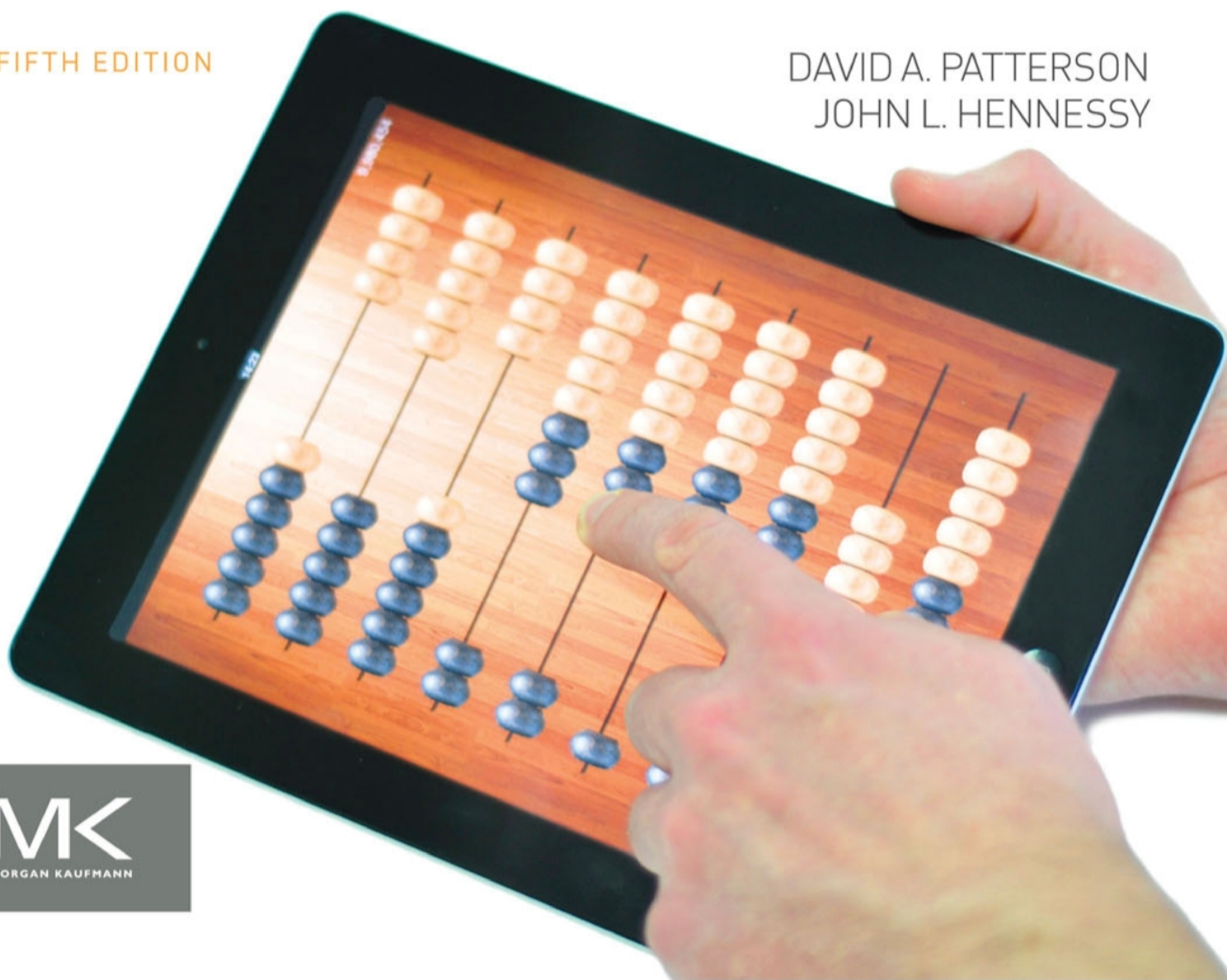


COMPUTER ORGANIZATION AND DESIGN

THE HARDWARE/SOFTWARE INTERFACE

FIFTH EDITION

DAVID A. PATTERSON
JOHN L. HENNESSY



MK
MORGAN KAUFMANN

计算机组成原理



*Civilization advances
by extending the
number of important
operations which we
can perform without
thinking about them.*

Alfred North Whitehead,
An Introduction to Mathematics, 1911

Computer Abstractions and Technology

- 1.1 Introduction** 3
- 1.2 Eight Great Ideas in Computer Architecture** 11
- 1.3 Below Your Program** 13
- 1.4 Under the Covers** 16
- 1.5 Technologies for Building Processors and Memory** 24

第1章 计算机概要与技术

1.1 引言

冯诺依曼体系结构

① 二进制表示信息

指令 = 操作码 + 地址码

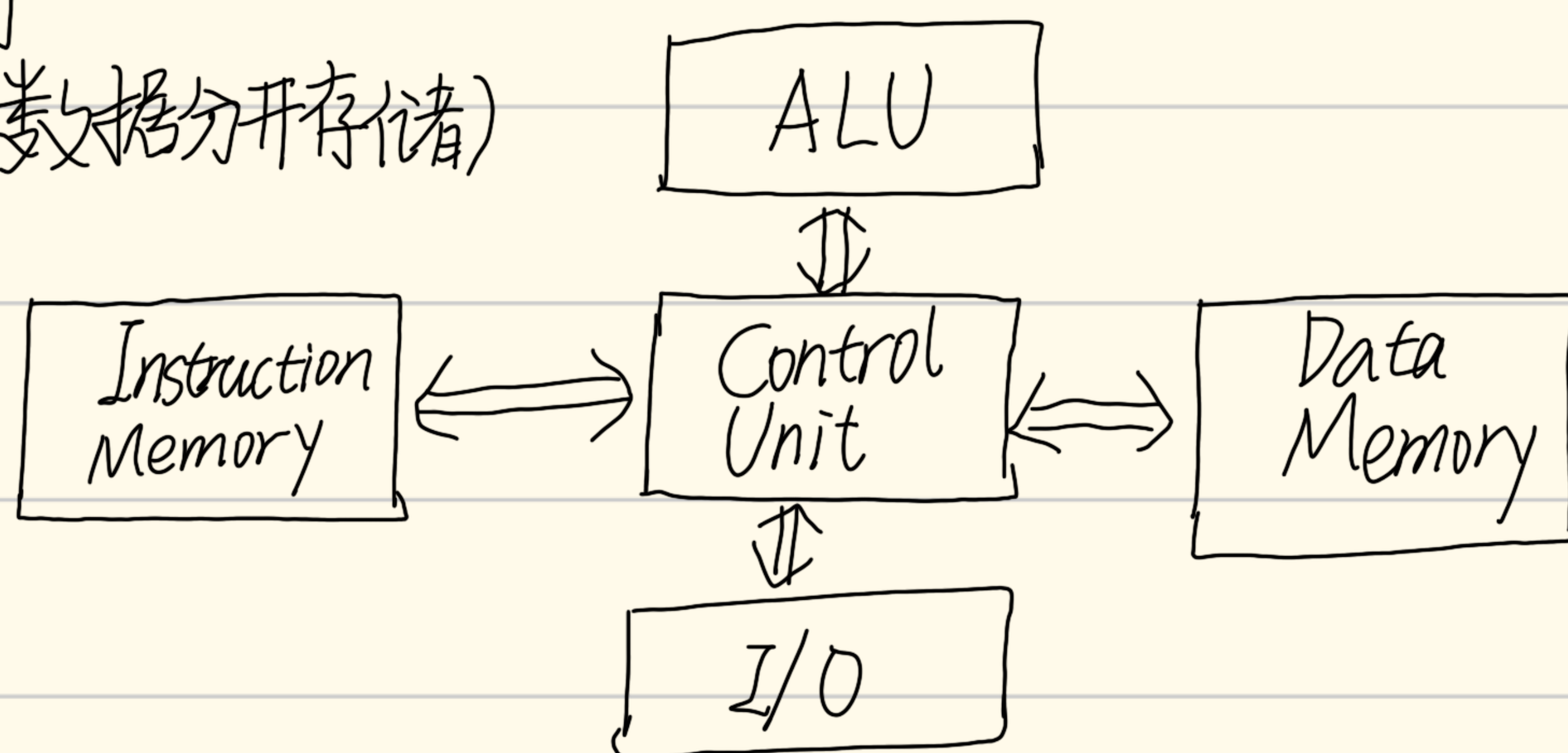
② 存储程序控制

③ 五大基本部件

存储、运算、控制、输入、输出

哈佛结构

(程序与数据分开存储)



计算机的分类

① Personal Computer (PC)

② Server

③ Embedded Computer

④ Personal Mobile Device (PMD)

⑤ Warehouse Scale Computer (WSC)

1.2 系统结构中的7个伟大思想

① 抽象简化

② 大概率事件加速

③ 并行

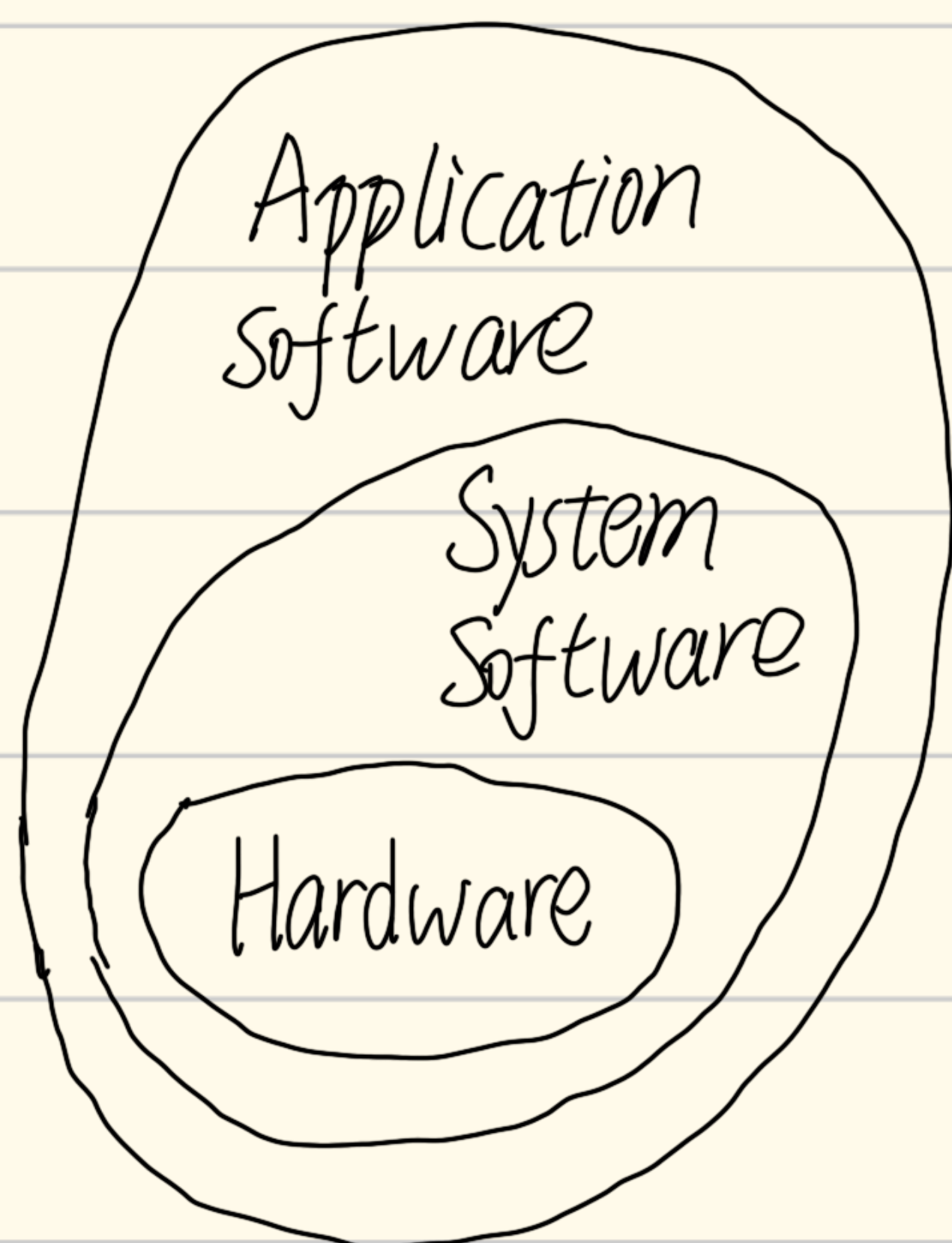
④ 流水线

⑤ 预测

⑥ 存储器分层

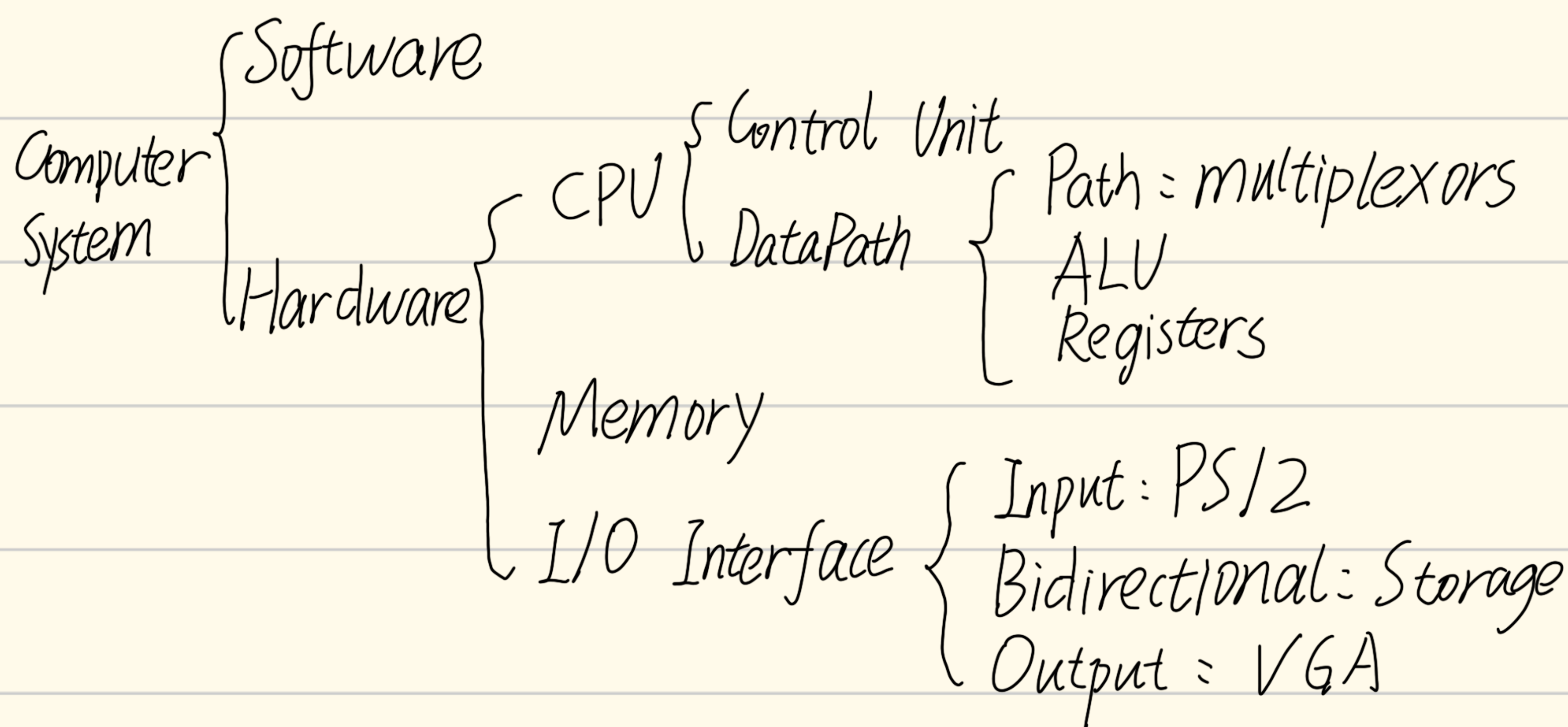
⑦ 冗余

1.3 程序表象之下



机器指令、机器语言、汇编语言、汇编器 (assembler)、
高级语言、编译器 (compiler)

1.4 硬件概念入门



三级存储体系结构:

高速缓存 — 主存储器 — 二级存储器

(1) 主存储器

由CPU编程访问, 由DRAM (动态随机访问存储器) 芯片组成, 称为内存。

(2) 高速缓存

解决CPU与内存之间的速度匹配, 由SRAM芯片组成, 比DRAM更快、集成度低

(3) 二级存储器

磁盘、闪存

计算机硬件系统组织:

总线(Bus) = 能为多个部件分时共享的一组信息发送线路

1.5 处理器和存储器制造技术

电路 + 晶元 $\rightarrow$ chip 连线 $\rightarrow$ 集成电路

1.6 性能

性能的定义

① response time 响应时间 $\rightarrow$ 完成某任务的时间

② throughput 吞吐率 (bandwidth 带宽) $\rightarrow$ 单位时间
完成任务数

性能的度量

① 时钟周期

CPU 主频 $f_c = \frac{1}{T_c}$, T_c 为主时钟

② CPU 执行时间

CPU 执行时间 = 用户 CPU 时间 + 系统 CPU 时间

= CPU 时钟周期数 $\times T_c$

= CPU 时钟周期数 / f_c

CPU 时钟周期数 = 指令平均时钟周期数 (CPI)
 $\times$ 程序的指令数

③ MIPS

$\Rightarrow$ 经典的 CPU 性能公式

$$T_{CPU} = I_N \cdot CPI \cdot T_c$$

注: $1 \text{ GHz} = 10^9 \text{ Hz}$

$1 \text{ MHz} = 10^6 \text{ Hz}$

$1 \text{ ns} = 10^{-9} \text{ s}$

$1 \mu\text{s} = 10^{-6} \text{ s}$

2

*I speak Spanish
to God, Italian to
women, French to
men, and German to
my horse.*

Charles V, Holy Roman Emperor
(1500–1558)

Instructions: Language of the Computer

2.1	Introduction	62
2.2	Operations of the Computer Hardware	63
2.3	Operands of the Computer Hardware	66
2.4	Signed and Unsigned Numbers	73
2.5	Representing Instructions in the Computer	80
2.6	Logical Operations	87
2.7	Instructions for Making Decisions	90

第二章 指令：计算机的语言

2.1 数据信息的表示方法

原码：① 0分为00000000
10000000

② 表示数范围

$$-(2^7-1) \sim 2^7-1$$

即 $|x| < 2^n$, 不能取到端点

补码：① 数0只有一种 00000000
(10000000为补码最小值 -2^7)

② 表示数范围

$$-2^7 \sim 2^7-1$$

负方向可以取到最大值

③ 补码与十进制转换

$$(1010)_{\text{补}} = (1 \times (-2^3) + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)_{10}$$

