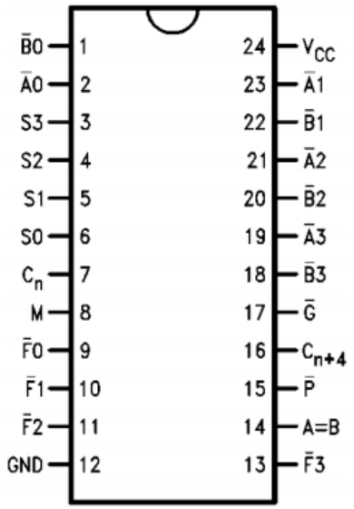
- Combines a variety of mathematical and logical operations into a single unit a combinational logic circuit



$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

Example: the 74181, a four-bit ALU





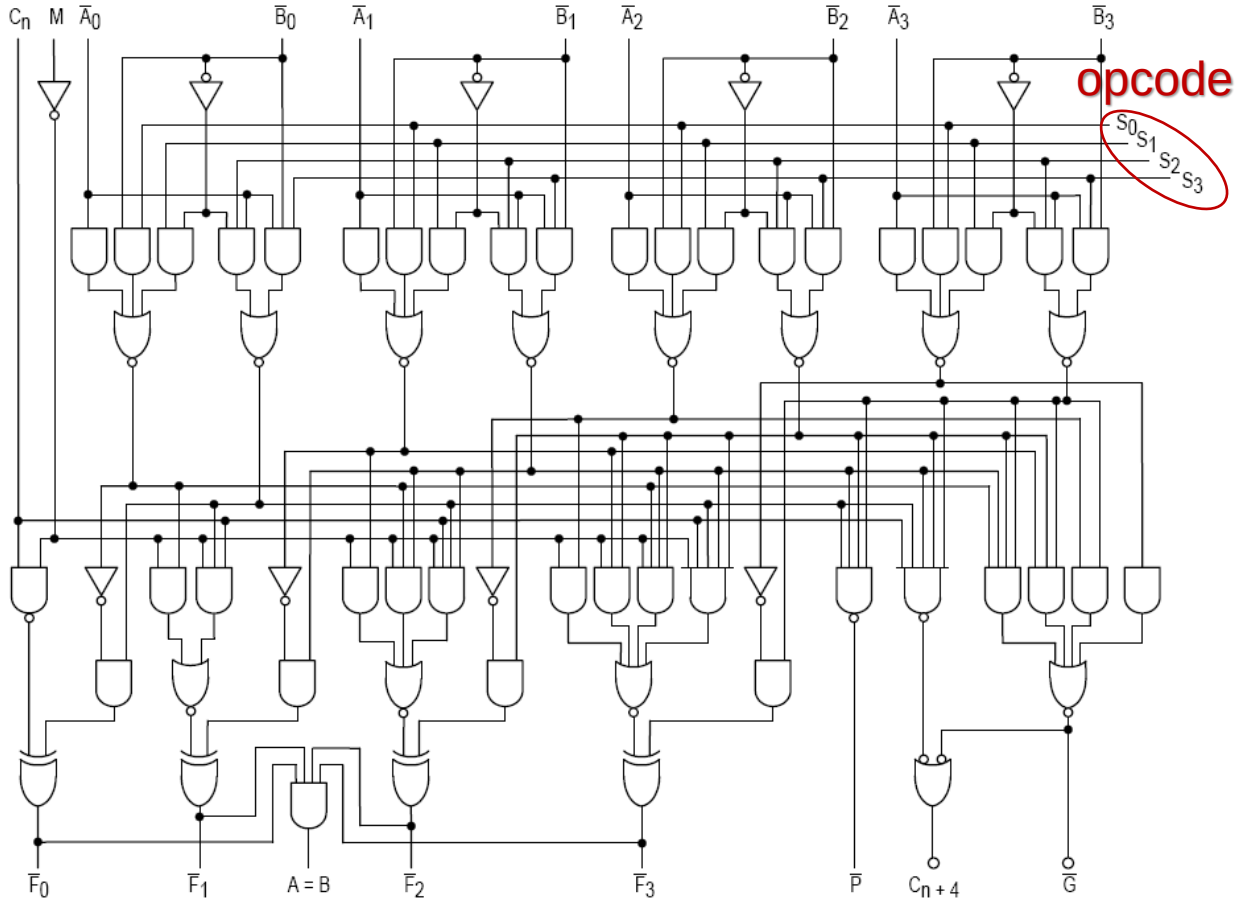
Pin Names	Description
$\overline{A}0$–$\overline{A}3$	Operand Inputs (Active LOW)
$\overline{B}0$–$\overline{B}3$	Operand Inputs (Active LOW)
S0–S3	Function Select Inputs
M	Mode Control Input
C_n	Carry Input
$\overline{F}0$–$\overline{F}3$	Function Outputs (Active LOW)
A = B	Comparator Output
$\overline{G}$	Carry Generate Output (Active LOW)
$\overline{P}$	Carry Propagate Output (Active LOW)
C_{n+4}	Carry Output

Realization

- Function table

S3	S2	S1	S0	Logic (M = H)	Arithmetic (Note 2) (M = L) (C_n = L)
L	L	L	L	$\overline{A}$	A minus 1
L	L	L	H	$\overline{AB}$	AB minus 1
L	L	H	L	$\overline{A} + B$	$A\overline{B}$ minus 1
L	L	H	H	Logic 1	minus 1
L	H	L	L	$\overline{A} + \overline{B}$	A plus $(A + \overline{B})$
L	H	L	H	$\overline{B}$	AB plus $(A + \overline{B})$
L	H	H	L	$\overline{A} \oplus \overline{B}$	A minus B minus 1
L	H	H	H	$A + \overline{B}$	$A + \overline{B}$
H	L	L	L	$\overline{A}$ B	A plus $(A + B)$
H	L	L	H	$A \oplus B$	A plus B
H	L	H	L	B	$A\overline{B}$ plus $(A + B)$
H	L	H	H	$A + B$	$A + B$
H	H	L	L	Logic 0	A plus A (Note 1)
H	H	L	H	$A\overline{B}$	AB plus A
H	H	H	L	AB	$A\overline{B}$ minus A
H	H	H	H	A	A

- Logic realization



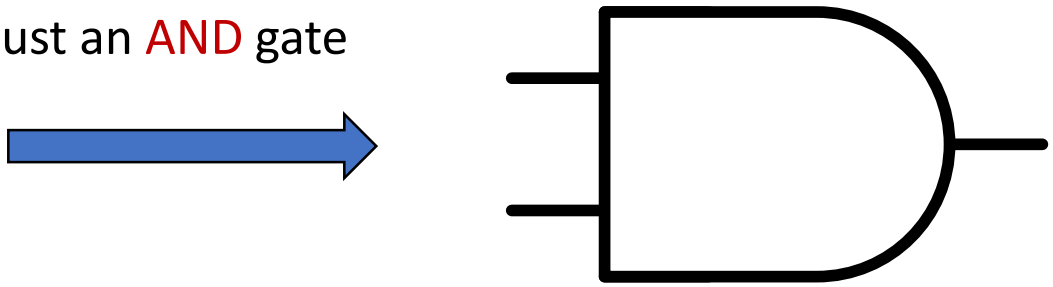
One bit multiplication

- Truth table

$$P = A \times B$$

A	B	P
0	0	0
0	1	0
1	0	0
1	1	1

Just an AND gate



More bits multiplication

- Decimal

```
    230   multiplicand
×    42   multiplier
-------
    460   partial
+ 920     products
-------
   9660   result
```