④ 符号扩展

$$(1100)_2 = (1111\ 1100)_2$$

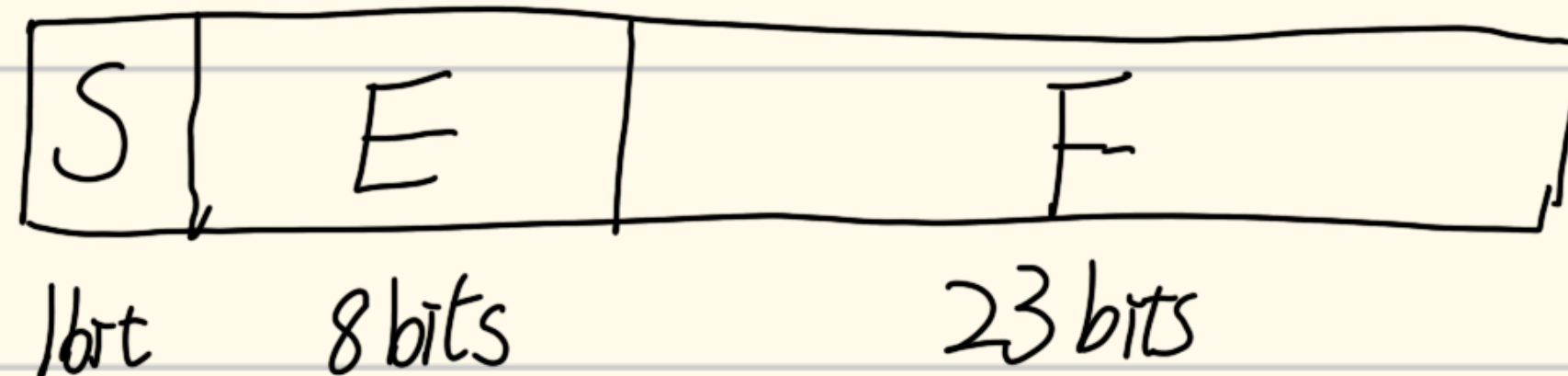
浮点数：① 通用浮点表示形式

$$N = (-1)^S \cdot F \cdot 2^E$$

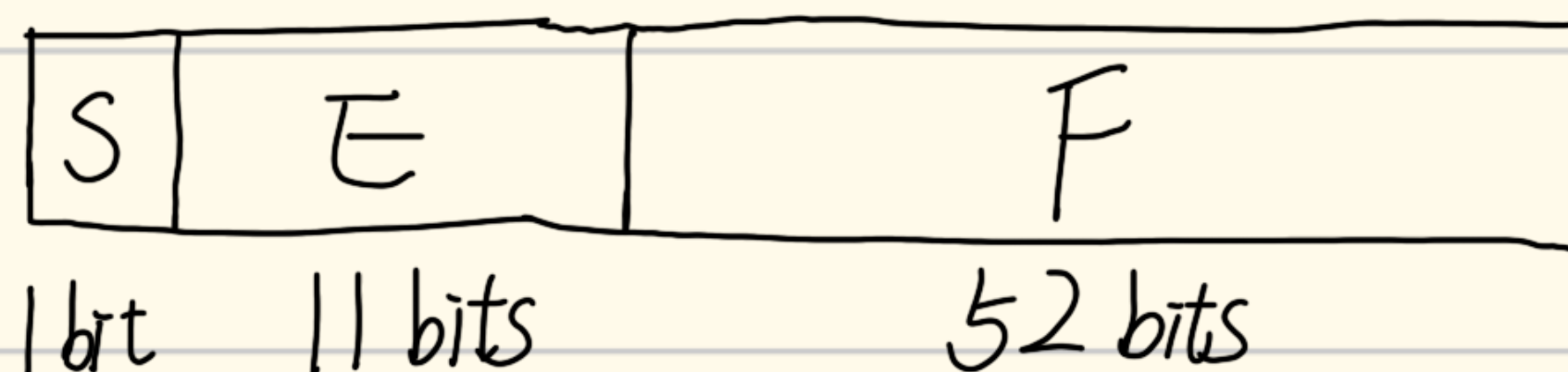
其中S为符号, F为尾数, E为阶码

$$\text{如 } N = (-1)^0 \cdot 0.1101001 \cdot 2^{-3}$$

② 单精度浮点数



双精度浮点数



规格化数: $1.2 \times 10^{-9} \checkmark$

0.12×10^{-9} (X, 不应有前导零)
 12.0×10^{-10} (X, 整数部分只一位)

二进制规格化浮点数:

尾数部分为 1.XXX, 必为 1 点几, 而不是纯小数

偏码 (移码):

浮点加减时, 需要对位标齐, 用移码表示指数

$$[E]_{\text{移}} = E + \text{bias}$$

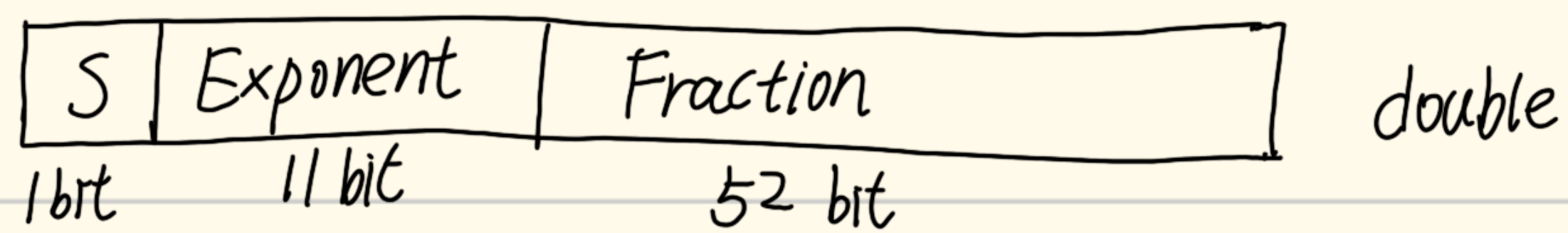
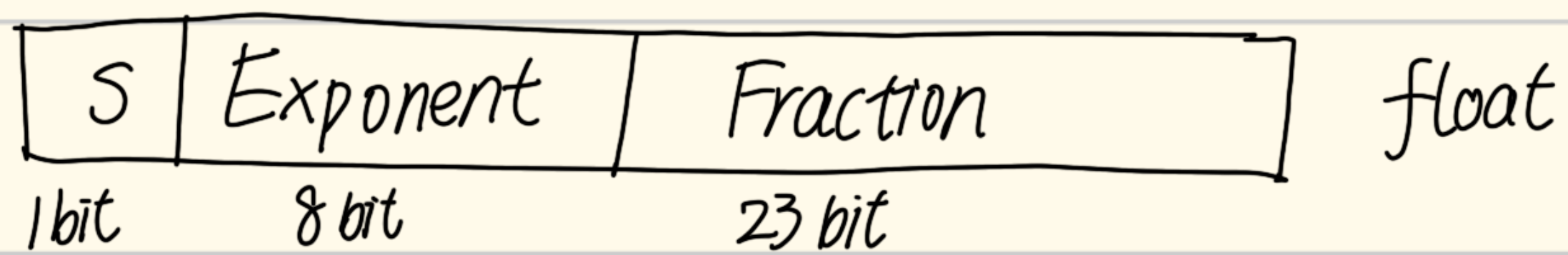
其中 E 为其真值, bias 为偏阶 2^n

IEEE 754 标准:

$$N = (-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

规定偏阶 $\text{Bias} = \begin{cases} 127, & \text{单精度浮点数} \\ 1023, & \text{双精度浮点数} \end{cases}$

在 MIPS 中表示



其中 Fraction 尾数域为纯小数, Exponent 指数域为移码

当 Exponent 全为 0 时, 计算尾数不加前导 1 (用于表示 0), 指数域为 0

单精度		表示对象
指数域	尾数域	
0	0	0
0	非 0	非规格化数
1~254	任何值	浮点数
255	0	无穷
255	非 0	NaN

Exp - Bias 指
 数值范围为
 -126 ~ 127

→ 尾数加前导 1, 指数加 127 偏阶 / 1023 偏阶

$$\text{Fraction} = 1.1 - 1 = (0.1)_2$$

$$\text{Exponent} = -1 + 127 = 126 = (01111110)_2$$

字符的表示 < ASCII 码
 Unicode

2.2 MIPS指令集(RISC)

相关规定:

MIPS算术运算指令操作数必须来自寄存器
32个32位寄存器

名称	寄存器号	用途	调用时是否保存
\$zero	0	常数0	不适用
\$v0 ~ \$v1	2 ~ 3	计算结果和表达式求值	否
\$a0 ~ \$a3	4 ~ 7	参数	否
\$t0 ~ \$t7	8 ~ 15	临时变量	否
\$s0 ~ \$s7	16 ~ 23	保存的寄存器	是 ✓
\$t8 ~ \$t9	24 ~ 25	更多临时变量	否
\$gp	28	全局指针	是 ✓
\$sp	29	栈指针	是 ✓
\$fp	30	帧指针	是 ✓
\$ra	31	返回地址	是 ✓

寄存器操作数

按字节编址, 一个字占4个字节



寄存器 ← 数据传送指令 → 寄存器
lw, sw

例如: 存数 array[8], 基址存于 \$s0

sw \$t0, 32(\$s0)

取数 array[i], 基址存于 \$s0, i 存于 \$t1

add \$t0, \$t1, \$s0 # 偏移量 = 4个字节 × i
add \$t0, \$t0, \$t0
add \$t0, \$t0, \$s0 # 地址 = 偏移量 + 基址
lw \$t1, 0(\$t0)

立即数操作数

addi \$s0, \$s0, 4 (没有 subi)

MIPS指令的表示

指令长度均为32位 → R型指令op全为0

R-format | op | rs | rt | rd | shamt | funct | 算术运算, 逻辑运算
6 5 5 5 5 6

I-format | op | rs | rt | offset | 立即数算术运算, 数据传输, 分支转移
6 5 5 16

J-format | op | target address(word) | 无条件跳转
6 26

注: add \$t0, \$s0, \$s1
rs rt rd
source

lw/sw \$t0, 100(\$s0)
rs rt offset
!!! (基址寄存器)

逻辑操作

逻辑左移 ($\times 2^k$)、逻辑右移 ($\div 2^k$) [R型]

sll \$t0, \$s0, 4 srl

rs rt rd shamt

按位与、按位或、按位取反 (NOR with 0) [R型]

and or nor \$t0, \$t1, \$zero (取反) no

立即数与、立即数或 [I型]

andi ori

决策指令

相等分支、不等分支 [I型]

beq bne

无条件跳转 [J型]

j

例如: while (array[i] == k) i: \$s3 j: \$s4 k: \$s5
 i = i + j; array: \$s6

Loop: sll \$t0, \$s3, 2
 add \$t0, \$t0, \$s6
 lw \$t1, 0(\$t0)
 bne \$t1, \$s5, Exit
 add \$s3, \$s3, \$s4
 j Loop

Exit:

小于则置位 [R型]、立即数小于则置位 [I型]、无符号整数小于则置位 [R型]

slt \$t0, \$s0, \$s1 slti

(\$t0 = 1 if \$s0 < \$s1)

sltu

寄存器跳转 [R型]

jr \$register

op	rs	rt	rd	shamt	funct
0	rs	0	0	0	8

例如:

```
switch (k) {  
  case 0: f = i + j; break;  
  case 1: f = g + h; break;  
  case 2: f = g - h; break;  
  case 3: f = i - j; break;  
}
```

```
slt $t3, $s5, $zero    # test if k < 0  
bne $t3, $zero, Exit    # if k < 0, go to Exit  
slt $t3, $s5, $t2    # test if k < 4  
beq $t3, $zero, Exit    # if k >= 4, go to Exit  
add $t1, $s5, $s5    # temp reg $t1 = 2 * k (0 <= k <= 3)  
add $t1, $t1, $t1    # temp reg $t1 = 4 * k  
add $t1, $t1, $t4    # $t1 = address of JumpTable[k]  
lw $t0, 0($t1)    # temp reg $t0 = JumpTable[k]  
jr $t0    # jump based on register $t0
```

<

jump address table

\$t1 = \$t4 + 4 * k:

L0: address

L1: address

L2: address

L3: address

```
L0: add $s0, $s3, $s4    # k = 0 so f gets i + j  
     j Exit    # end of this case so go to Exit  
L1: add $s0, $s1, $s2    # k = 1 so f gets g + h  
     j Exit    # end of this case so go to Exit  
L2: sub $s0, $s1, $s2    # k = 2 so f gets g - h  
     j Exit    # end of this case so go to Exit  
L3: sub $s0, $s3, $s4    # k = 3 so f gets i - j  
Exit:    # end of switch statement
```


基本块：顺序执行、单入口、单出口

过程

过程调用寄存器： $\$a_0 - \a_3 传参数
 $\$v_0 - \v_1 返回值
 $\$ra$ 返回起始点

过程调用指令

PC (程序计数器)

保存当前运行指令的地址

$$PC = PC + 4$$

调用者：跳转链接指令 [J型]

`jal ProcessAddress`

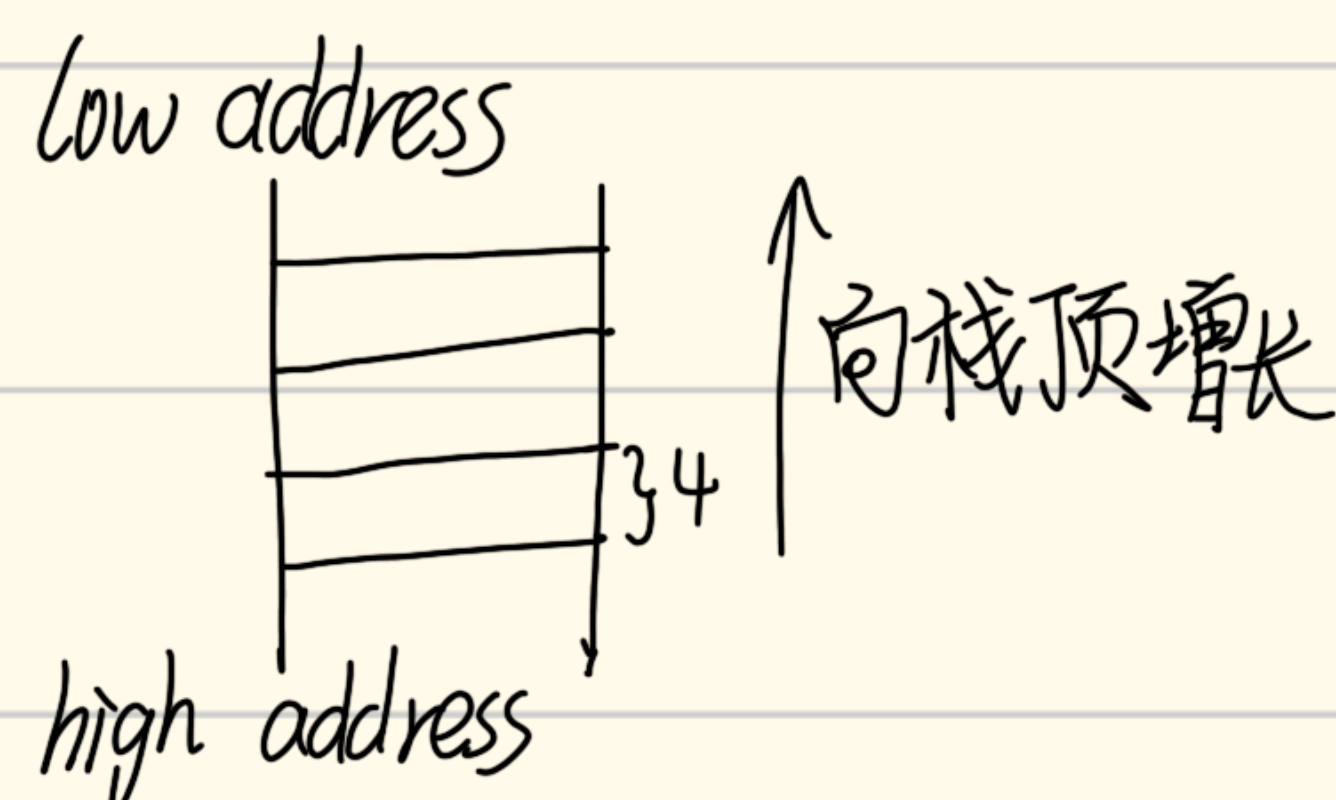
(同时执行 $\$ra \leftarrow PC + 4$)

被调用：寄存器跳转 [R型]

`jr $ra`

现场的保存与恢复：栈

入栈
`addi $sp, $sp, -4`
`sw $s0, 0($sp)`
出栈
`lw $s0, 0($sp)`
`addi $sp, $sp, 4`



需保留的内容

$\$s_0 \sim \s_7 、嵌套过程中 $\$ra$ 、 $\$sp$ (先减后必加)
还可能需保留 $\$a_0 \sim \a_3

嵌套过程

例如

```
int fact ( int n )
{
    if ( n < 1 ) return ( 1 );
    else return ( n * fact ( n - 1 ) );
}

fact: addi $sp, $sp, -8           # adjust stack for 2 items
      sw  $ra, 4($sp)           # save the return address
      sw  $a0, 0($sp)           # save the argument n

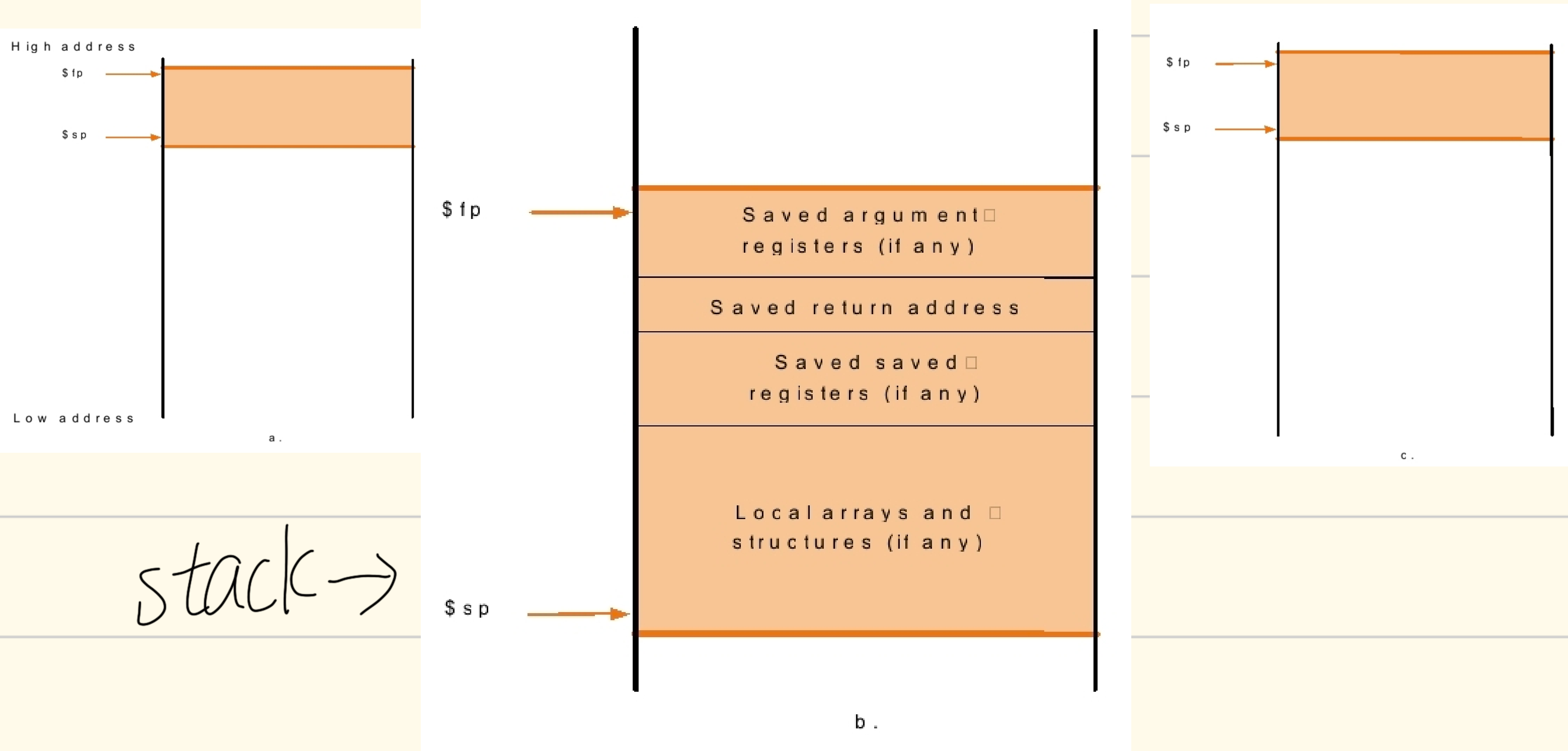
      slti $t0, $a0, 1           # test for n < 1
      beq  $t0, $zero, L1        # if n >= 1, go to L1 (else)
      addi $v0, $zero, 1         # return if n < 1
      addi $sp, $sp, 8           # Recover $sp (Why not recover $ra and $a0 ?)
      jr   $ra                  # return to after jal

L1: addi $a0, $a0, -1            # n >= 1: argument gets ( n - 1 )
      jal  fact                 # call fact with ( n - 1 )
      lw  $a0, 0($sp)           # return from jal: restore argument n
      lw  $ra, 4($sp)           # restore the return address
      addi $sp, $sp, 8           # adjust stack pointer to pop 2 items
      mul $v0, $a0, $v0         # return n * fact ( n - 1 )
      jr   $ra                  # return to the caller
```


硬件对过程的支持

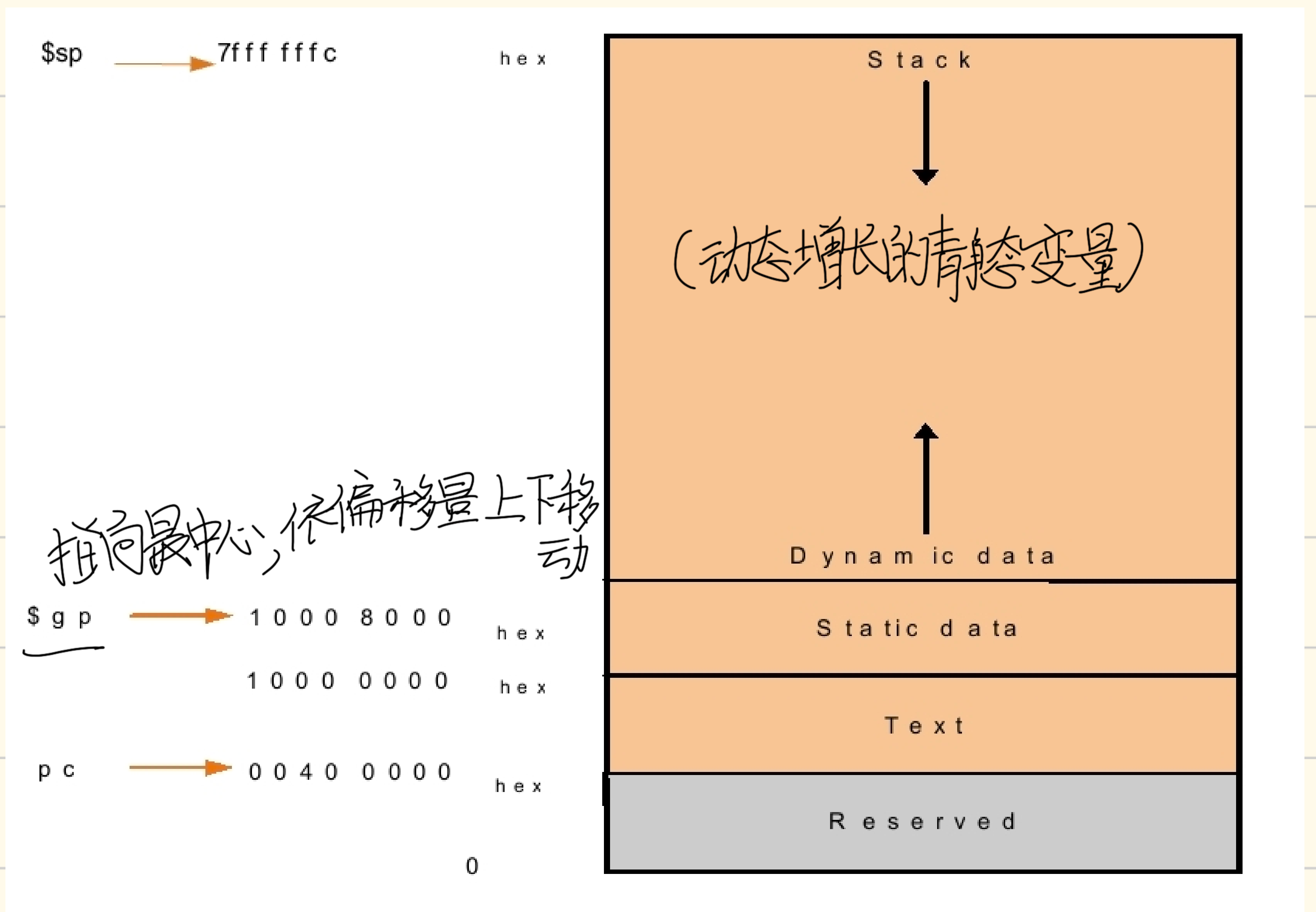
(语言存储域: 动态 automatic / 静态 static)

动态 ← 过程帧和帧指针: 过程中的寄存器和局部变量(存于栈中) → \$fp
静态 ← 全局指针: 指向静态数据区 → \$gp



stack →

heap ↴



MIPS中32位立即数

lui指令: 装入寄存器高16位, 低16位置0

Instruction

001111	00000	01000	0000 0000 1111 1111
--------	-------	-------	---------------------

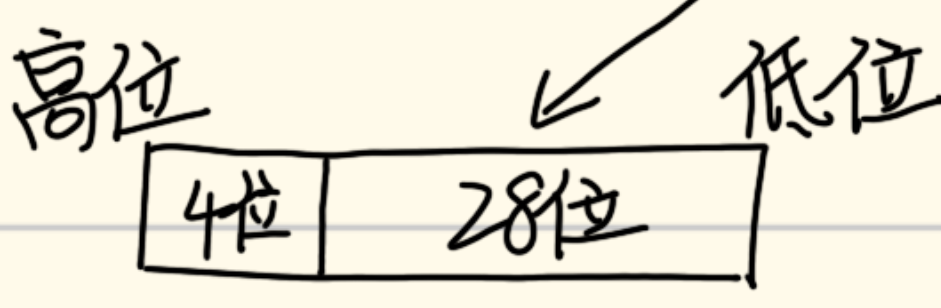
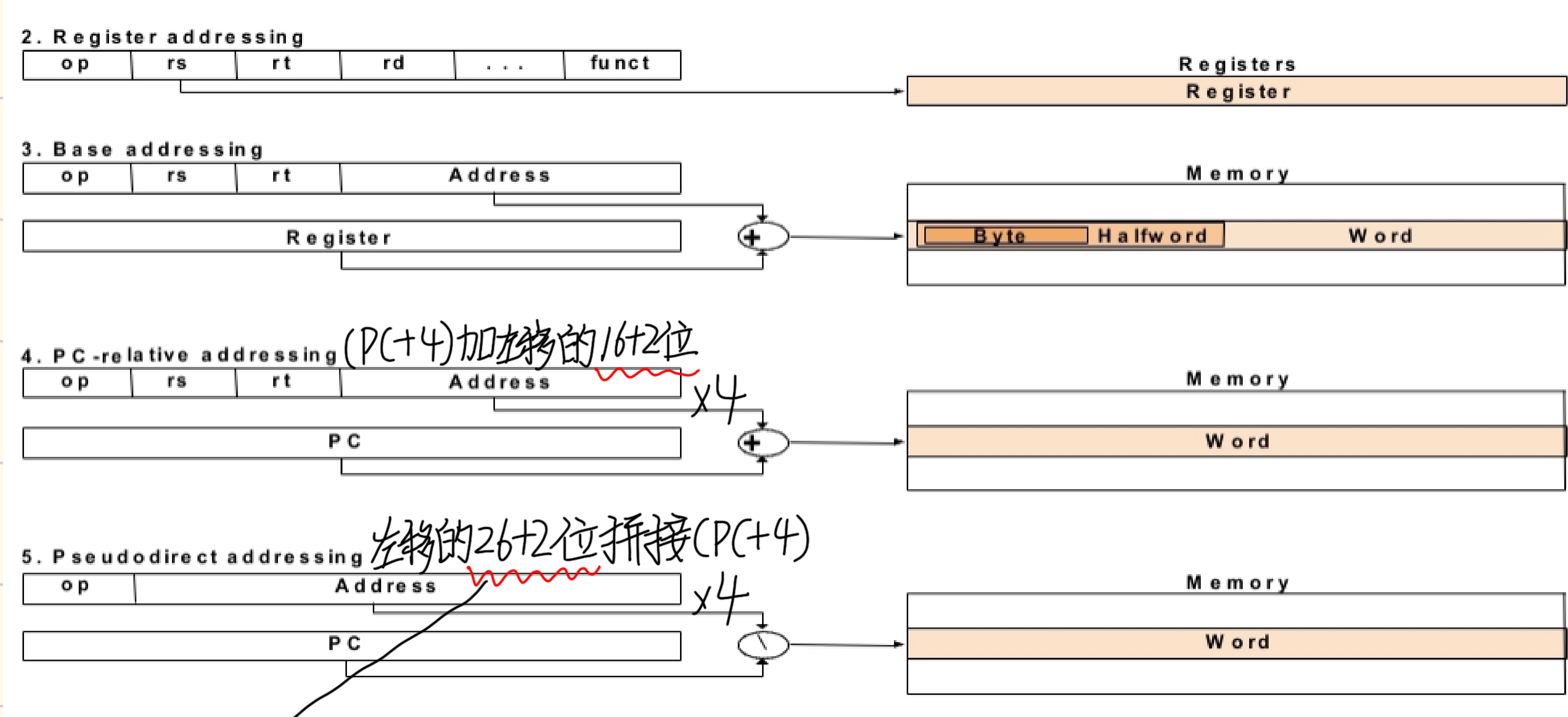
Register

0000 0000 1111 1111	Filling with "0" 0000 0000 0000 0000
---------------------	---

lui \$s0, 61 # 61 decimal = 0000 0000 0011 1101 binary
 (The value of \$s0 afterward is: 0000 0000 0011 1101 0000 0000 0000 0000)
ori \$s0, \$s0, 2304 # 2304 decimal = 0000 1001 0000 0000 binary
 (The value of \$s0 afterward is: 0000 0000 0011 1101 0000 1001 0000 0000)

- | | |
|----------|-----------------------------------|
| ① 立即数寻址 | <code>addi \$S0, \$S0, 4</code> |
| ② 寄存器寻址 | <code>add \$S0, \$S1, \$S2</code> |
| ③ 基址寻址 | <code>lw \$S1, 0(\$S0)</code> |
| ④ PC相对寻址 | <code>beq \$S0, \$S1, L1</code> |
| ⑤ 伪直接寻址 | <code>j Address</code> |

op	rs	rt	Immediate
----	----	----	-----------



机器语言解码

0000 0000 1010 1111 1000 0000 0010 0000

op → R型

funct → add

op	rs	rt	rd	shamt	funct
000000	00101	01111	10000	00000	100000

↓ ↓ ↓

\$a1 \$t7 \$s0

⇒ add rd, rs, rt

伪指令

move \$t0, \$t1

≡ add \$t0, \$zero, \$t1

blt ≡ slt + beq 小于则跳转

li rdest, imm

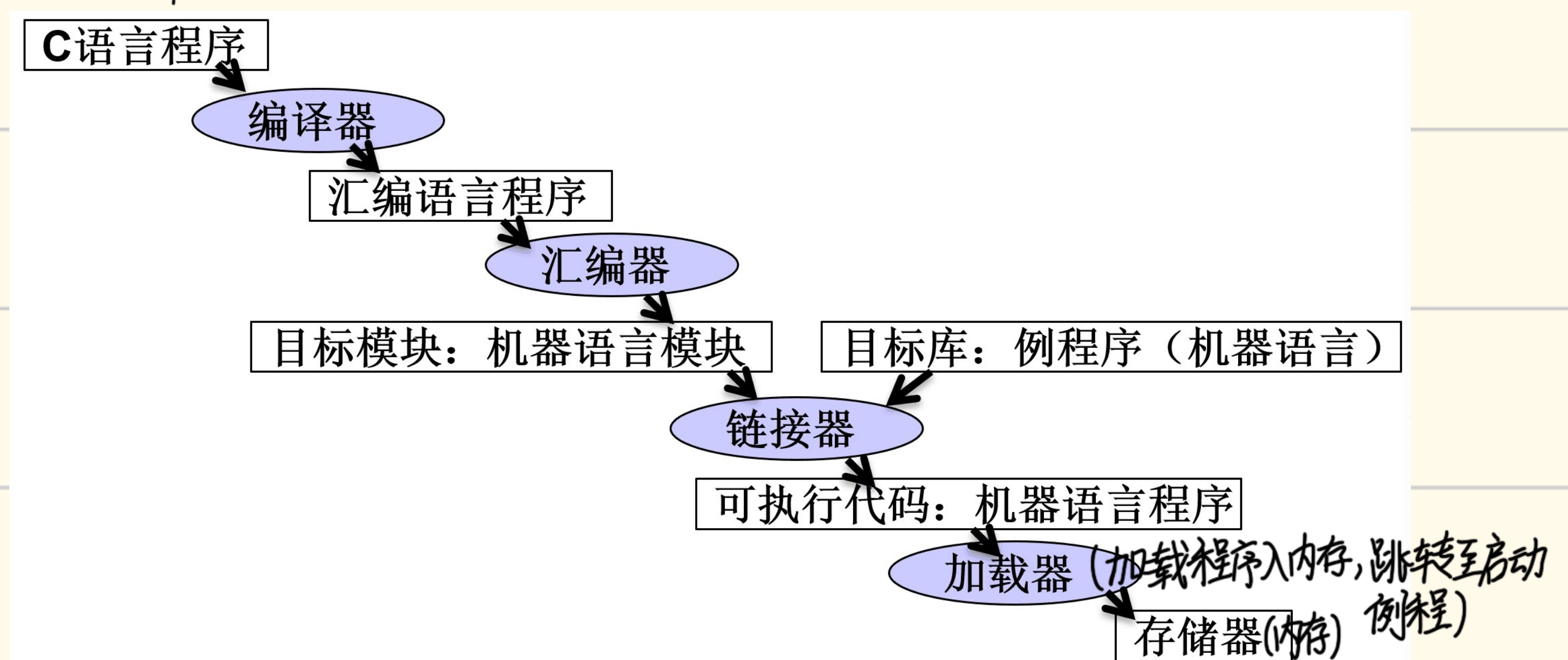
≡ lui \$at, imm高半部

ori rdest, \$at, imm低半部

(\$at为汇编器保留, \$k为操作系统保留)

程序翻译

编译 → 汇编 → 链接 → 加载



目标文件 6 个部分:

文件头 (描述文件大小、位置)

代码段

静态数据段

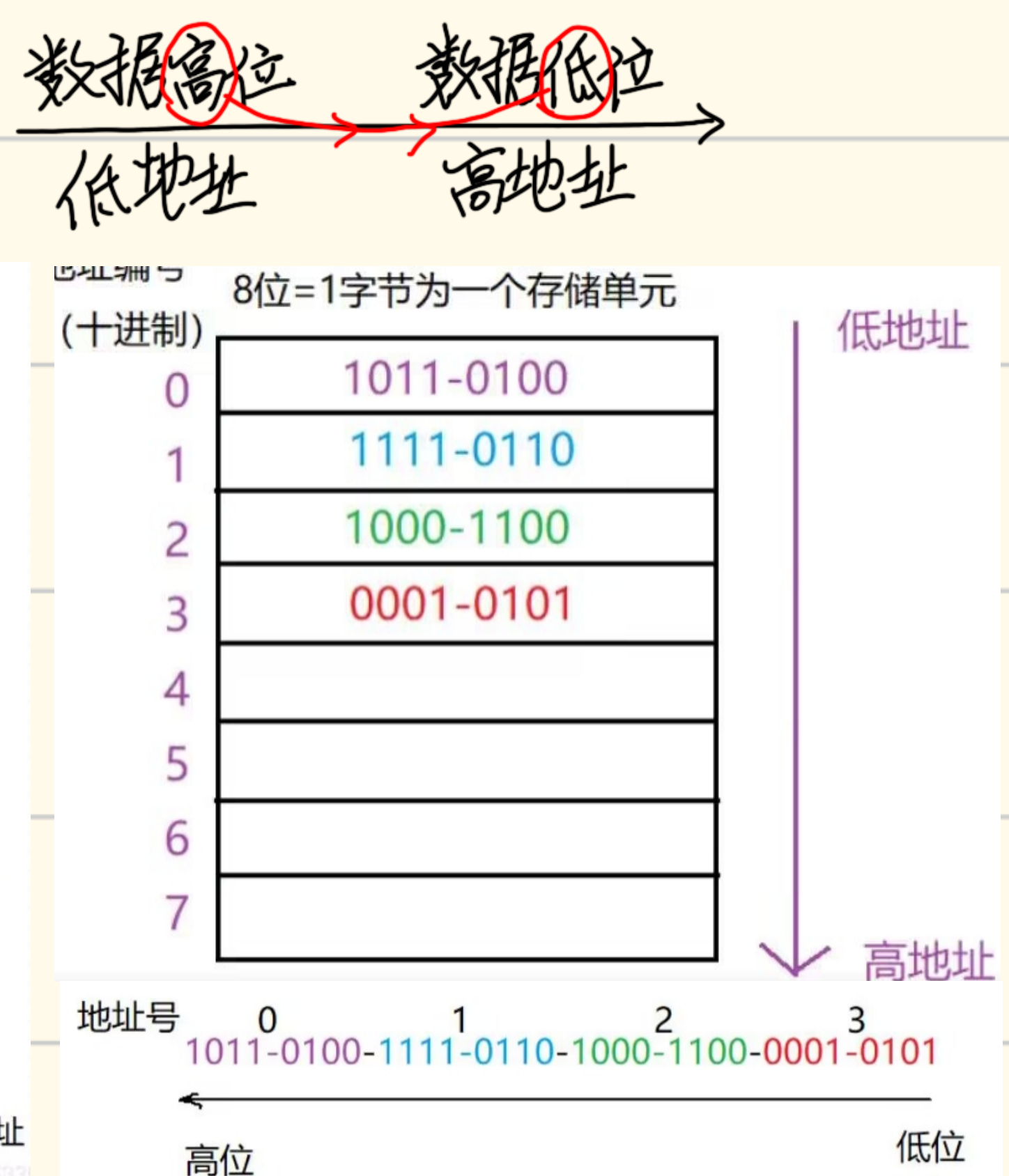
重定位信息

符号表

调试信息

补充: 小端编址

大端编址



3

*Numerical precision
is the very soul of
science.*

Sir D'arcy Wentworth Thompson
On Growth and Form, 1917

Arithmetic for Computers

3.1	Introduction	178
3.2	Addition and Subtraction	178
3.3	Multiplication	183
3.4	Division	189
3.5	Floating Point	196
3.6	Parallelism and Computer Arithmetic: Subword Parallelism	222
3.7	Real Stuff: Streaming SIMD Extensions and Advanced Vector Extensions in x86	224

第3章 计算机的算术运算

3.2 定点加减运算

正溢
负溢

溢出的判别

① 操作数数符 S_A, S_B 结果数符 S_f

$$\text{溢出} = \overline{S_A} \cdot \overline{S_B} \cdot S_f + S_A \cdot S_B \cdot \overline{S_f}$$

② 符号位进位 C_f 最高有效数位进位 C

$$\text{溢出} = C_f \oplus C$$

③ 双符号位

00 — 结果为正，无溢出 **01** — 结果正溢

10 — 结果负溢

11 — 结果为负，无溢出

eg.