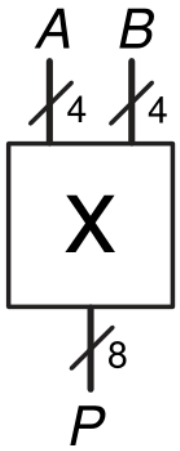
$2\ 30 \times 42 = 9660$

- Binary

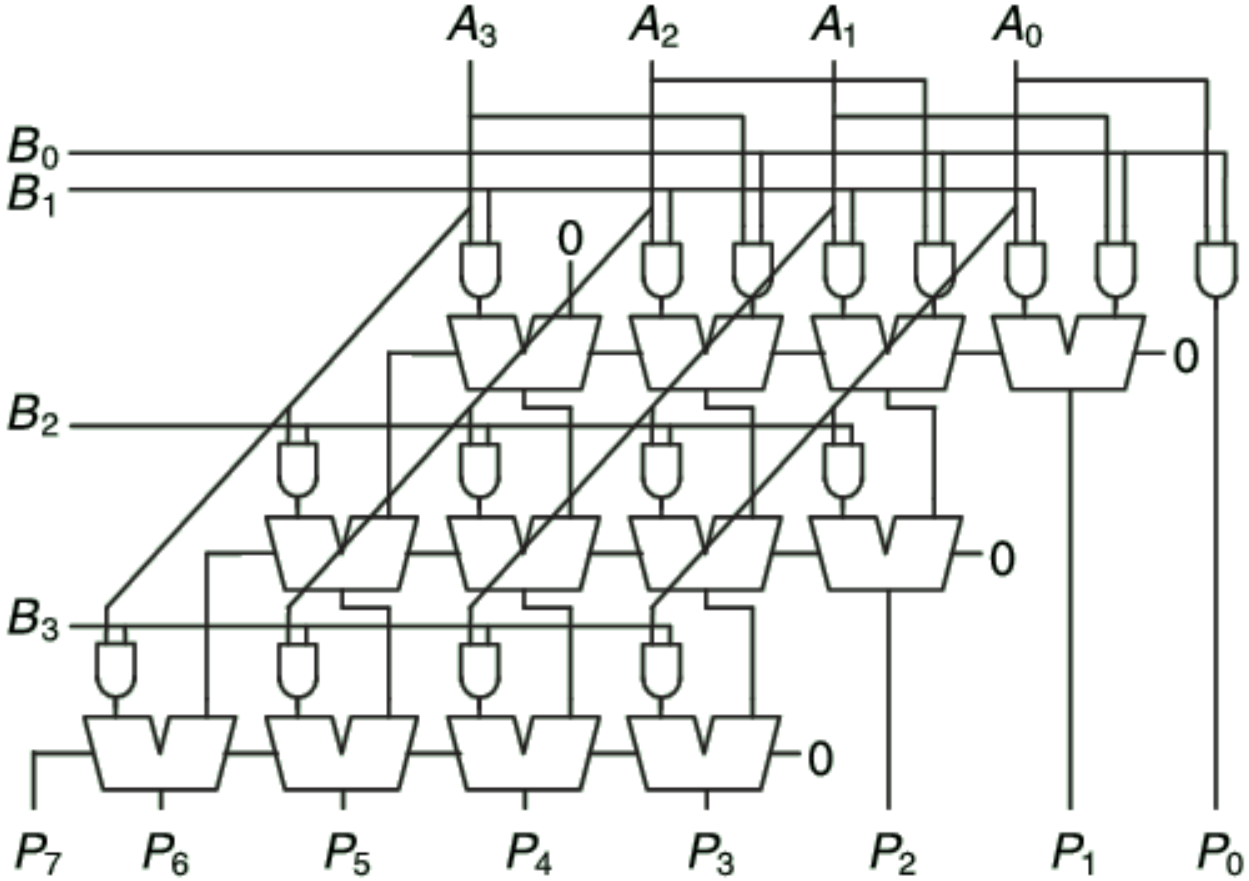
```
     0101
×    0111
---------
     0101
    0101
   0101
+ 0000
---------
  0100011
```

$5 \times 7 = 35$

Implementation



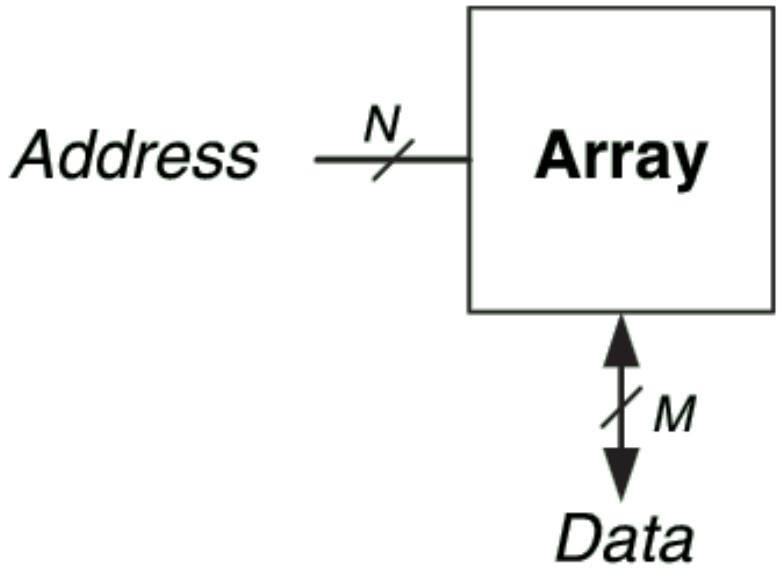
$$
\begin{array}{cccccccc}
 & & A_3 & A_2 & A_1 & A_0 & & \\
\times & & B_3 & B_2 & B_1 & B_0 & & \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & & \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & & & \\
 & A_3B_2 & A_2B_2 & A_1B_2 & A_0B_2 & & & \\
+ & A_3B_3 & A_2B_3 & A_1B_3 & A_0B_3 & & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}
$$



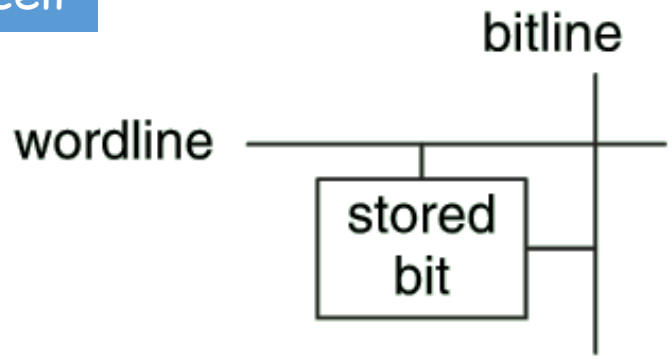
Memory array

- Two-dimensional array of memory cells
- The row is specified as *Address*
- Each row of data is called a *Word*
- The array contains 2^N *M-bit* words
- *Depth* = 2^N
- *Width* = 2^M
- Types:
  - Dynamic random access memory (DRAM)
  - Static random access memory (SRAM)
  - Read only memory (ROM)
  - etc.

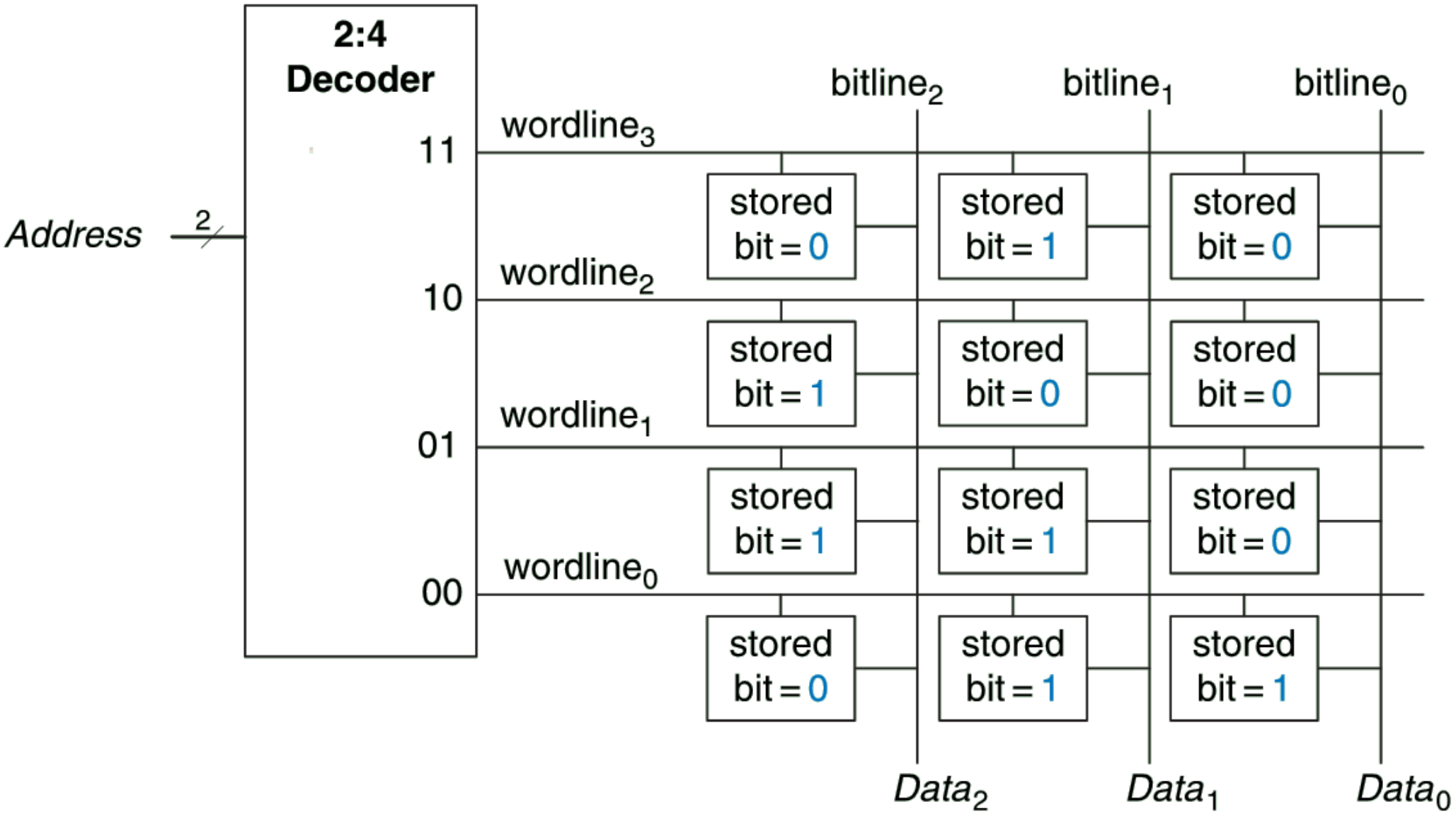
General symbol



Bit cell

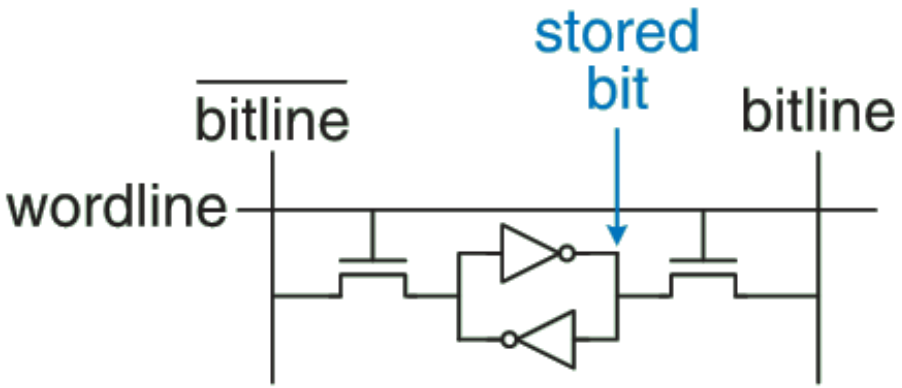


Example: 4×3 Memory Array

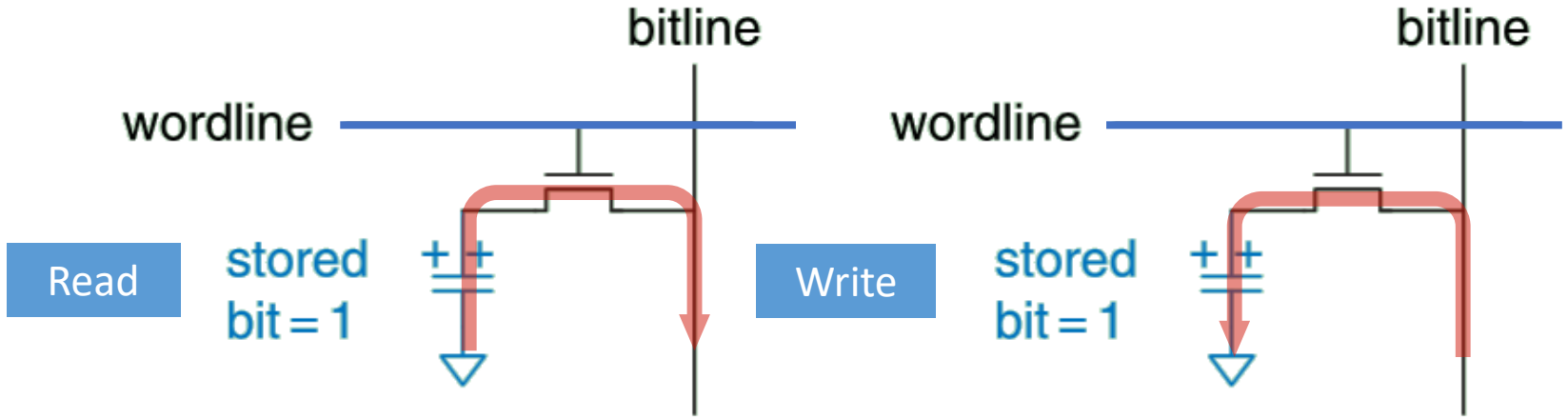


Volatile memory 易失（非永久）性存储器

- Static Random Access Memory (SRAM)

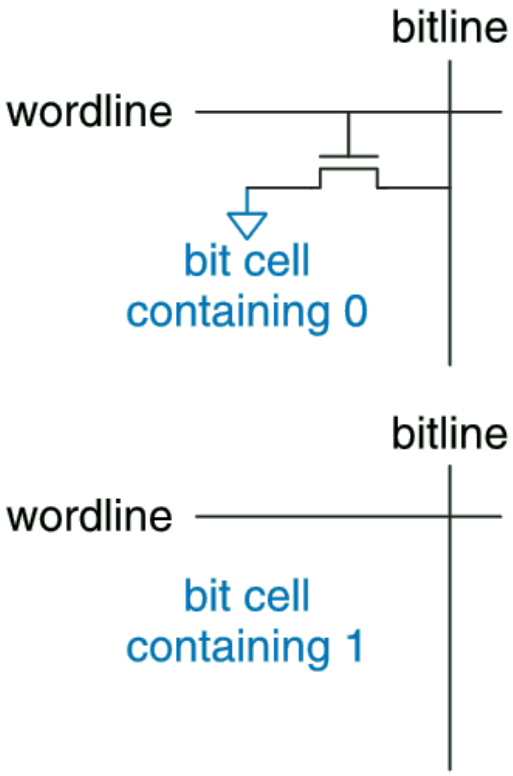


- Dynamic Random Access Memory (DRAM)

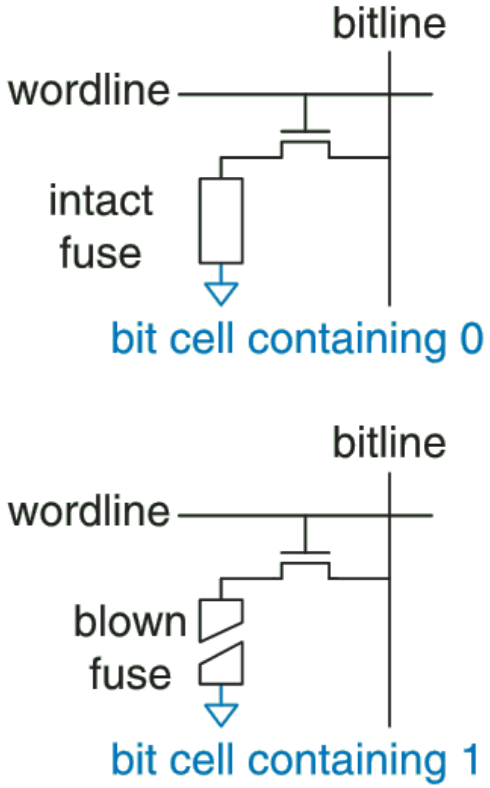


Nonvolatile memory

- Read only memory (ROM)



ROM bit cells



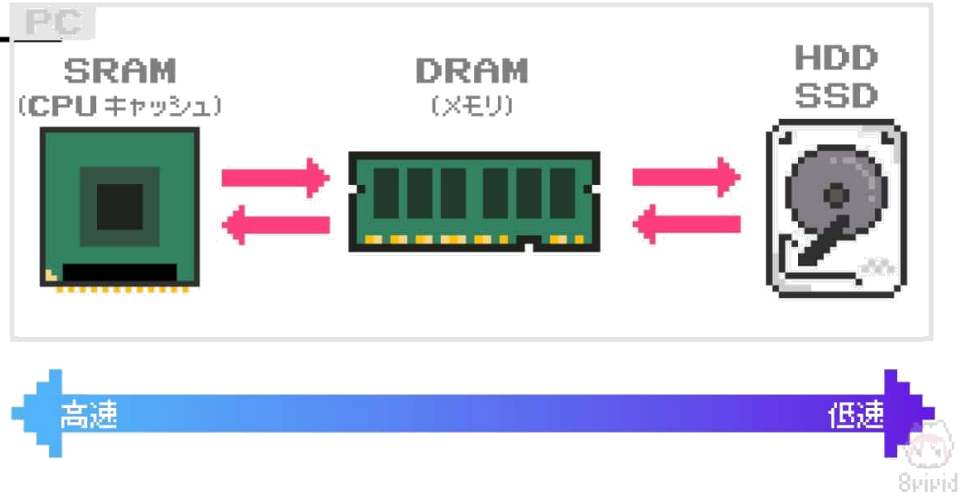
Programmable ROM

- Modern ROMs can programmed (written) as well. For example flash memory
- Generally, ROMs take a longer time to write then RAMs, but are nonvolatile.