$9+3=12$ $\begin{array}{r} 001001 \\ + 000011 \\ \hline 001100 \end{array}$	$11+7=18$ $\begin{array}{r} 001011 \\ + 000111 \\ \hline 010010 \end{array}$
$(-9)+(-3)=-12$ $\begin{array}{r} 110111 \\ + 111101 \\ \hline 110100 \end{array}$	$(-11)+(-7)=-18$ $\begin{array}{r} 110101 \\ + 111001 \\ \hline 101110 \end{array}$

溢出的处理

由ALU硬件检测，产生中断，指令地址保存至EPC，跳转至异常处理程序

异常的产生：

有符号算术运算指令溢出时会产生异常

加法 (add)、立即数加法 (addi)、减法 (sub)

无符号算术运算指令溢出时不会产生异常

无符号加法 (addu)、立即数无符号加法 (addiu)、无符号减法 (subu)

(注：addiu也会高16位填充)

MIPS中只有整字32位运算操作指令

但有字节8位、半字数据传送指令

3.4 乘法运算

N次移位和累加

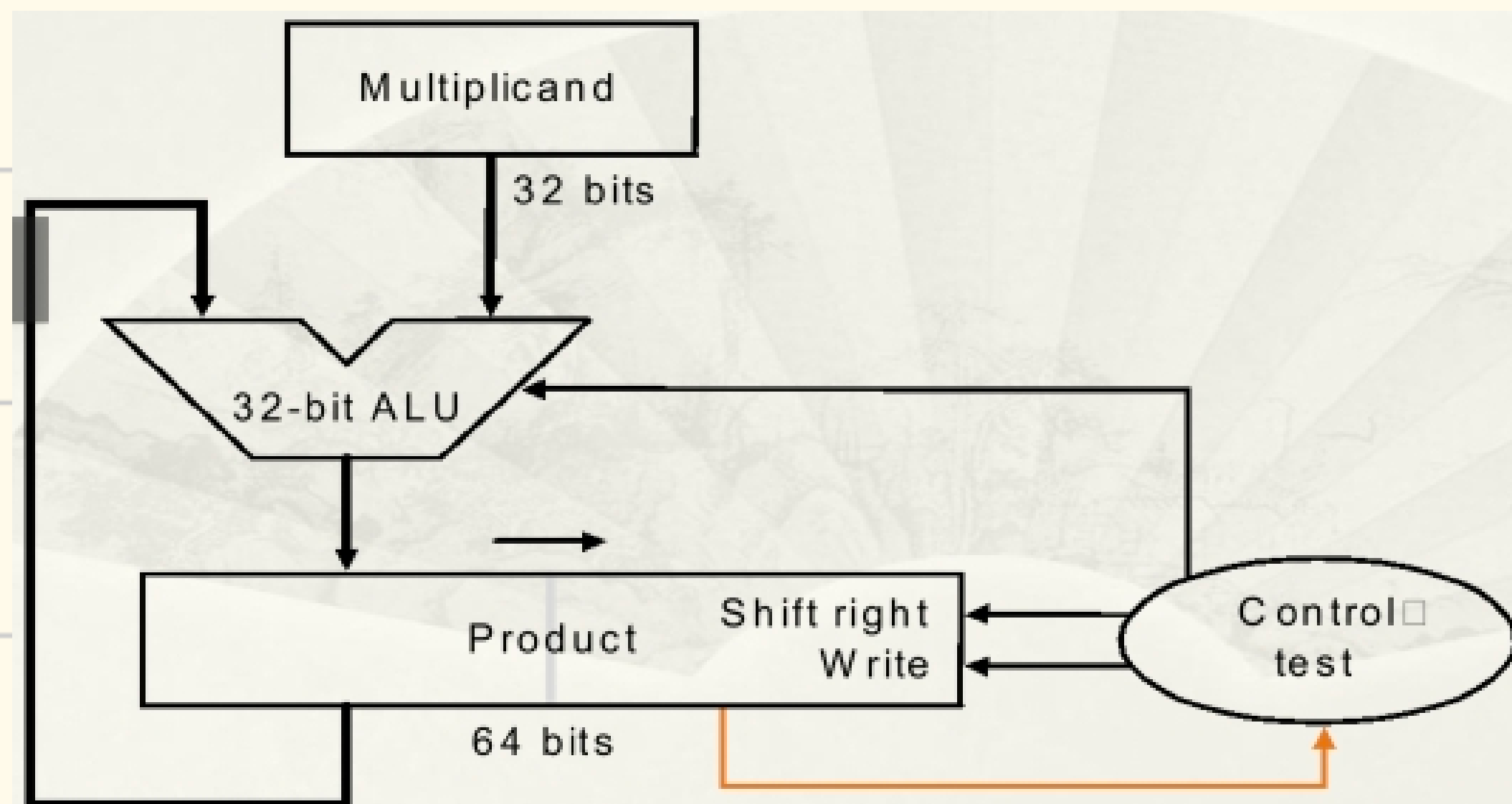
乘积右移法

用相加后的部分积右移,代替被乘数左移

multiplicand, ALU: 32 bits

product: 64 bits (实际65位,多一位暂存进位)

!!! 加法只在 product 高 32 位进行,右移在整个 64 位 product 进行



eg. 0001 x 0111

```

00000111
0001
-----
00010111
移位 00001011
0001
-----
00011011
移位 00001101
0001
-----
00011101
  
```

```

移位 00001110
0000
-----
00001110
移位 00000111
⇒ 积为 00000111
  
```

有符号乘法

先做无符号乘法,再异或符号位

32位乘法算法[原码]

$A \leftarrow 0$ (乘积高位) $B \leftarrow |X|$ (被乘数) $C \leftarrow |Y|$ (乘数、乘积低位)

循环 32 次:

if 最后一次:

$A, C \leftarrow A + 0$ 右移一位

else:

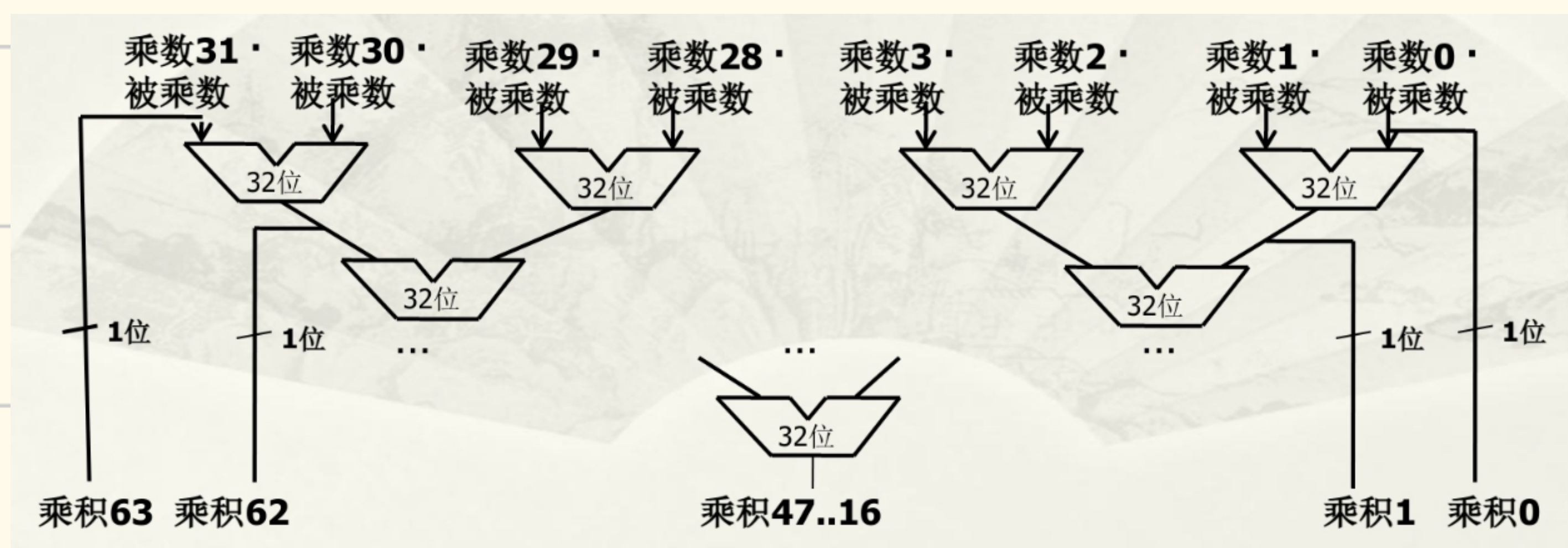
$A, C \leftarrow A + B$ 右移一位

$SA \leftarrow S_x \oplus S_y$

eg. 0110 x (11011)

$$\begin{array}{r}
 00.1101 \\
 \times 00.1011 \\
 \hline
 00.0000 \ 1011 \\
 + 00.1101 \\
 \hline
 00.1101 \ 1011 \\
 \text{移位, } 00.0110 \ 1101 \\
 + 00.1101 \\
 \hline
 01.0011 \ 1101 \\
 \text{移位, } 00.1001 \ 1110 \\
 \text{移位, } 00.0100 \ 1111 \\
 + 00.1101 \\
 \hline
 01.0001 \ 1111 \\
 \text{移位, } 00.1000 \ 1111 \\
 \text{符号位: } (+) \times (-) \rightarrow (-) \\
 \Rightarrow \text{积为 } 1.0001 \ 1111
 \end{array}$$

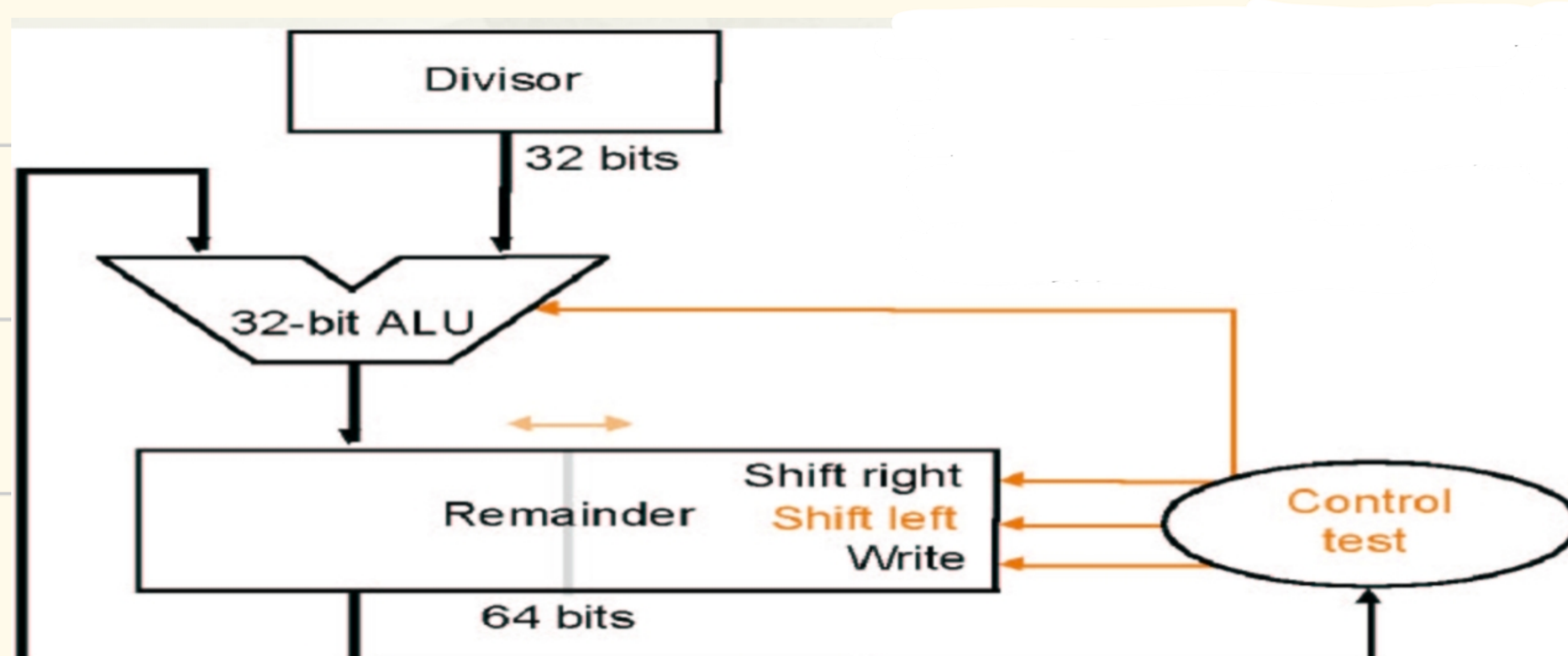
并行乘法



3.5

除法运算

改进的除法



初始值 $divisor \leftarrow$ 除数 $remainder \leftarrow 0$, 被除数

循环32次:

32位 余数 | 商 | 32位

$remainder$ 左移一位

余数 $\leftarrow$ 余数 - 除数

if 余数 < 0 :

恢复余数, 商最后一位置0

else:

商最后一位置1

eg. $0111/0010$

初始值 $00000111 \div 0010$

左移 00001110

-0010

11101110

恢复余数, 商0 00001110

左移 00011100

-0010

11111100

恢复余数, 商0 00011100

左移 00111000

-0010

00011000

够减, 商1 00011001

左移 00110010

-0010

00010010

够减, 商1 00010011

$\Rightarrow 0111/0010 = 0011 \dots 0001$

3.6

浮点运算

IEEE 754:

$$N = (-1)^S \times (1 + \text{Fraction}) \times 2^{\text{Exp} - \text{Bias}}$$

单精度 1-8-23

Bias = 127

双精度 1-11-52

Bias = 1023

溢出： $\begin{cases} \text{上溢} \\ \text{下溢} \end{cases}$

舍入：保护位、舍入位、粘贴位
尾数末多保留第1、2位 舍入标志位

①向上舍入 ②向下舍入 ③截断舍入

④ (default) 最近舍入 $\begin{cases} \leq 4 \rightarrow \text{舍} \\ \geq 6 \rightarrow \text{入} \\ = 5 \begin{cases} \text{若粘贴位}=1, \text{则入} \\ \text{否则向最近的偶数舍入} \end{cases} \end{cases}$

浮点加法

①对阶

小阶向大阶对齐：小阶阶码增大，尾数右移

②尾数相加

③规格化：右移增大阶码，左移减小阶码
检查溢出

④舍入尾数（多保留的2位），保留适当位数

⑤再次判断是否规格化，跳转③或结束

eg. $0.5 + (-0.4375)$

$$0.5 = 1 \times 2^{-1}$$

$$-0.4375 = -1.110 \times 2^{-2}$$

$$\text{对阶, } -1.110 \times 2^{-2} = -0.111 \times 2^{-1}$$

$$\text{尾数相加, } 1 - 0.111 = 0.001$$

$$\text{规格化, } 0.001 \times 2^{-1} = 1 \times 2^{-4}$$

$$\Rightarrow 0.5 + (-0.4375) = 0.0625$$

浮点乘法

①指数相加，再减偏阶 ($\text{Exp1} + \text{Exp2} - \text{Bias}$)

②尾数相乘

③规格化乘积
检查溢出

④舍入尾数（多保留的2位），保留适当位数

⑤再次判断是否规格化，跳转③或⑥

⑥设置积的正负符号

eg. $0.5 \times (-0.4375)$

$$\text{即 } (1.000 \times 2^{-1}) \times (-1.110 \times 2^{-2})$$

$$\text{指数相加 } (-1) + (-2) = (-3)$$

$$\text{或 } (-1 + 127) + (-2 + 127) - 127 = 124$$

$$\text{尾数相乘 } 1.110 \times 1.000 = 1.110000$$

$$\text{舍入 } 1.110000 \times 2^{-3} = 1.110 \times 2^{-3}$$

$$\text{设置符号 } -1.110 \times 2^{-3}$$

$$\Rightarrow 0.5 \times (-0.4375) = -0.21875$$

4

In a major matter, no details are small.

French Proverb

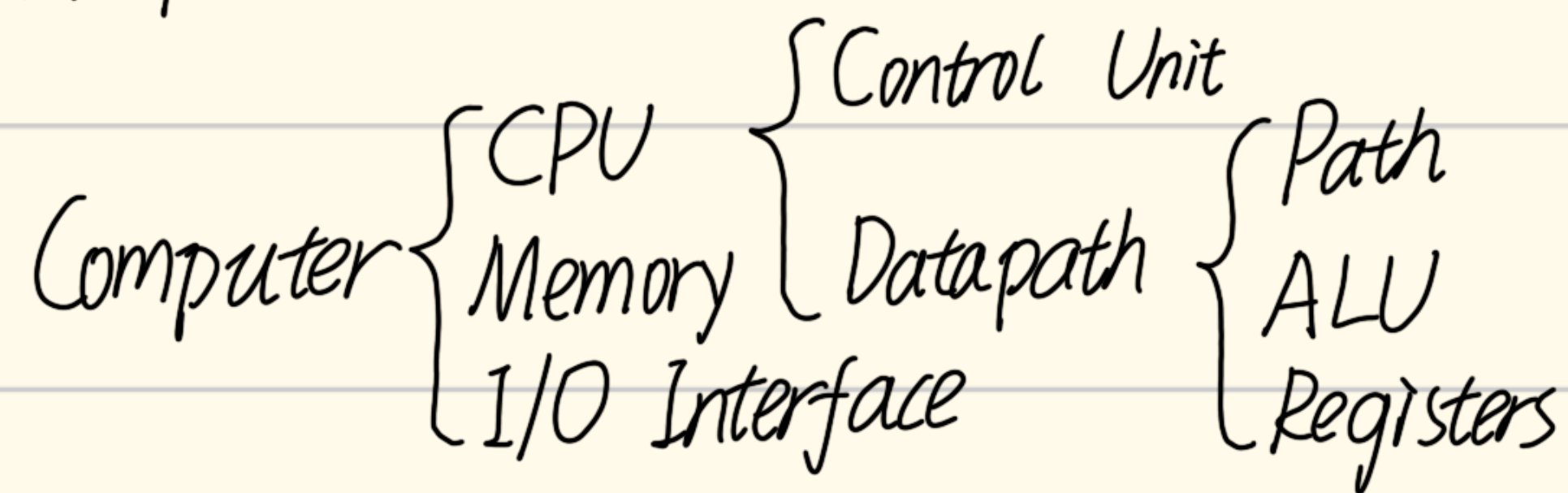
The Processor

4.1	Introduction	244
4.2	Logic Design Conventions	248
4.3	Building a Datapath	251
4.4	A Simple Implementation Scheme	259
4.5	An Overview of Pipelining	272
4.6	Pipelined Datapath and Control	286
4.7	Data Hazards: Forwarding versus Stalling	303
4.8	Control Hazards	316
4.9	Exceptions	325
4.10	Parallelism via Instructions	332