Memory comparison

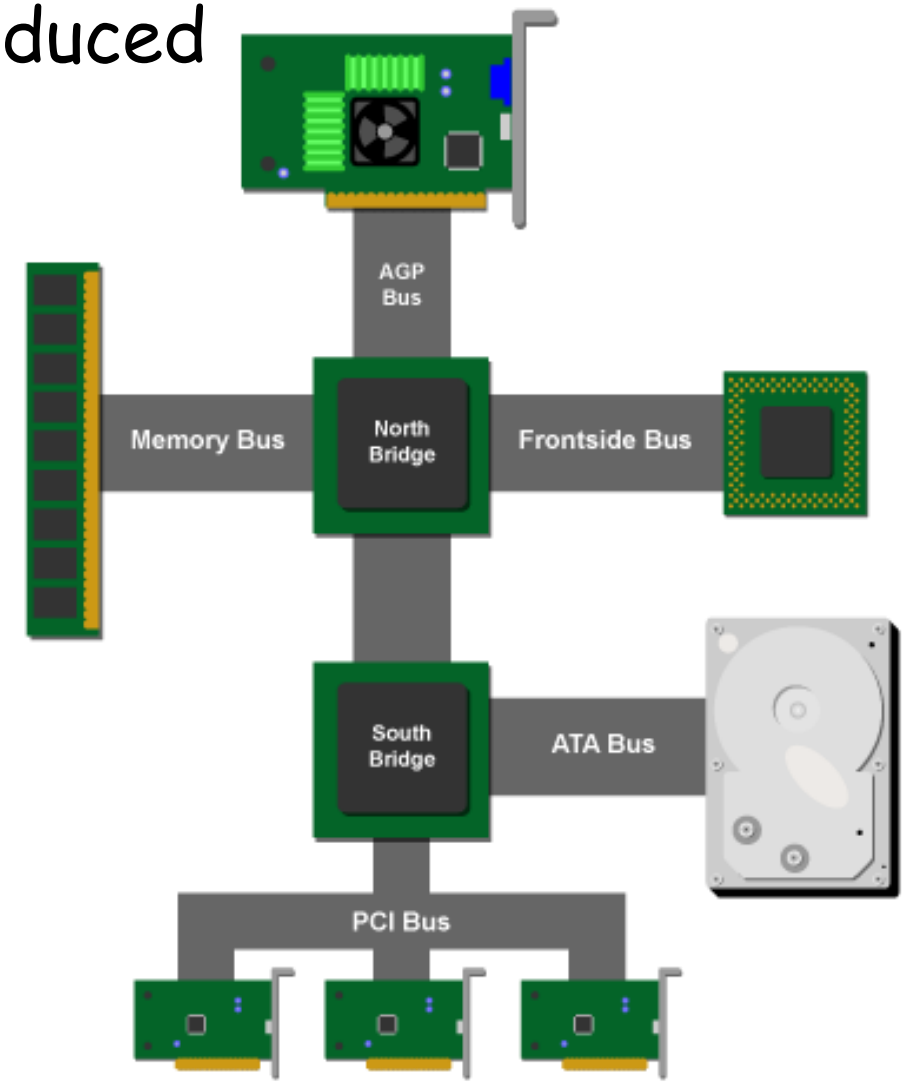
- The best memory type for a particular design depends on the *speed*, *cost*, and *power constraints*.

Memory Type		Transistors per Bit Cell	Latency
flip-flop		~20	fast
SRAM	register, cache	6	medium
DRAM	main memory	1	slow

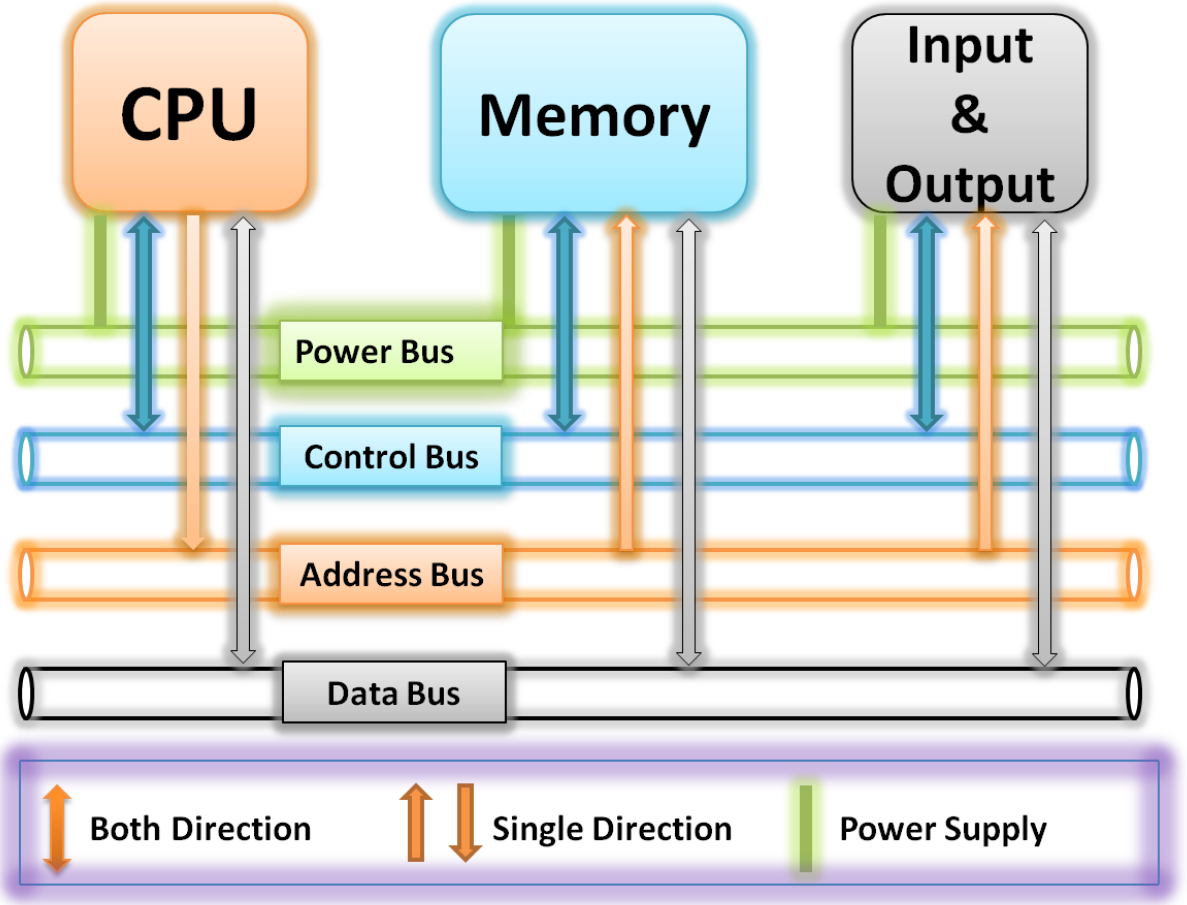


Computer organization

- In the last lecture, we introduced

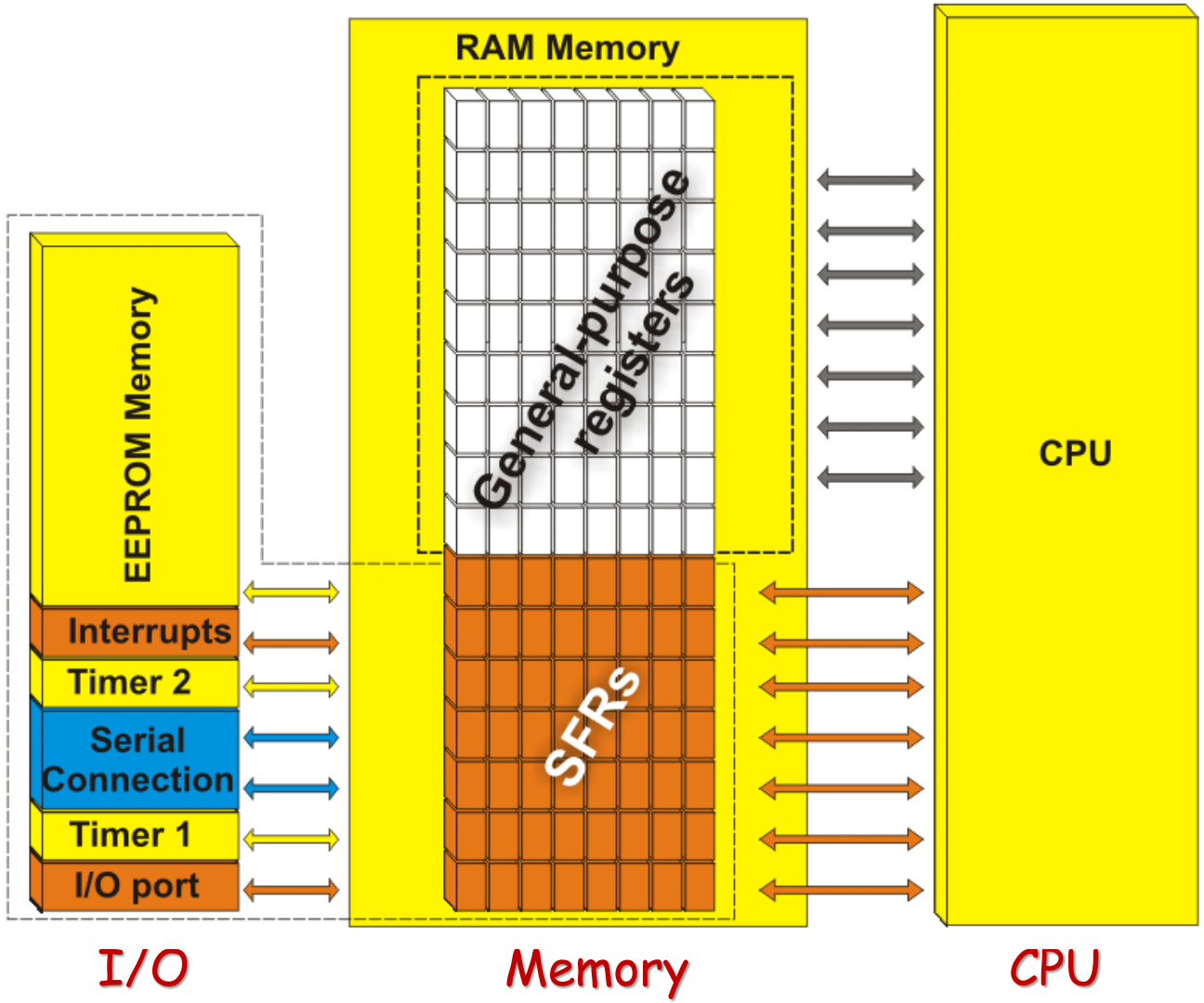


- Buses in a computer



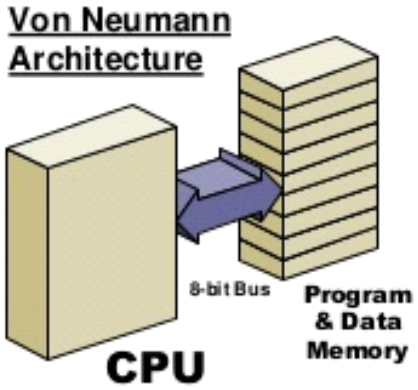
Computer architecture

- Defines the operations among CPU, memory, and I/O peripherals

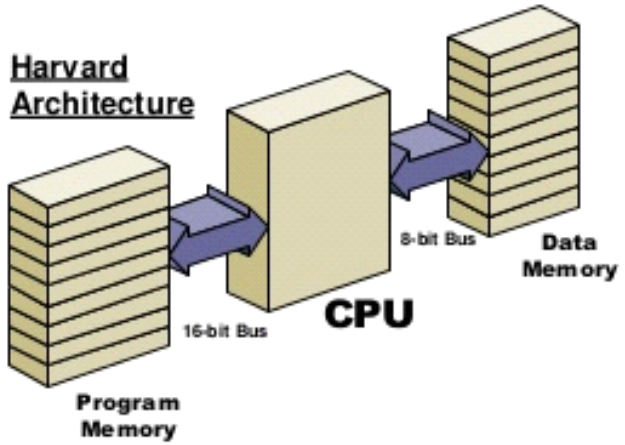


Two major architectures

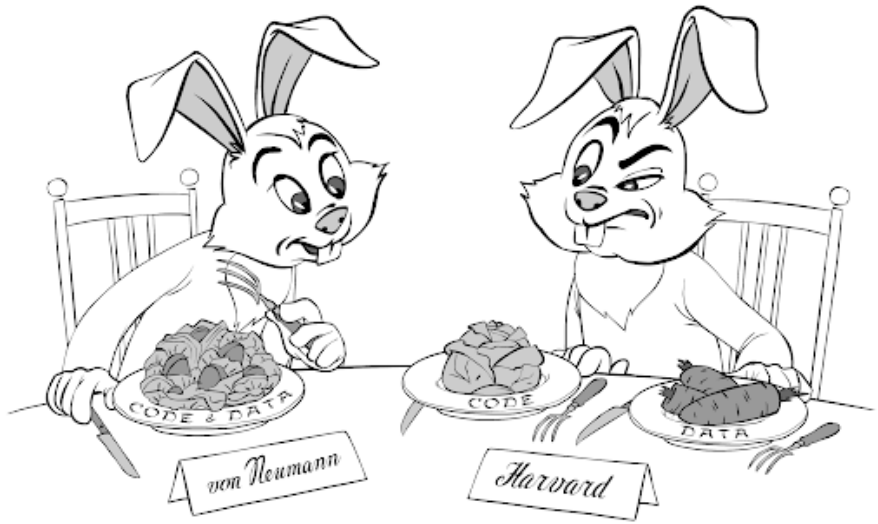
Harvard Architecture vs Von Neumann Architecture



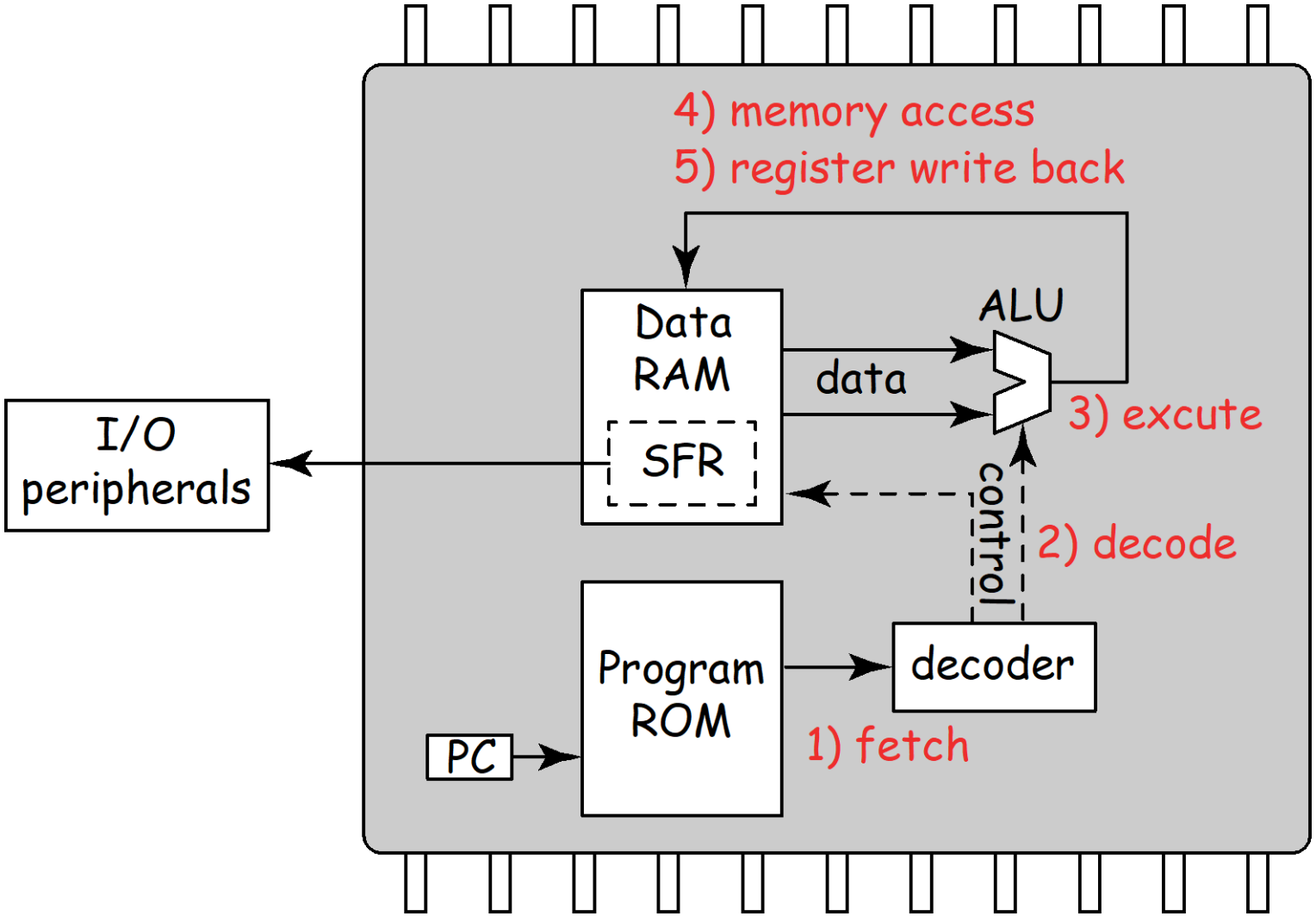
- Von Neumann Architecture: X86
  - Used single memory space for program and data.
  - Limits operating bandwidth