第4章 处理器

4.1 引言

计算机的组成



CPU的基本功能

- ① 程序控制 ② 操作控制 ③ 时间控制 ④ 数据加工

4.2 逻辑设计的一般方法

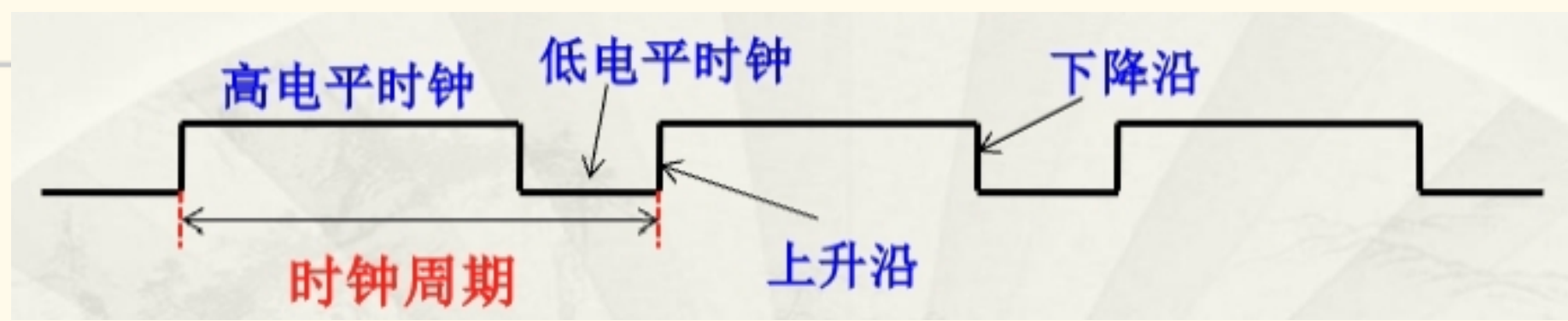
主要逻辑单元

- ① 组合单元 ② 状态单元

时序系统

- ① 时序发生器
- 一个脉冲源
 - 一组计数分频逻辑

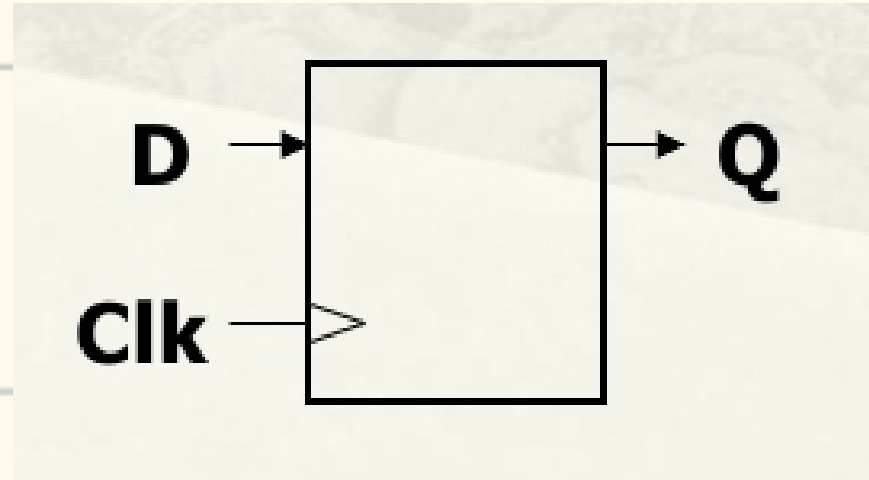
② 时钟



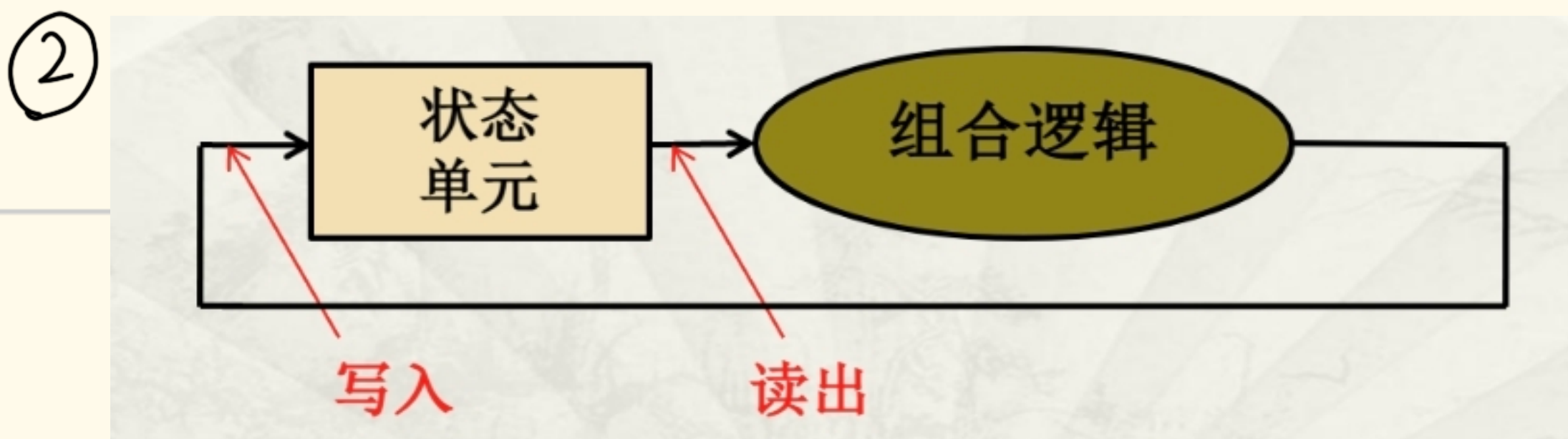
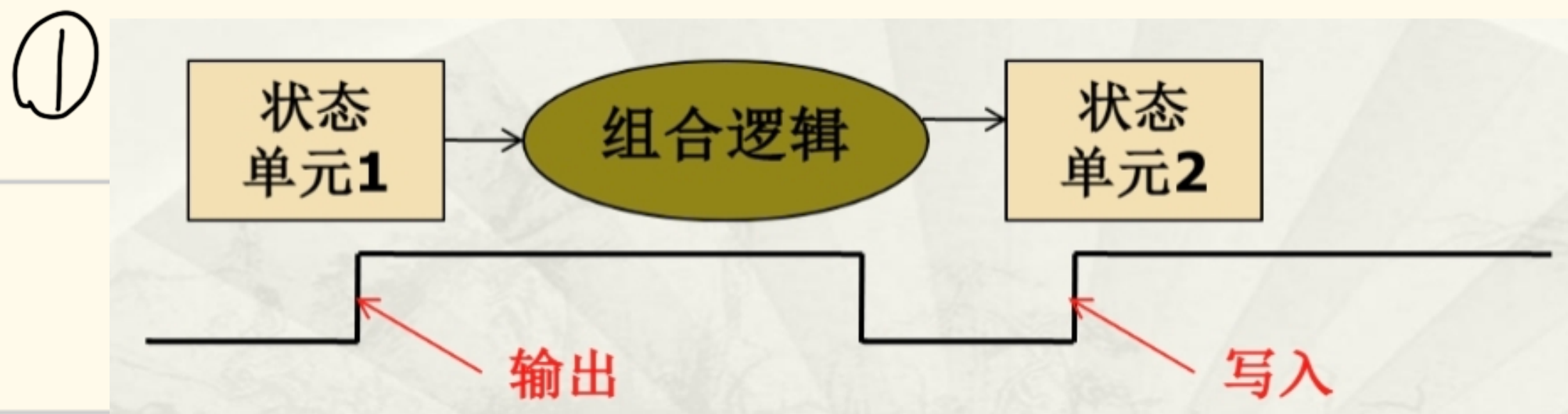
③ 时钟(同步)方法

同步控制 (非异步控制)

④ 边沿触发时钟

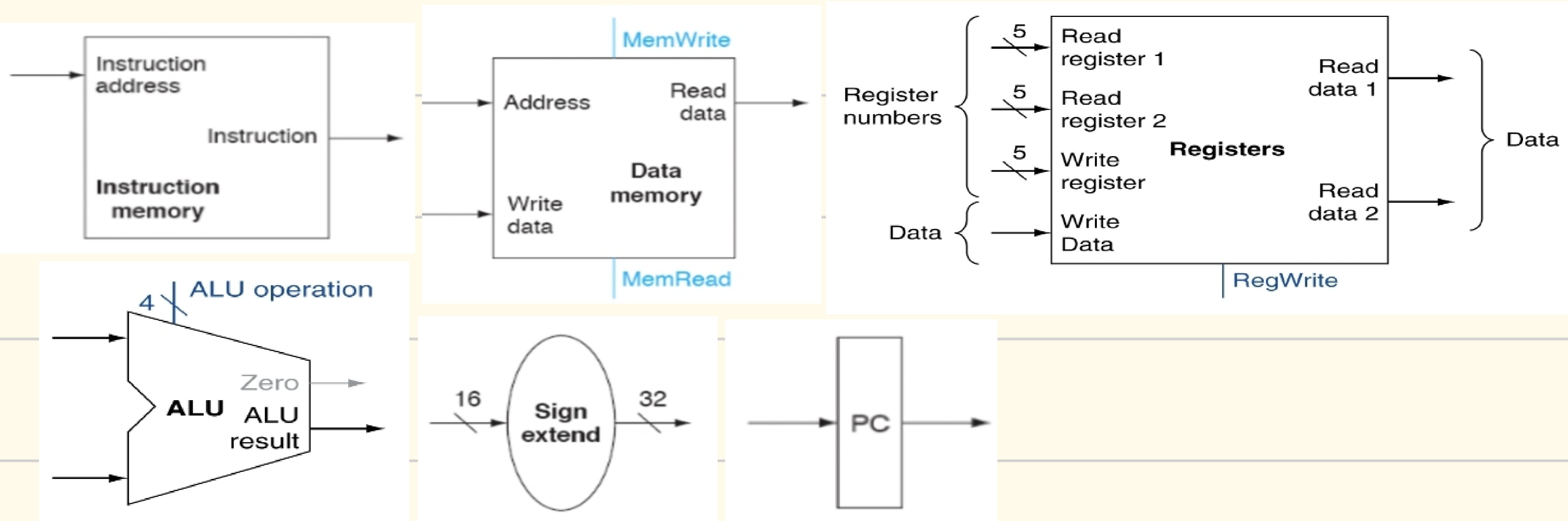


同步时序逻辑设计



4.3 建立数据通路

主要数据通路部件



MIPS指令执行过程

① 取指令

1° 从指令寄存器中取出指令

2° 修改PC, 指向下一条指令

② 译码 & 读操作数

1° 指令译码

2° 读入寄存器中的操作数 (即使后续无需使用)

③ 执行: 控制 ALU 实现相应操作

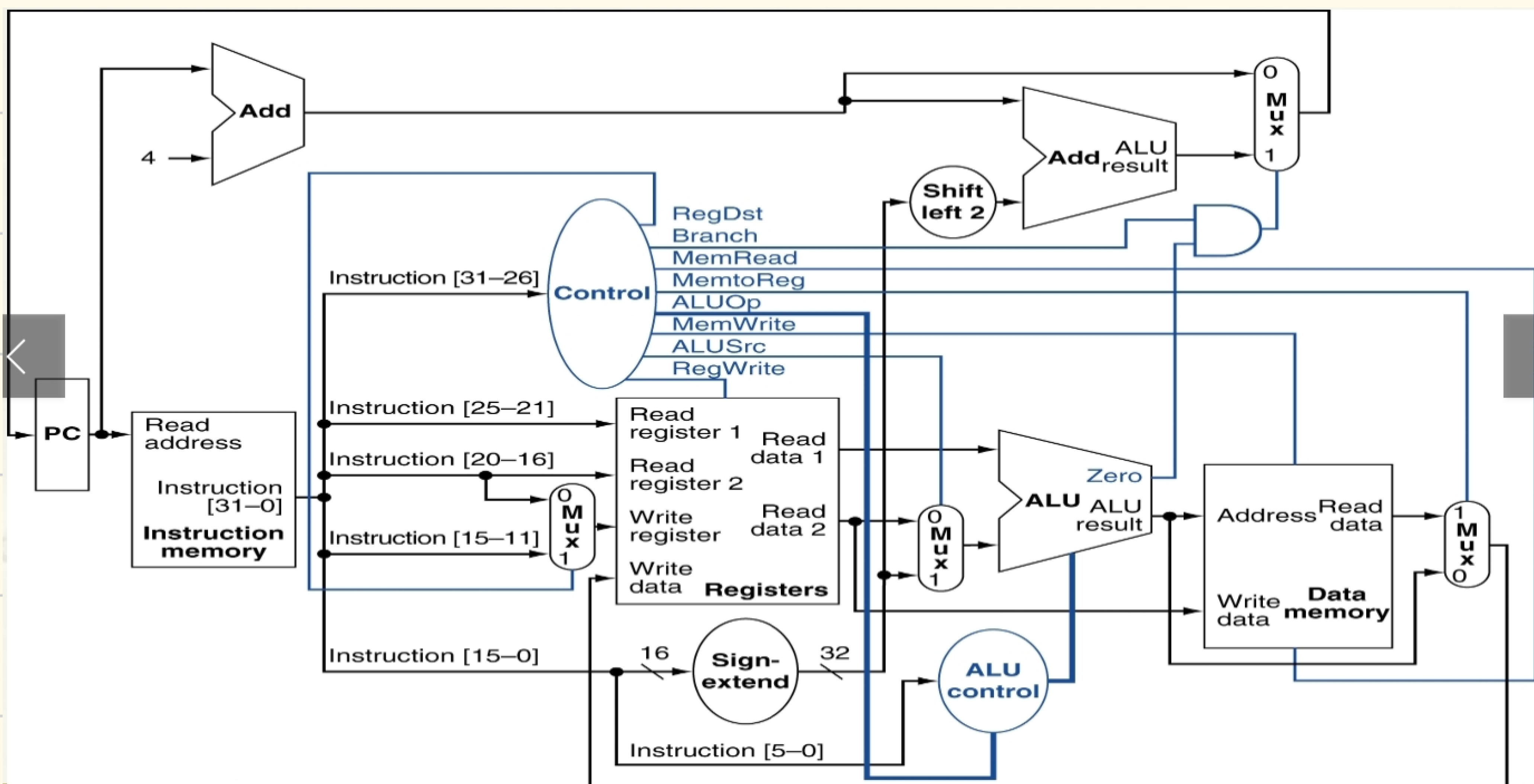
④ 存储器访问 (只有 LW/SW)

⑤ 写结果至寄存器

I型: 写入 rt

R型: 写入 rd

4.4 简单的实现机制



主控制单元

指令操作码(6)

主控制器

ALU op (2)

Mux (4)

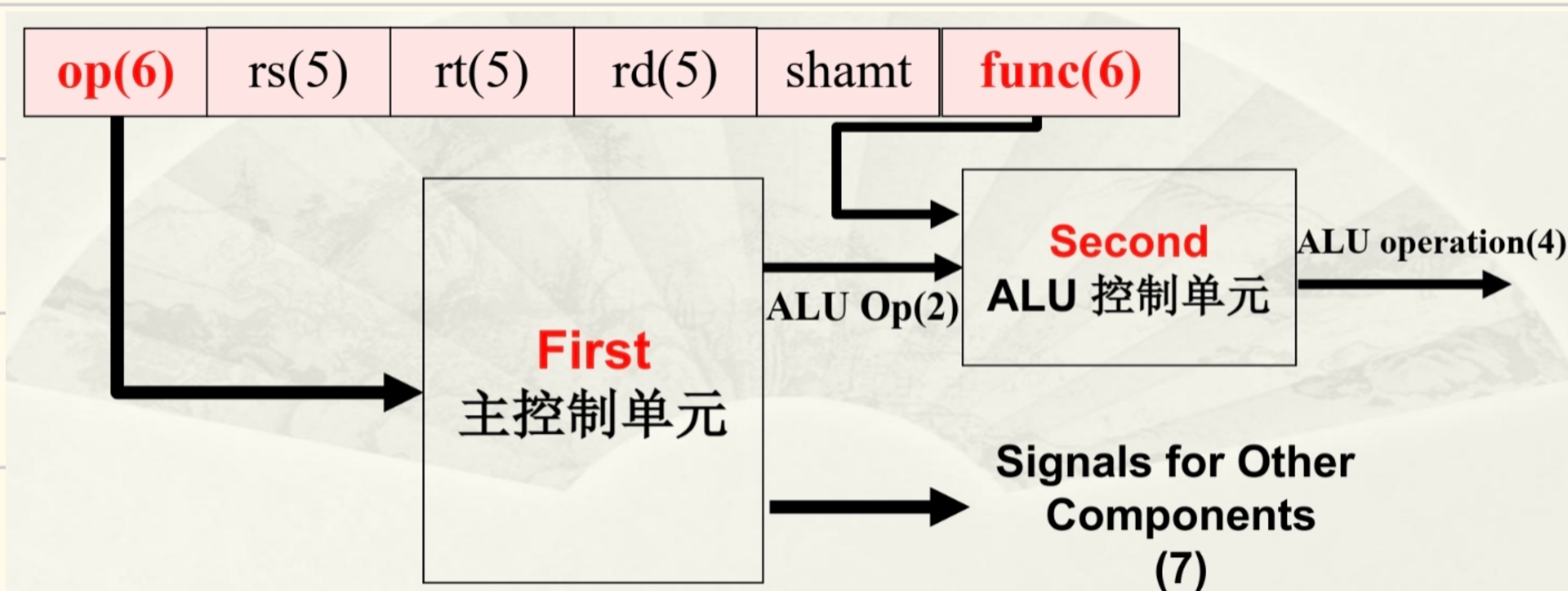
R/W (3)

LW	00
SW	00
Beq	01
R-type	10

RegDst	写reg相关
ALUScr	branch相关
PCSrc	branch相关
MemtoReg	

MemRead
MemWrite
RegWrite

两级译码



主控制信号

信号名	无效时(=0)的含义	有效时(=1)的含义
RegDst	写寄存器的目标寄存器号来自rt字段(20:16)	写寄存器的目标寄存器号来自rd字段(15:11)
RegWrite	无	寄存器堆写使能有效
ALUScr	第二个ALU操作数来自寄存器堆的第二个输出(Read data 2)	第二个ALU操作数为指令低16位的符号扩展
PCSrc (Branch)	PC 由PC+4取代	PC由分支目标地址取代
MemRead	无	数据存储器读使能有效
MemWrite	无	将写数据端 (Write data) 的数据写入到指定地址的存储单元中
MemtoReg	写入寄存器的数据来自ALU	写入寄存器的数据来自数据存储器

指令操作码	ALUOp
lw	00
sw	00
beq	01
R-type	10

lw/sw $\Rightarrow$ ALUOp = 00 $\Rightarrow$ 加法

beq $\Rightarrow$ ALUOp = 01 $\Rightarrow$ 减法

R型 $\Rightarrow$ ALUOp = 10 $\Rightarrow$ 加、减、与、或、小于则置位
取决于func(6)字段

ALU控制单元

指令操作码	ALUOp	指令操作	funct字段	ALU 动作	ALU 控制信号	
lw	00	取字	XXXXXX	add	0010	
sw	00	存字	XXXXXX	add	0010	
beq	01	相等则分支	XXXXXX	subtract	0110	
R-type	10	加	100000	add	0010	
		减	100010	subtract	0110	
		与	100100	AND	0000	
		或	100101	OR	0001	
		小于则置位	101010	set-on-less-than	0111	

ALU控制信号	功能	ALU控制信号	功能
0000	与	0110	减
0001	或	0111	小于则置位
0010	加	1100	或非

真值表→逻辑关系式→卡诺图化简→门电路

ALUOp		Funct										
O ₁	O ₀	F ₅	F ₄	F ₃	F ₂	F ₁	F ₀	C ₃	C ₂	C ₁	C ₀	
0	0	X	X	X	X	X	X	0	0	1	0	lw/sw
X	1	X	X	X	X	X	X	0	1	1	0	beq
1	X	X	X	0	0	0	0	0	0	1	0	add
1	X	X	X	0	0	1	0	0	1	1	0	sub
1	X	X	X	0	1	0	0	0	0	0	0	and
1	X	X	X	0	1	0	1	0	0	0	1	or
1	X	X	X	1	0	1	0	0	1	1	1	slt

对C₃, C₃=0

对C₂, $C_2 = O_0 + O_1(\overline{F_3}\overline{F_2}F_1\overline{F_0} + F_3\overline{F_2}F_1\overline{F_0})$