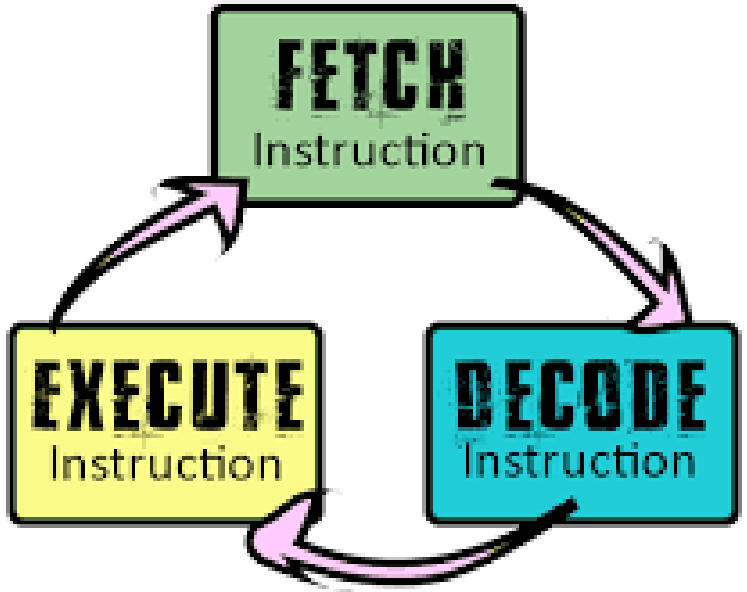


- Harvard Architecture: ARM
  - Uses two separate memory spaces for program instructions and data
  - Improved operating bandwidth
  - Allows for different bus ~~widths~~



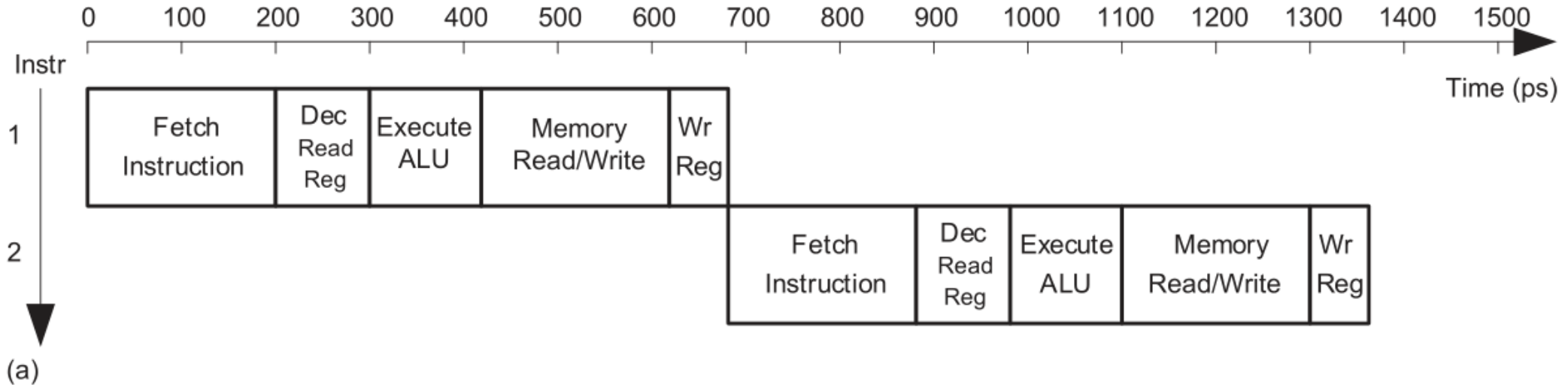
Instruction cycle



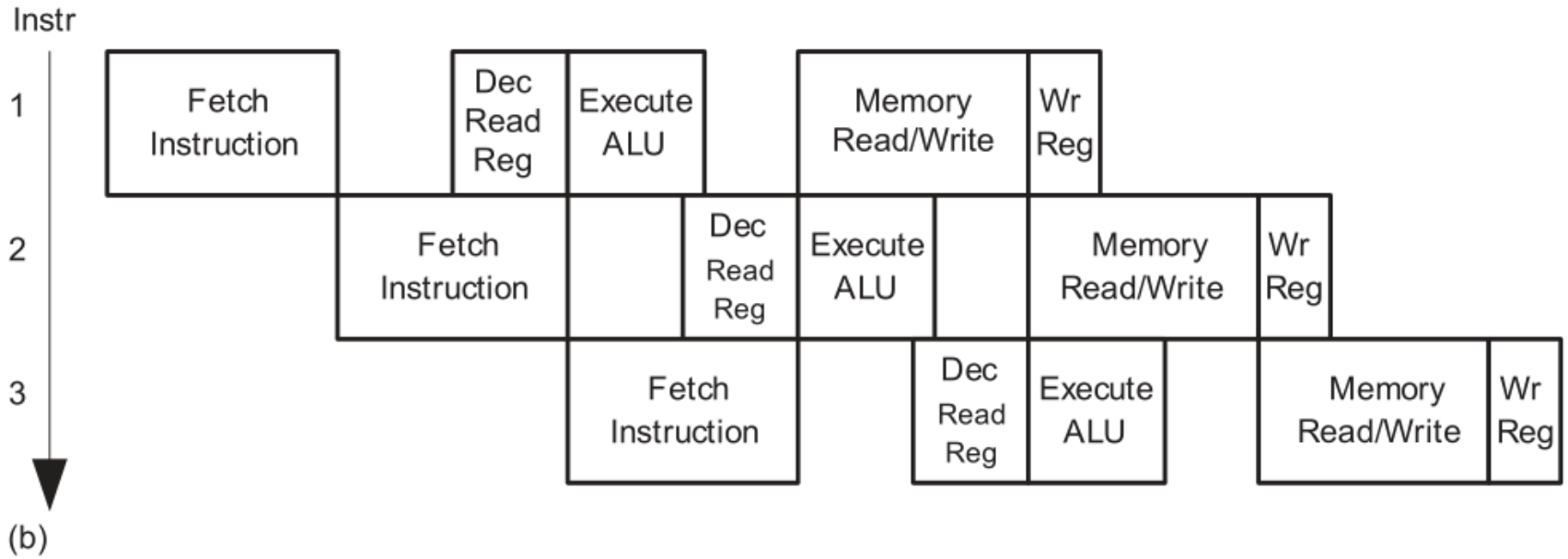


Pipelined processor

Single cycle processor

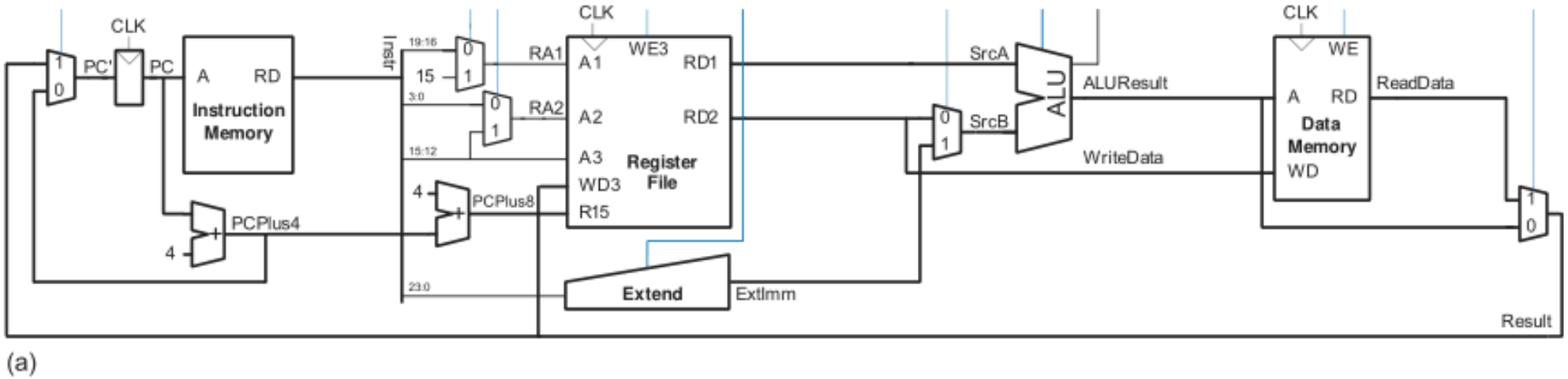


Pipelined processor

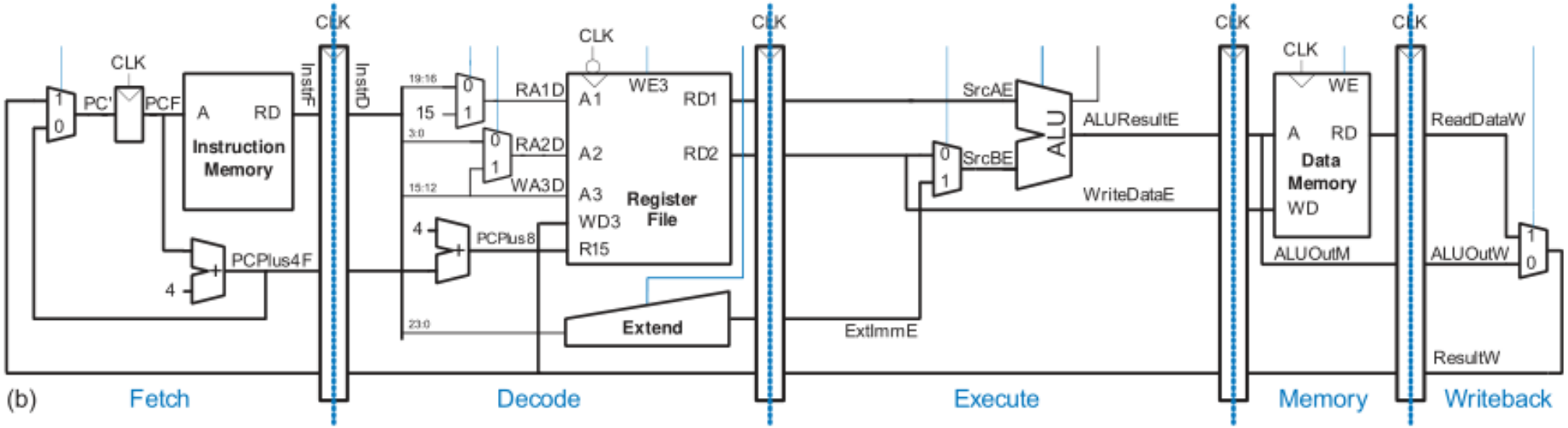


Datapaths

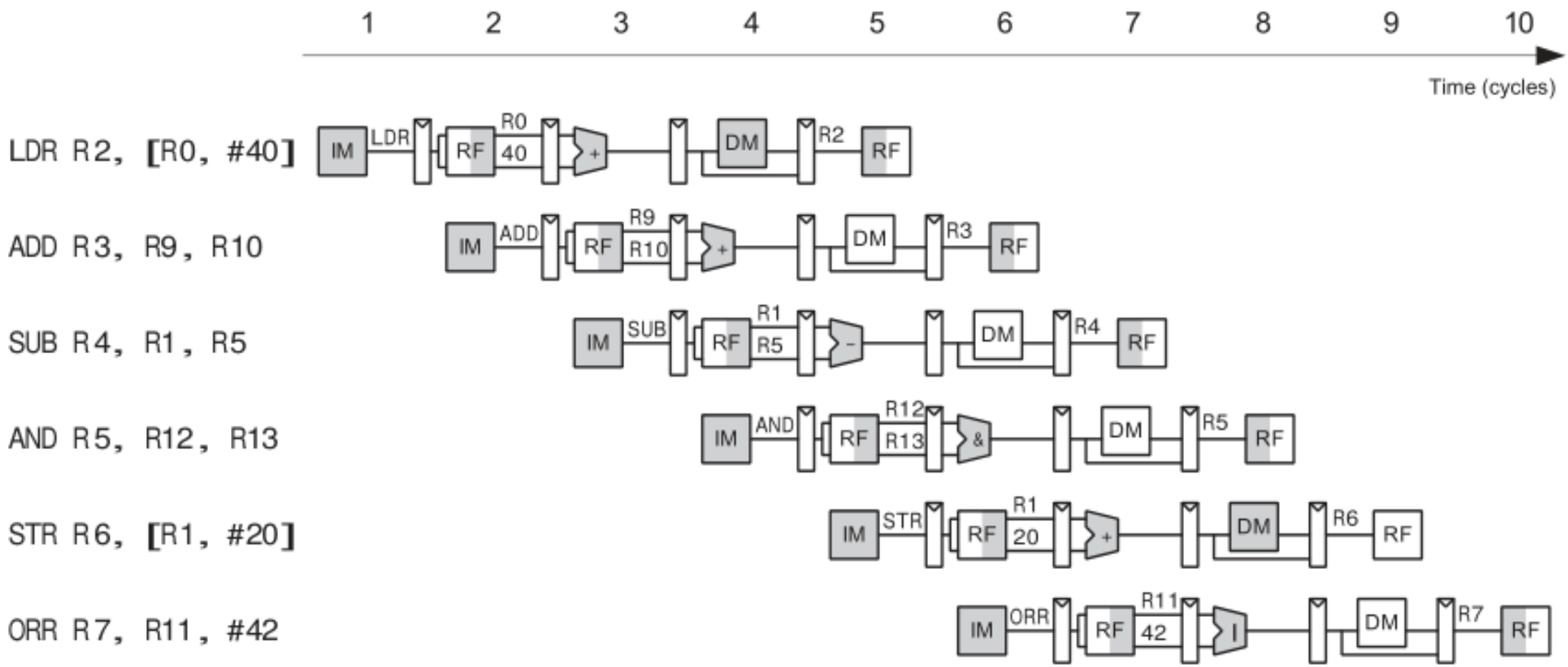
Single cycle processor



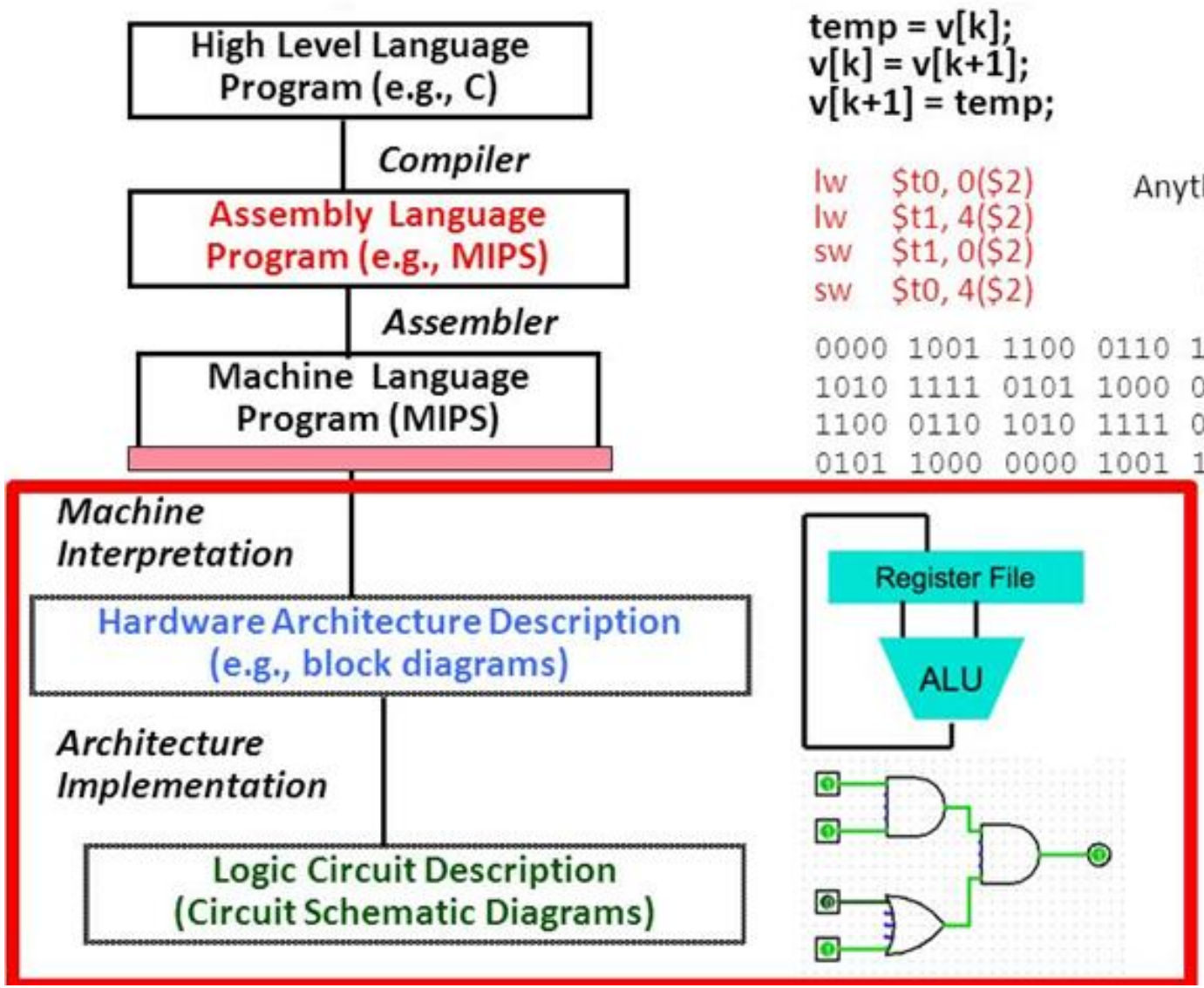
Pipelined processor



Pipelined processor example

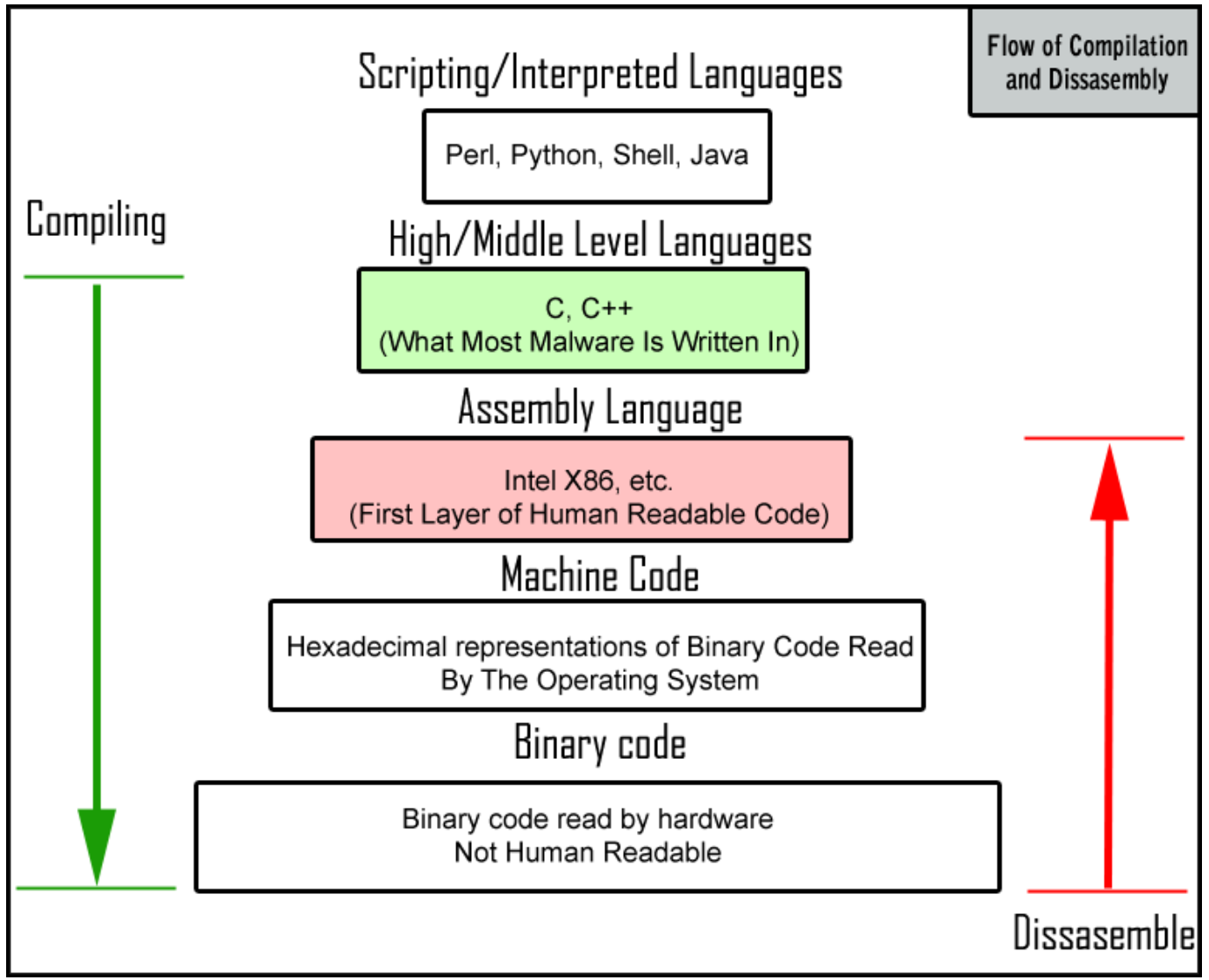


How to get your computer (a larger logic circuits) work?



Assembly Language (汇编语言)

- A low-level programming language
- First layer of human readable code
- Strong correspondence to particular computer architecture's machine code instructions
- Example: ***MIPS***, a reduced instruction set computer (RISC) instruction set architecture (ISA)