$F_3 F_2 \backslash F_1 F_0$		00	01	11	10
		0	x	x	1
00	0	x	x	1	
01	0	0	x	x	
11	x	x	x	x	
10	x	x	x	1	

$$\Rightarrow C_2 = O_1 F_1 + O_0$$

对 C_1 ,

$F_3F_2 \backslash F_1F_0$	00	01	11	10
00	1	X	X	1
01	0	0	X	X
11	X	X	X	X
10	X	X	X	1

$$\Rightarrow C_1 = \overline{O_1}\overline{O_0} + O_0 + O_1\overline{F_2}$$

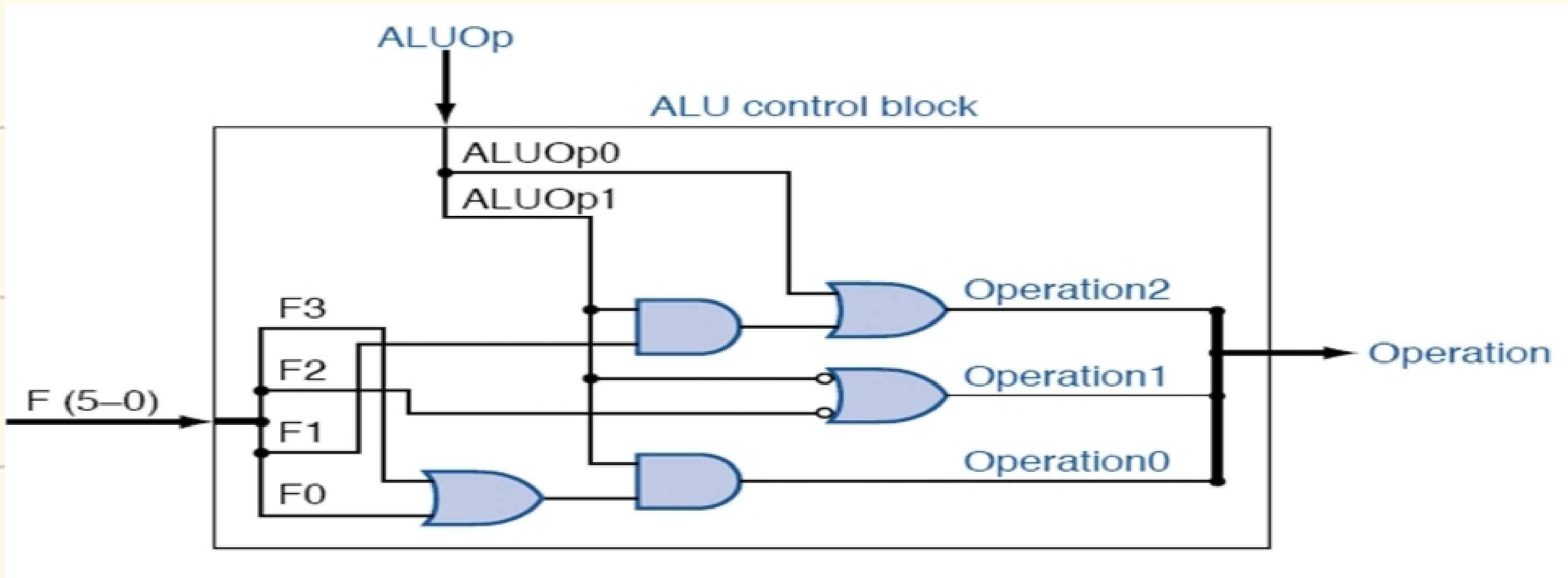
F ₂ \ O ₁ O ₀		00	01	11	10
0	1	1	1	1	1
1	1	1	1	1	0

$$\Rightarrow C_1 = \overline{O_1} + O_0 + \overline{F_2}$$

对 C_0 , F_1F_0 F_3F_2

	00	01	11	10
00	0	X	X	0
01	0	1	X	X
11	X	X	X	X
10	X	X	X	1

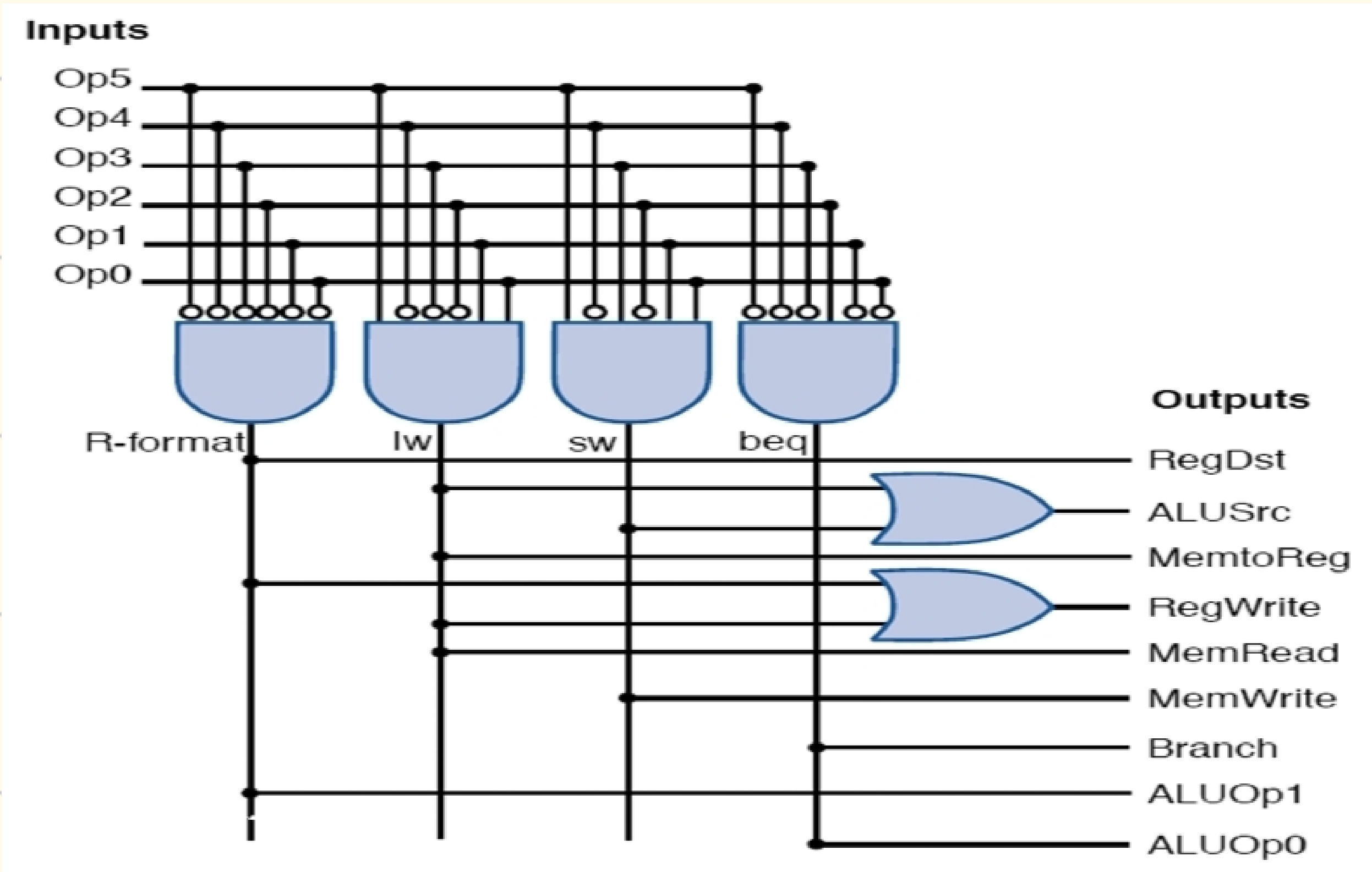
$\Rightarrow C_0 = O_1(F_0 + F_3)$



主控制单元

Input		Output								
Instruction	OPCode	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	000000	1	0	0	1	0	0	0	1	0
Lw(I-Type)	100011	0	1	1	1	1	0	0	0	0
sw(I-Type)	101011	X	1	X	0	0	1	0	0	0
beq(I-Type)	000100	X	0	X	0	0	0	1	0	1

未使用的 < 多路器 → 无关项
读写使能 → 置0



时钟周期长度确定

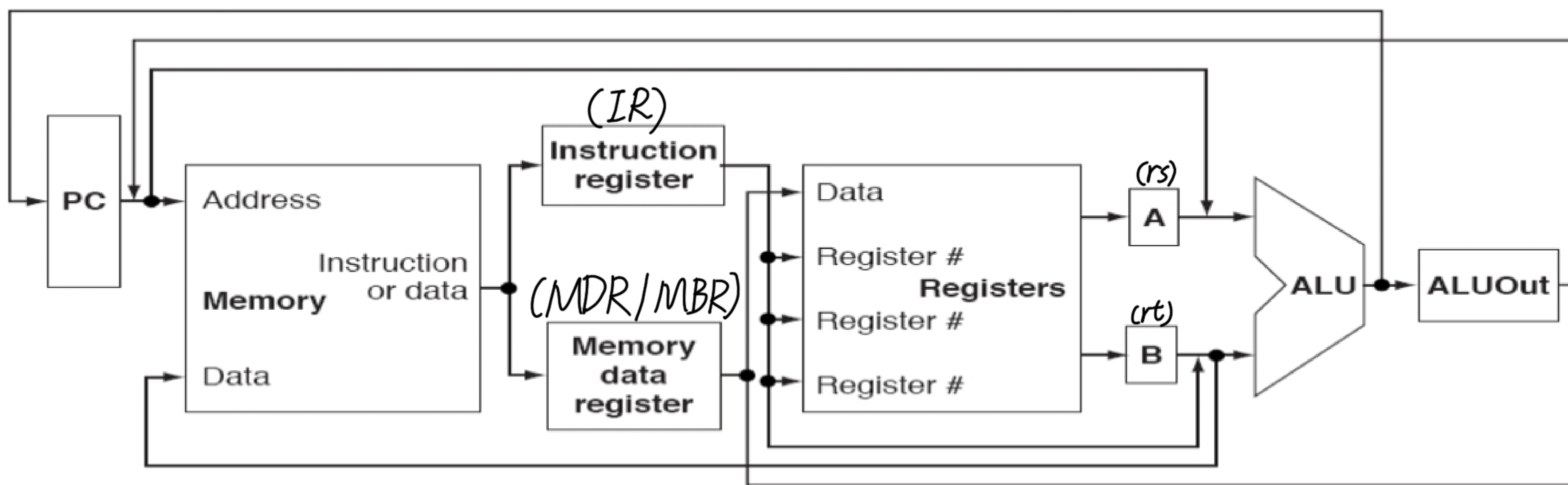
Instruction class	Instruction memory (time)	Register read (time)	ALU operation (time)	Data Memory (time)	Register Write (time)	Total Time
R-type	200	50	100		50	400
LW	200	50	100	200	50	600
SW	200	50	100	200		550
BEQ	200	50	100			350
Jump	200					200

⇒ 时钟周期长度

控制信号表

Input		Output									
Instruction	OPCode	RegDst	ALUSrc	Memto-Reg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0	jump
R-format	000000	1	0	0	1	0	0	0	1	0	0
Lw(I-Type)	100011	0	1	1	1	1	0	0	0	0	0
sw(I-Type)	101011	X	1	X	0	0	1	0	0	0	0
beq(I-Type)	000100	X	0	X	0	0	0	1	0	1	0
j		X	X	X	0	X	0	0	X	X	1

4.5 多周期实现机制



部件变化：指令存储器、数据存储器合为一个

3个ALU合为一个

各功能单元后均有寄存器暂存：IR、MDR、A、B、ALUOut

五个周期：

1. **IF**：取指
2. **ID**：译码和读寄存器
3. **EX (BC)**：执行/地址计算（分支完成）
4. **MEM (WB)**：存储器访问（R型指令写回）
5. **WB**：写回

一条指令占用3~5周期

取指周期IF (RTL描述)

$$IR = Memory[PC];$$

$$PC = PC + 4;$$

$$Mem[PC] = B$$

$$Reg[Reg[IR[15:11]]] = ALUOut$$

译码和读寄存器ID

$$A = Reg[IR[20:21]];$$

$$B = Reg[IR[20:16]];$$

$$ALUOut = PC + (sign-extend(IR[15:0]) << 2); \quad // \text{计算分支目标地址}$$

执行/地址计算EX

①访存lw/sw

$$ALUOut = A + sign-extend(IR[15:0]);$$

②R型

$ALUOut = A \text{ op } B;$

③条件分支

$if(A==B) \quad PC=ALUOut;$

④跳转指令

$PC = \{PC[31-28], (IR[25-0] \ll 2)\};$

访存周期MEM

① lw

$MDR = Memory[ALUOut];$

② sw

$Memory[ALUOut]=B;$

R型写回周期WB

$Reg[IR[15-11]] = ALUOut;$

写回周期WB

lw:

$Reg[IR[20-16]] = MDR;$

Step	Step name	R-type	memory-reference	branches	jumps
1: IF	Instruction fetch	IR = Memory[PC] PC = PC + 4			
2: ID	Instruction decode/register fetch	A = Reg [IR[25-21]] B = Reg [IR[20-16]] ALUOut = PC + (sign-extend (IR[15-0]) << 2)			
3: EX	Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend(IR[15-0])	if (A ==B) then PC = ALUOut	PC = {PC [31-28], (IR[25-0]<<2)}
4: MEM	Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
5: WB	Memory read completion		Load: Reg[IR[20-16]] = MDR		

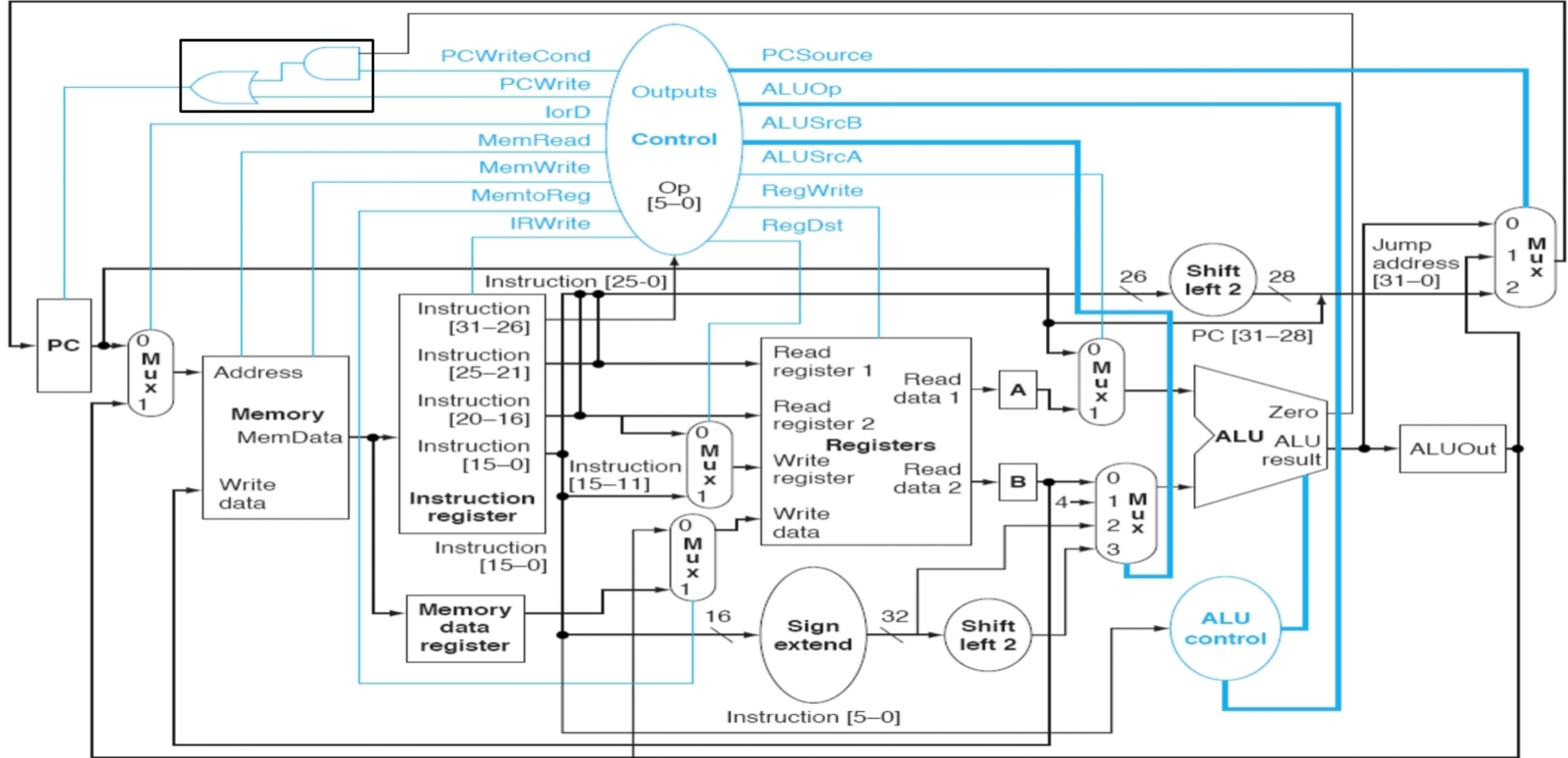
CPI

(指令平均周期数)

Load: 5/store : 4/R-type : 4/branch : 3/jump: 3

$CPI = \sum \frac{i \text{指令条数}}{\text{指令总条数}} \times CPI_i$

多周期实现



控制信号

信号名	无效时(=0)的含义	有效时(=1)的含义
RegDst	写寄存器的目标寄存器号来自rt字段(20:16)	写寄存器的目标寄存器号来自rd字段(15:11)
RegWrite	无	寄存器堆写使能有效
ALUScrA	ALU 第一个操作数来源于PC	ALU 第一个操作数来源于A 寄存器
MemRead	无	数据存储器读使能有效
MemWrite	无	将写数据端（Write data）的数据写入到指定地址的存储单元中
MemtoReg	写入寄存器的数据来源于ALUOut	写入寄存器的数据来源于MDR
lorD	PC 提供存储器访问地址	ALUOut提供存储器访问地址
IRWrite	无	将存储器中取出的指令写入IR (收取指令时为1)
PCWrite	无	PC写使能有效
PCWriteCond	无	带条件的PC写使能有效（Zero输出有效）

信号名	值	功能
ALUOp	00	ALU执行加法运算
	01	ALU执行减法运算
	10	由 funct 字段决定ALU的具体操作
ALUScrB	00	ALU 第二个操作数来源于B 寄存器
	01	ALU 第二个操作数是常数4
	10	ALU 第二个操作数是由IR中低16位立即数字段经扩展后得到
	11	ALU 第二个操作数是由IR中低16位立即数字段经扩展后并左移两位得到
PCSource	00	ALU的运算结果(PC+4) 被送往PC待写
	01	ALUOut 中的内容(分支目标地址)被送往PC待写
	10	跳转目标地址(IR[25:0]左移 2 位与PC+4[31:28]拼接)被送往PC待写

控制实现

有限状态机 —— 组合逻辑控制器(CISC) ✓
微程序编程 —— 微程序控制器(x86)

例如: lw \$t0, 4(\$t1) 实现过程

① 转为机器指令

$\frac{100011}{IR[31:26]} \quad \frac{01001}{IR[25:21]} \quad \frac{01000}{IR[20:16]} \quad \frac{0000000000000000}{IR[15:0]}$

② 执行过程

IF: $IR = Memory[PC];$

$PC = PC + 4;$

ID: $A = Reg[IR[25:21]];$

$B = Reg[IR[20:16]];$

$ALUOut = PC + (sign-extend(IR[15:0]) \ll 2);$

EXE: $ALUOut = A + sign-extend(IR[15:0]);$

MEM: $MDR = Memory[ALUOut];$

WB: $Reg[IR[20:16]] = MDR$

③ 设置控制信号

IF: $IorD = 0, MemRead = 1, MemWrite = 0,$
 $IRWrite = 1, ALUSrcA = 0, ALUSrcB = 01,$
 $ALUOp = 00, PCSrc = 00$

ID: $ALUSrcA = 0, ALUSrcB = 11, ALUOp = 00$

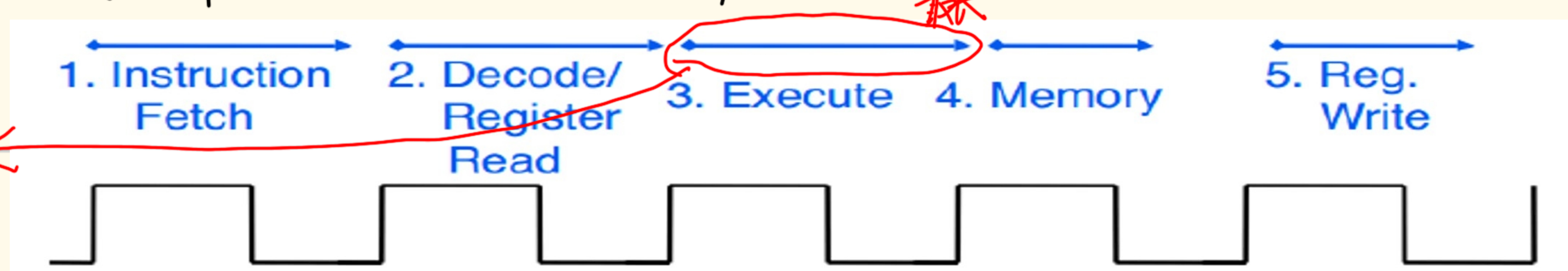
EXE: $ALUSrcA = 1, ALUSrcB = 10, ALUOp = 00$

MEM: $MemRead = 1, IorD = 1$

WB: $RegDest = 0, MemtoReg = 1, RegWrite = 1$

时钟周期长度的确定

时钟周期



5

Ideally one would desire an indefinitely large memory capacity such that any particular ... word would be immediately available. ... We are ... forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

A. W. Burks, H. H. Goldstine, and J. von Neumann

Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, 1946

Large and Fast: Exploiting Memory Hierarchy

5.1	Introduction	374
5.2	Memory Technologies	378
5.3	The Basics of Caches	383
5.4	Measuring and Improving Cache Performance	398
5.5	Dependable Memory Hierarchy	418
5.6	Virtual Machines	424
5.7	Virtual Memory	427

第5章 存储器

5.1 引言

存储器分类

高速缓存 (Cache)
内存 (Memory)
辅存 (外存)