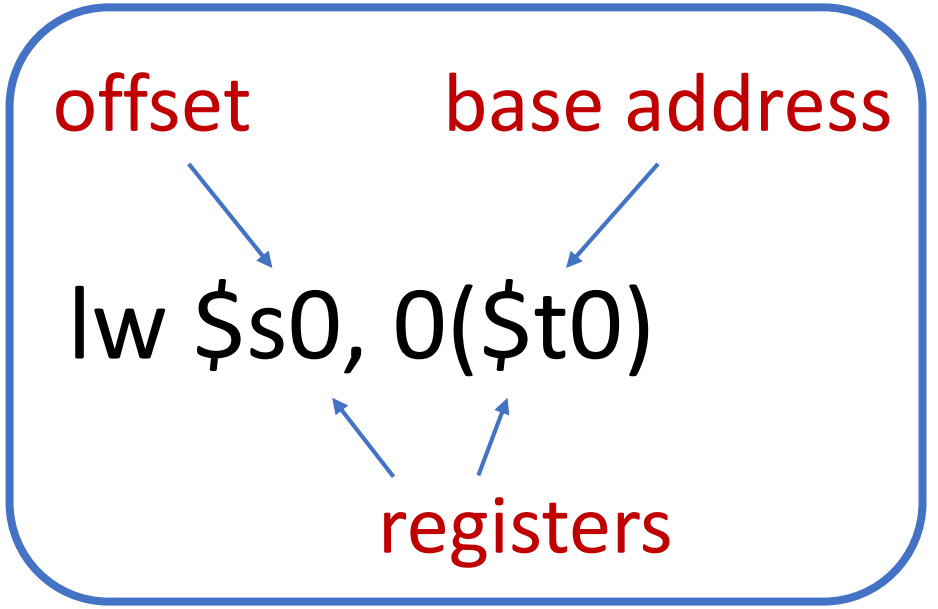
Example: doing addition with MIPS

- In C or Java

```
z = w + y;
```

- With MIPS

```
la $t0, w          # put address of w into $t0
lw $s0, 0($t0)     # put contents of w into $s0
la $t1, y          # put address of y into $t1
lw $s1, 0($t1)     # put contents of y into $s1
add $s2, $s0, $s1      # add w + y, put result in $s2
la $t2, z          # put address of z into $t2
sw $s2, 0($t2)     # put contents of $s2 into z
```



MIPS instruction

- Instruction set architecture (ISA) 指令集架构

Type	31					 0
R	opcode (6)	rs (5)	rt (5)	rd (5)	shamt (5)	funct (6)
I	opcode (6)	rs (5)	rt (5)	immediate (16)		
J	opcode (6)	address (26)				

- R for registers; I for immediate value; J for jump

- Examples