性能指标

容量
速度
价格

提高性能的基本思路

时间局部性
空间局部性

三级存储体系结构

高速缓存 (SRAM) — 内存 (DRAM) — 外存 (磁盘/闪存)
主存储器 虚拟存储器

命中率、缺失率、命中时间、缺失代价

命中率：在高层存储器中找到目标数据的存储访问比例。

缺失率 (失效率)：在高层存储器中没有找到目标数据的存储访问比例。

命中时间：访问高层存储器所需要的时间，包括判断是否命中所需时间。

缺失代价 (开销)：将相应的块从低层存储器替换到高层存储器所需的时间。

平均访问时间 $AMAT = \text{命中时间} + \text{缺失率} \times \text{缺失代价}$

5.2 存储器技术

SRAM技术(cache)

利用双稳态触发器存储信息, $\Rightarrow$ 硬件规模大

DRAM技术(内存)

利用电容存储电荷原理 $\Rightarrow$ 硬件规模小, 密度大
因电荷泄露, 需周期性按行刷新

SRAM、DRAM共同点

对任何数据访问时间均固定 $\Rightarrow$ 随机访问
断电后丢失数据 $\Rightarrow$ 易失性存储器

flash闪存(外存)

电可擦除可编程的只读存储器 (EEPROM)

磁盘存储器(外存)

利用磁层上不同方向磁化区域
非破坏性读出、顺序存取方式
以扇区为最小寻址单位

5.3 数据检验方法

码距(汉明距离)

同一编码中, 任意两个合法编码之间不同二进制数位数的最小值

奇偶校验

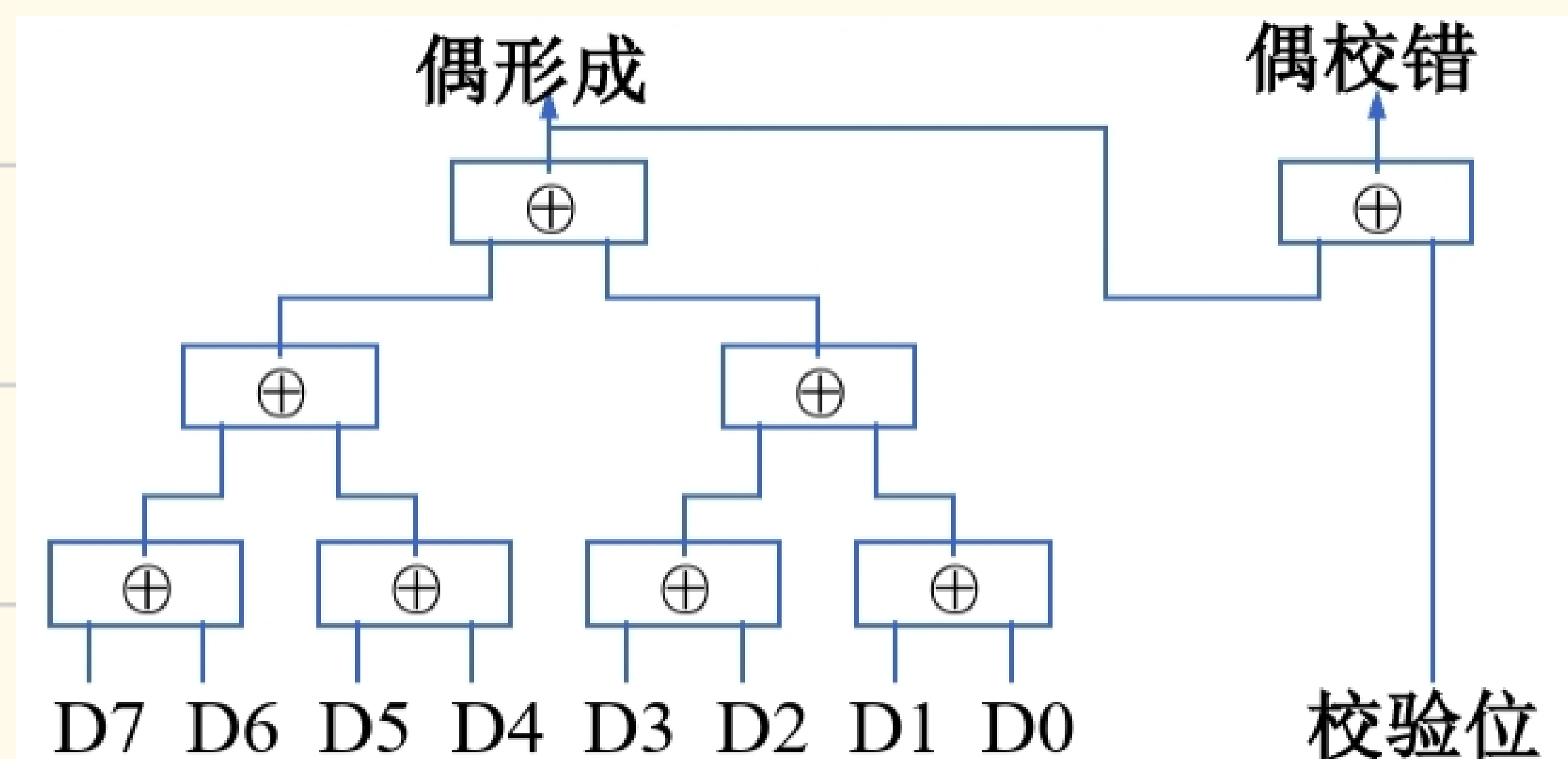
00000000

奇校验码 000000001

偶校验码 000000000

码距为2

仅能发现奇数个位数出错,只能检错



海明校验

校验位个数: $2^r - 1 \geq \text{总位数 } N = \text{有效信息 } k + \text{校验 } r$

k	2~4	5~11	12~26
r	3	4	5

校验位位置: 第 $2^0, 2^1, \dots, 2^{r-1}$ 位

P_i 负责校验连续的 2^{i-1} 个, 从第二段开始

p1负责校验: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9...

p2负责校验: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9....

p3负责校验: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16....

p4负责校验: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18....

eg. 1011 偶海明校验码生成

$$2^r - 1 \geq 4 + r \Rightarrow r = 3$$

1	2	3	4	5	6	7
P_1	P_2	1	P_3	0	1	1

P_1 负责 1, 3, 5, 7 位: 101 $\Rightarrow P_1 = 0$

P_2 负责 2, 3, 6, 7 位: 111 $\Rightarrow P_2 = 1$

P_3 负责 4, 5, 6, 7 位: 011 $\Rightarrow P_3 = 0$

检验 0110011

G_1 负责 1, 3, 5, 7 位: 0101 $\Rightarrow G_1 = 0$

G_2 负责 2, 3, 6, 7 位: 1111 $\Rightarrow G_2 = 0$

G_3 负责 4, 5, 6, 7 位: 0011 $\Rightarrow G_3 = 0$

$\Rightarrow$ 指误字 $G_3G_2G_1 = 000$, 无误

若 1 位错为 0110111, 指误字 $G_3G_2G_1 = 101$, 第 5 位错

码距为3: 检验+纠正1位错 or 检验2位错

改进: 纠正1位错 and 检测2位错

⇒ 增加一位整字奇偶校验码

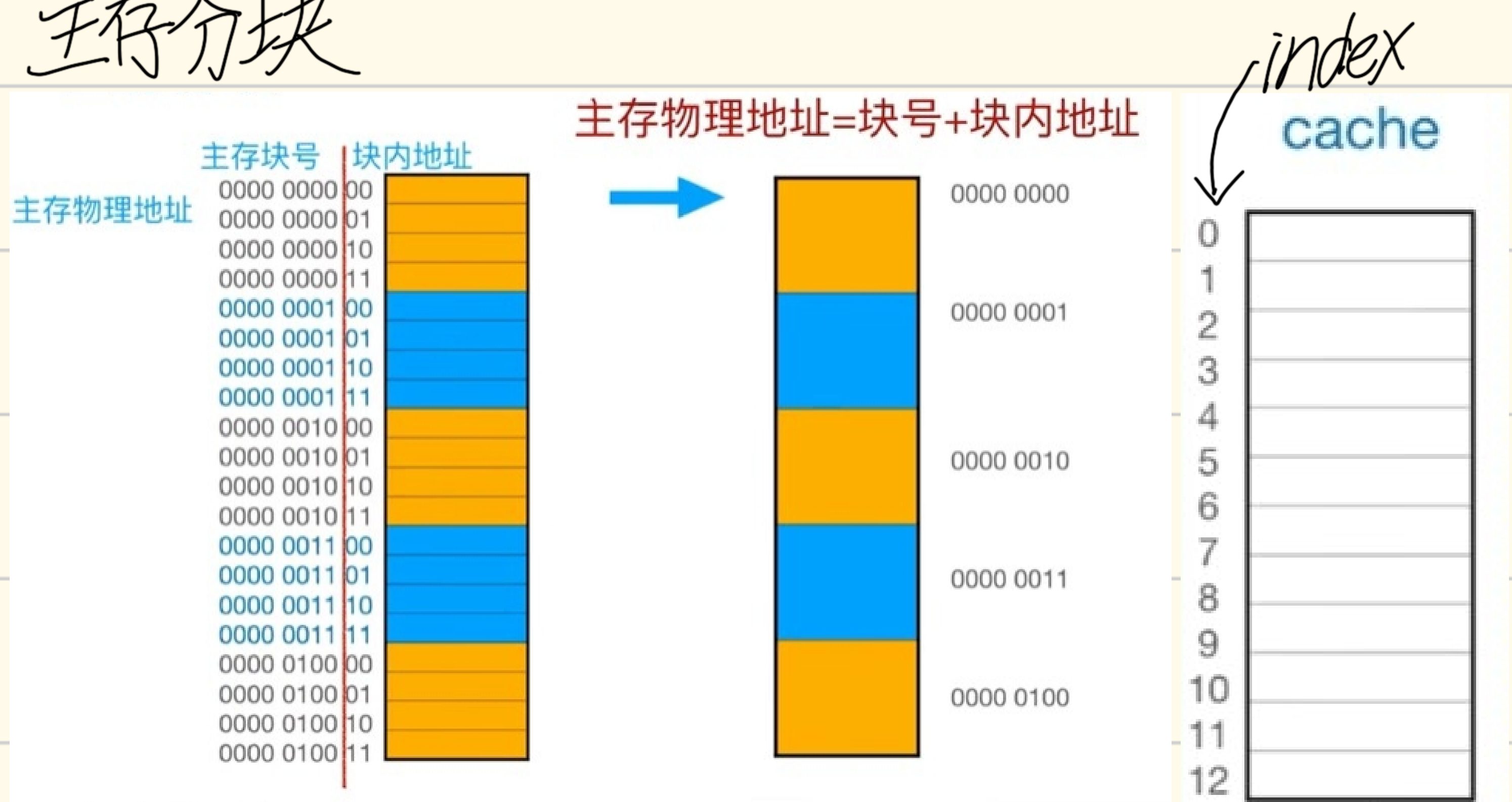
1	2	3	4	5	6	7	8
P1	P2	d1	P3	d2	d3	d4	P4

其中, $H=G3G2G1$, 整字校验结果为G4。

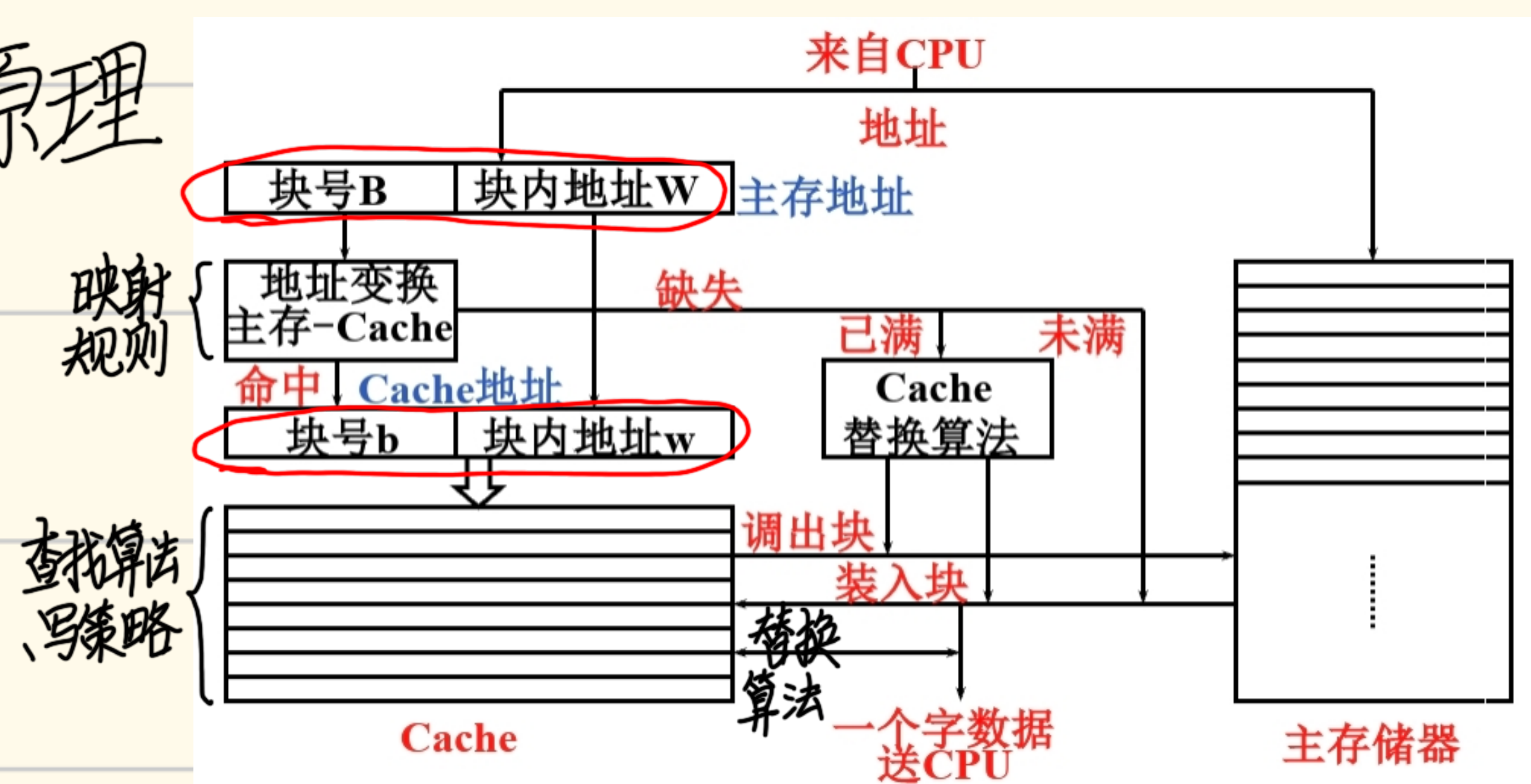
此时码距为4

5.4 Cache基本原理

主存分块

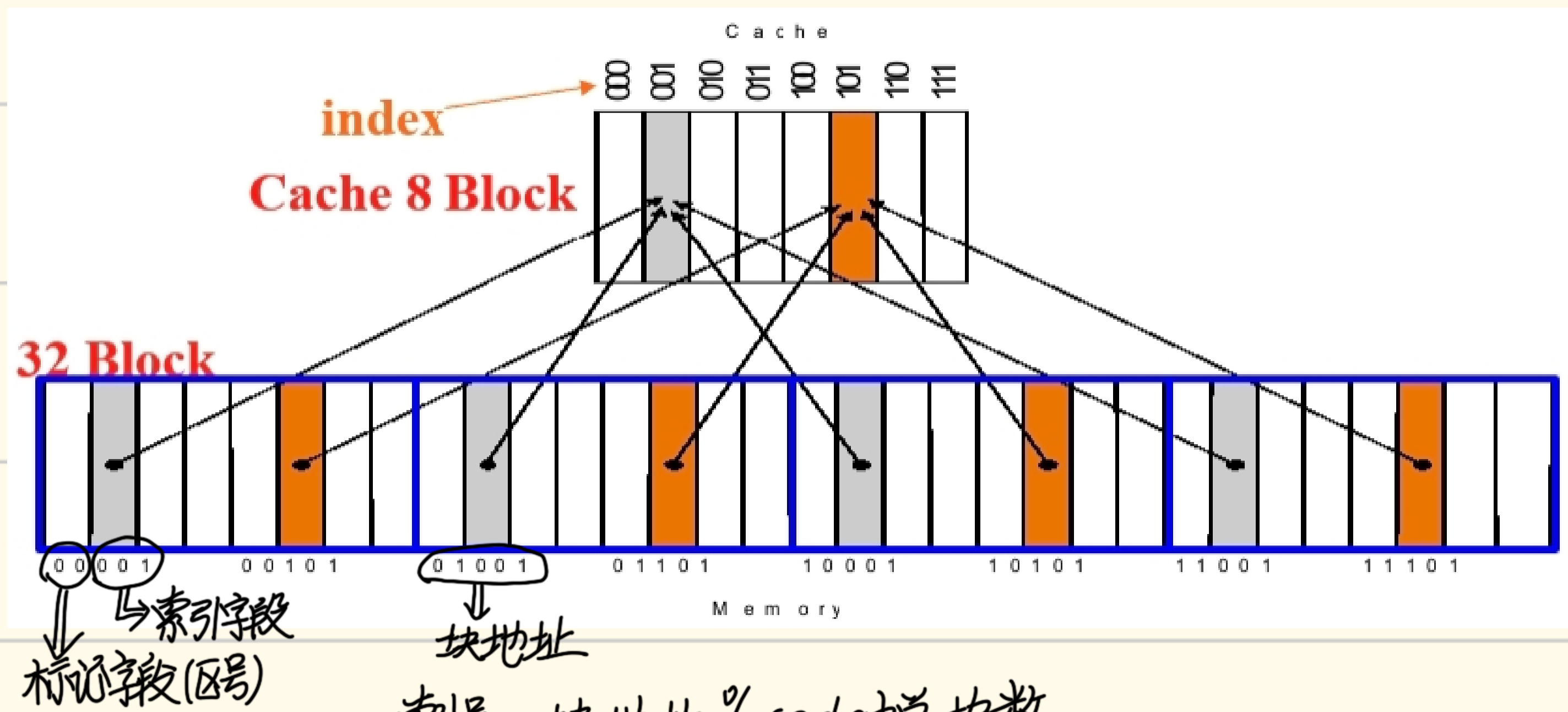


基本工作原理



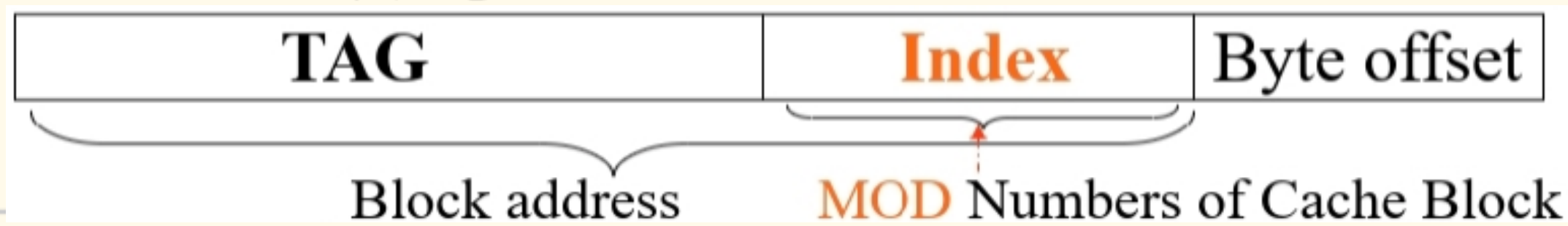
映射方式

直接映射



索引号 = 块地址 % cache中总块数

访问Cache的信息:



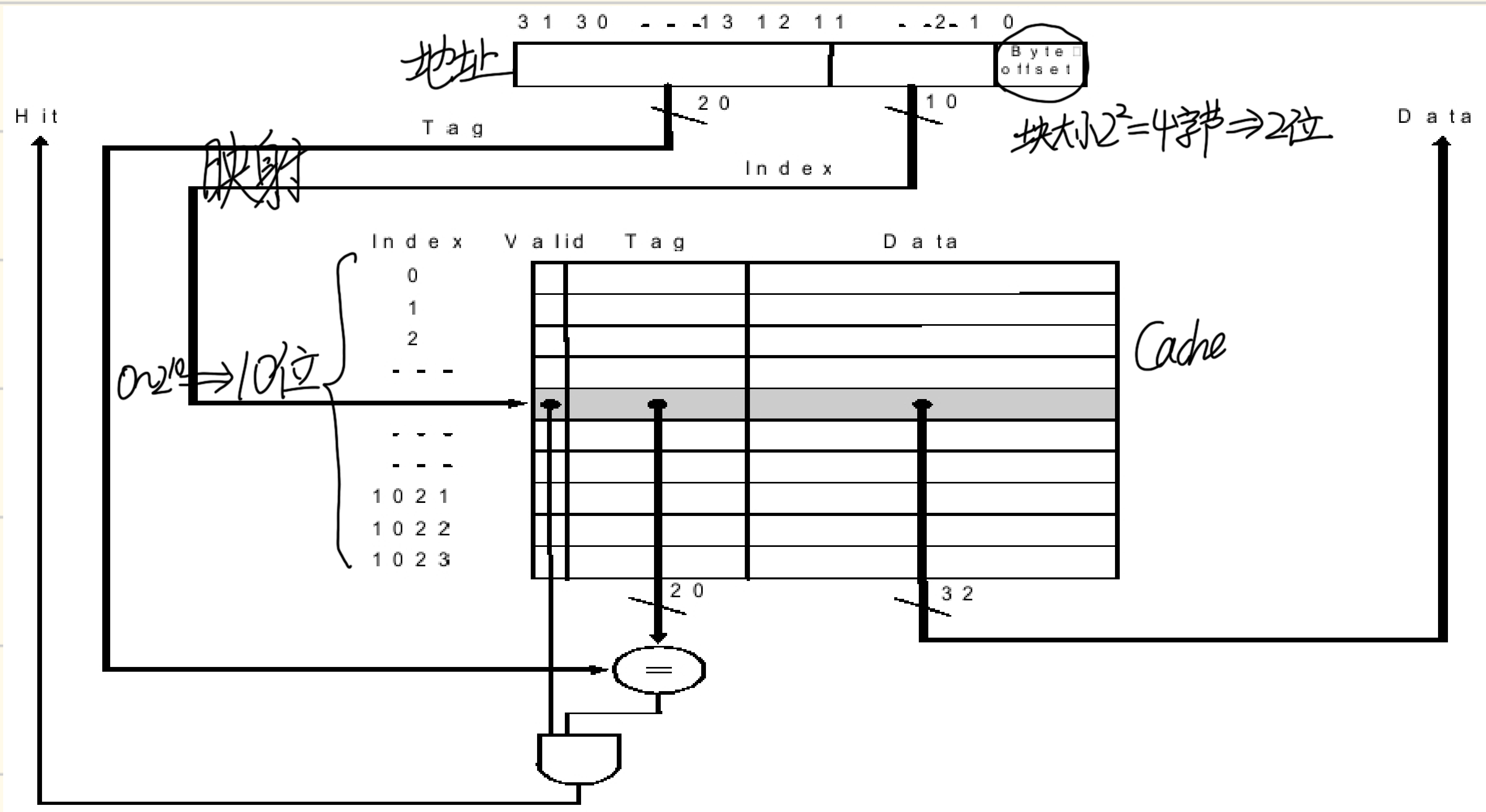
Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

a. The initial state of the cache after power-on

eg. 访问序列 10010 → index

Index	V	Tag	Data
000	Y	(10) ₂	Memory(10000)
001	N	区号不同 → 仍缺失	
010	Y	(11) ₂	Memory(11010)
011	Y	(00) ₂	Memory(00011)
100	N		
101	N		
110	Y	(10) ₂	Memory(10110)
111	N		

Index	V	Tag	Data
000	Y	(10) ₂	Memory(10000)
001	N		
010	Y	(11) → (10) ₂	Memory(10010)
011	Y	(00) ₂	Memory(00011)
100	N		
101	N		
110	Y	(10) ₂	Memory(10110)
111	N		



Cache位数计算

$$\text{Cache位数} = (\text{有效位}(1\text{位}) + \text{Tag} + \text{数据块位数}) \times \text{块数}$$

$$\text{地址位数} = \text{Tag} + \text{Index} + \text{Offset}$$

$$\text{其中 Index 有 } n \text{ 位} \Leftrightarrow \text{块数有 } 2^n \text{ 块} = \frac{\text{总数据量}}{\text{块大小}}$$

$$\text{Offset 有 } n \text{ 位} \Leftrightarrow \text{块大小为 } 2^n \text{ bits}$$

$$\text{数据块位数 } n \text{ 位} \Leftrightarrow \text{块大小为 } \frac{n}{8} \text{ byte} = \frac{n}{8} \times \frac{1}{4} \text{ word}$$

eg. 直接映射, 16KB数据, 块大小4个字, 内存地址32位。

$$\text{块数} = \frac{16 \times 2^{10} \text{ byte}}{4 \times 4 \text{ byte/块}} = 2^{10} \text{ 个} \Leftrightarrow \text{Index 10位}$$

$$\text{块大小 } 4 \text{ word} = 16 \text{ byte} \Leftrightarrow \text{offset 4位}$$

$$\text{Tag} = \text{内存32位} - \text{Index} - \text{offset} = 18 \text{ 位}$$

$$\text{Cache} = \text{块数} \times (\text{Valid 1位} + \text{Tag} + \text{块大小 } 16 \times 8 \text{ bit})$$

$$\Rightarrow \text{Cache} = 147 \times 2^{10} \text{ bit}$$

$$\text{块地址} = \text{主存字节地址} \div \text{块大小} (\text{向下取整})$$

$$\text{Index} = \text{块地址} \% \text{块数}$$

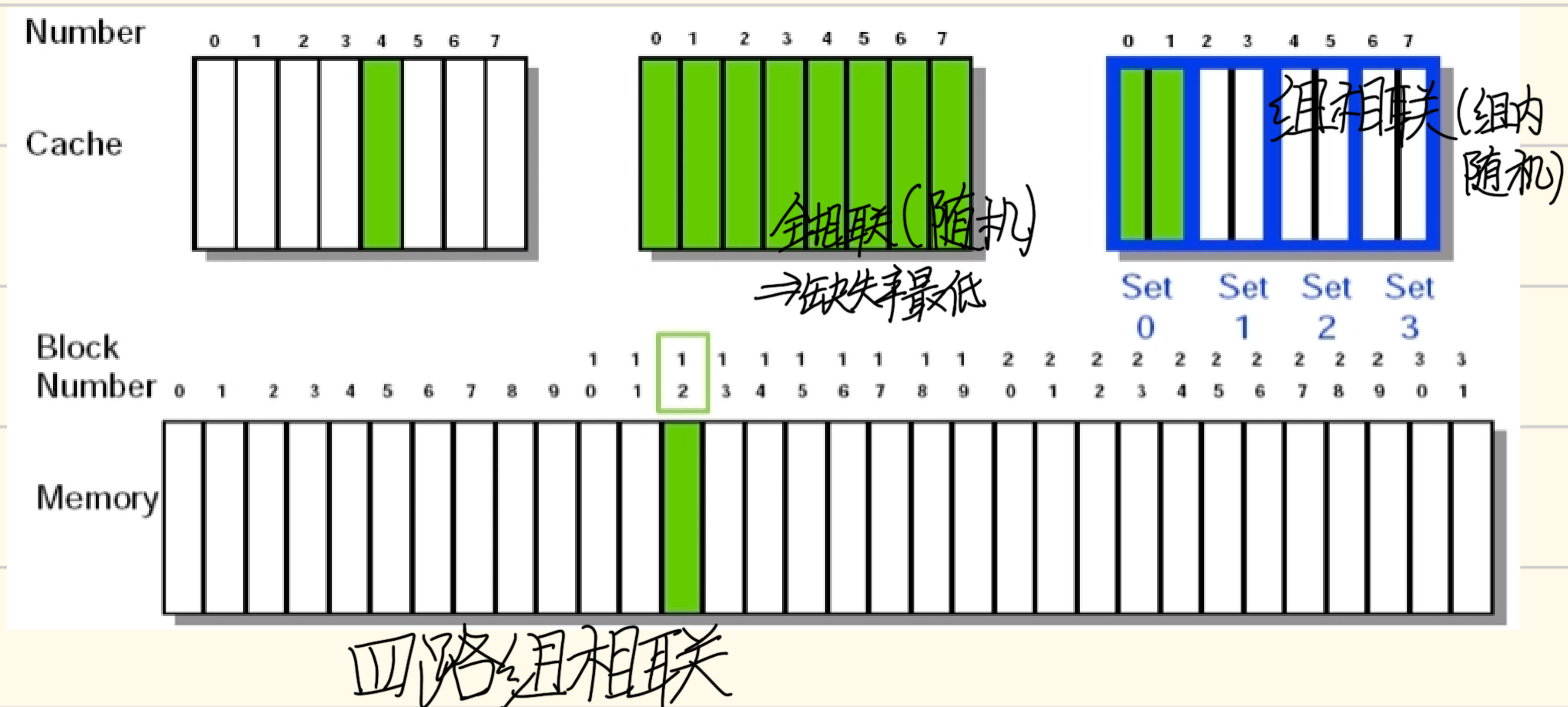
$$\text{Tag} = \text{块地址除去 Index 的前面部分}$$

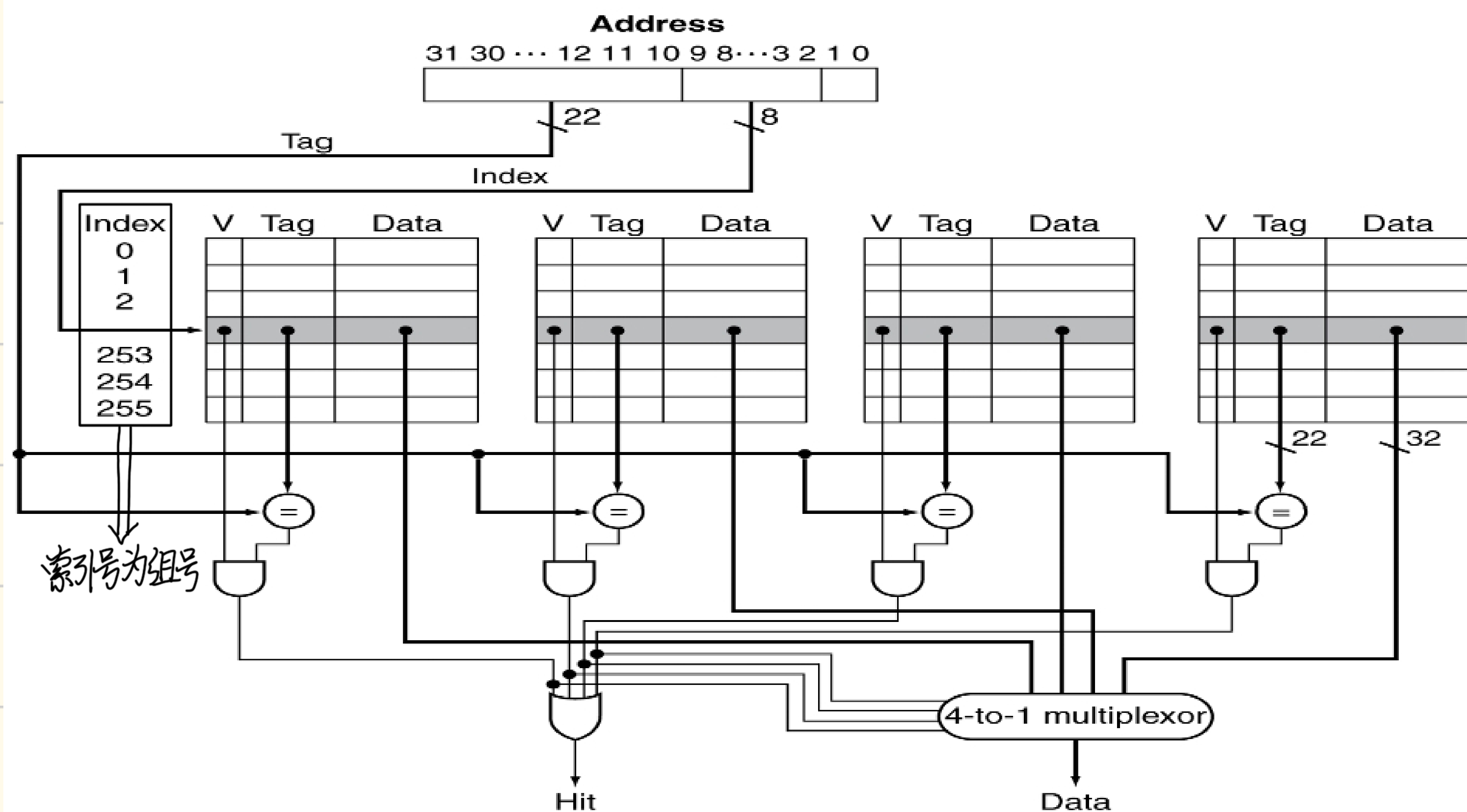
eg. 例2: 假设一个cache中有64个块, 每块大小为16字节, 采用直接映射。那么主存1200号单元所在主存块将被映射到cache中的哪一块?

$$\text{块地址} = 1200 / 16 = 75$$

$$\text{Index} = 75 \% 64 = 11$$

全相联、组相联





n -way 组相联 $\Rightarrow$ 每行 (每个 index) 含 n 个块

eg. 2GB 主存, 128KB 的 Cache, 块大小 64B,
内存地址 01000000 00010001 00001011
10000101, 2-way 组相联, 映射的 Tag = ?

块大小 64B = 2^6 byte $\Leftrightarrow$ offset = 6 位

组数 = $\frac{128\text{KB}}{2\text{路} \times 64\text{B}} = 2^{10}$ 组 $\Leftrightarrow$ index = 10 位

$\Rightarrow$

16 位	10 位	6 位
Tag	Index	Offset

$\Rightarrow$ Tag = 高 16 位 = 01000000 00010001

映射方式对比

- ① **直接映射**: 查找速度最快, 命中时间最短, 但缺失率最高。
- ② **全相联映射**: 缺失率最低, 查找速度最慢, 命中时间最长。
- ③ **组相联映射**: 介于两者之间。

写入策略

(1)写直达法 (Write-through) 硬件要求高、一致性好、一次写1 word

也叫写贯穿法，这种方法所实行的是，当CPU进行写操作时，在写Cache的同时也将内容写入到相应的主存单元中，即两个内容同时改写

(2)写回法 (Write-back) 需要外校验位(一次写1个数据块)

这种方法的要点有二，一是当CPU写数据时，只写Cache，不写主存；二是当已改写的块被替换出Cache时，将其内容写回主存。

置换策略

(直接映象无需置换算法, 替换位置均固定)

①随机法

②FIFO法

③LRU(最近最少使用)法 Least Recently used (最近被访问, LRU=1)

eg. 例: 2路组相联缓存的容量为16B, 块大小为2Bytes, 缓存替换采用LRU策略, 当前缓存状态见下表。

index	Way 0				Way 1			
	Tag	Off 1	Off 0	LRU	Tag	Off 1	Off 0	LRU
00	00	42	AC	0	11	45	A1	1
01	10	1E	33	1	01	31	41	1
10	11	32	34	0	00	34	63	0
11	10	76	72	1				

假设内存中的存放数据如下, 若按内存地址00011读取后, 缓存的状态是什么?

注意: LRU=1, 表示最近被访问过。

data 2byte $\Leftrightarrow$ 1位offset

组数 $\frac{16B}{2byte \times 2路} = 4组 = 2^2组 \Leftrightarrow 2位 index$

$\Rightarrow$

Tag	Index	Off set
2位	2位	1位

Address	Data
00000	AB
00001	3F
00010	12
00011	8C

★一个块一起替换!!!

Cache性能评估

CPU时间 = (CPU执行时钟周期数 + 存储器阻塞时钟周期数) $\times$ 时钟周期

存储器阻塞时钟周期数 = 存储器访问次数 $\times$ 缺失率 $\times$ 缺失代价
or
= 指令数 $\times$ 缺失率 $\times$ 缺失代价

AMAT平均访存时间 = 命中时间 + 缺失率 $\times$ 缺失代价

B

A P P E N D I X

I always loved that word, Boolean.

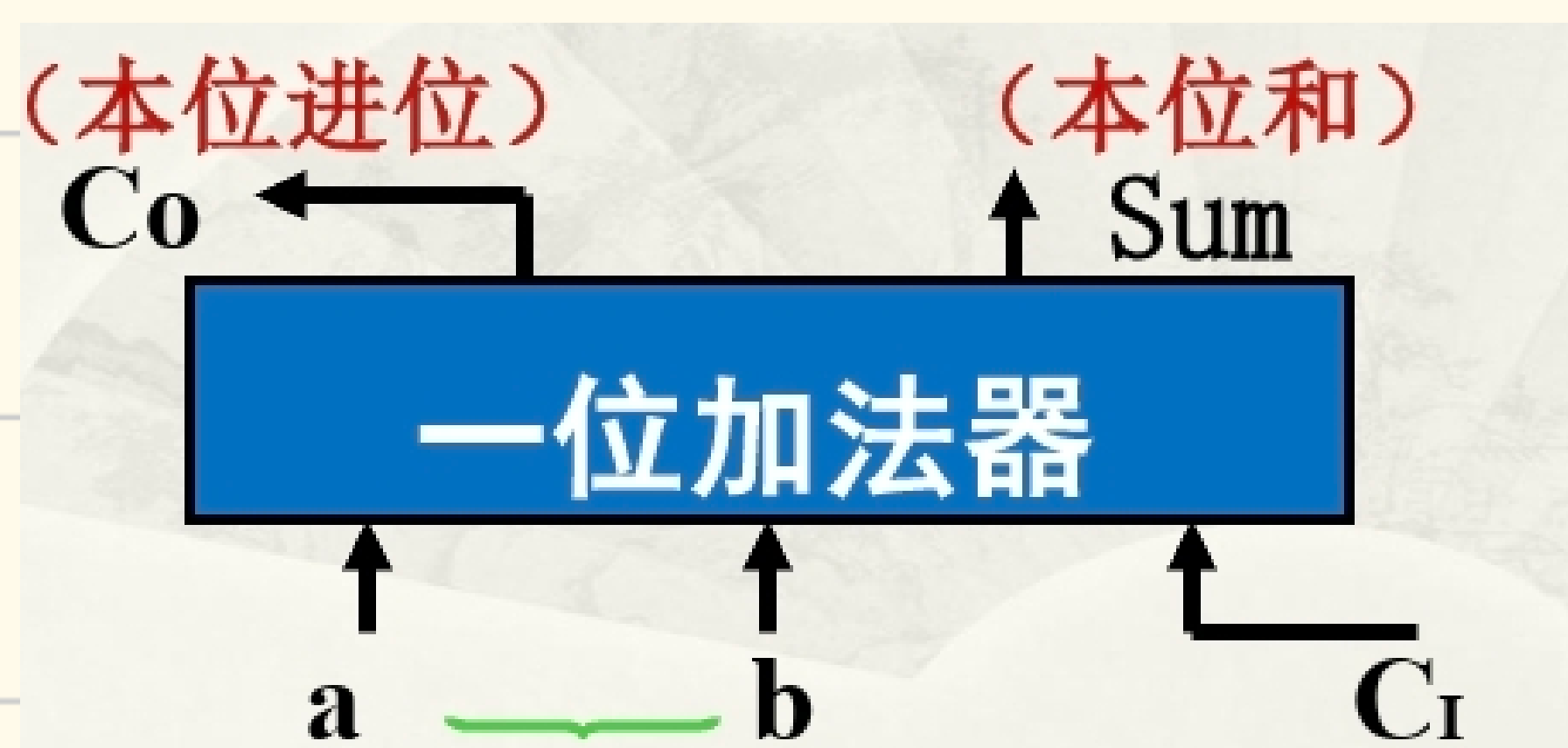
Claude Shannon
IEEE Spectrum, April 1992
(Shannon's master's thesis showed that the algebra invented by George Boole in the 1800s could represent the workings of electrical switches.)

The Basics of Logic Design

- B.1 Introduction** B-3
- B.2 Gates, Truth Tables, and Logic Equations** B-4
- B.3 Combinational Logic** B-9
- B.4 Using a Hardware Description Language** B-20
- B.5 Constructing a Basic Arithmetic Logic Unit** B-26
- B.6 Faster Addition: Carry Lookahead** B-38
- B.7 Clocks** B-48

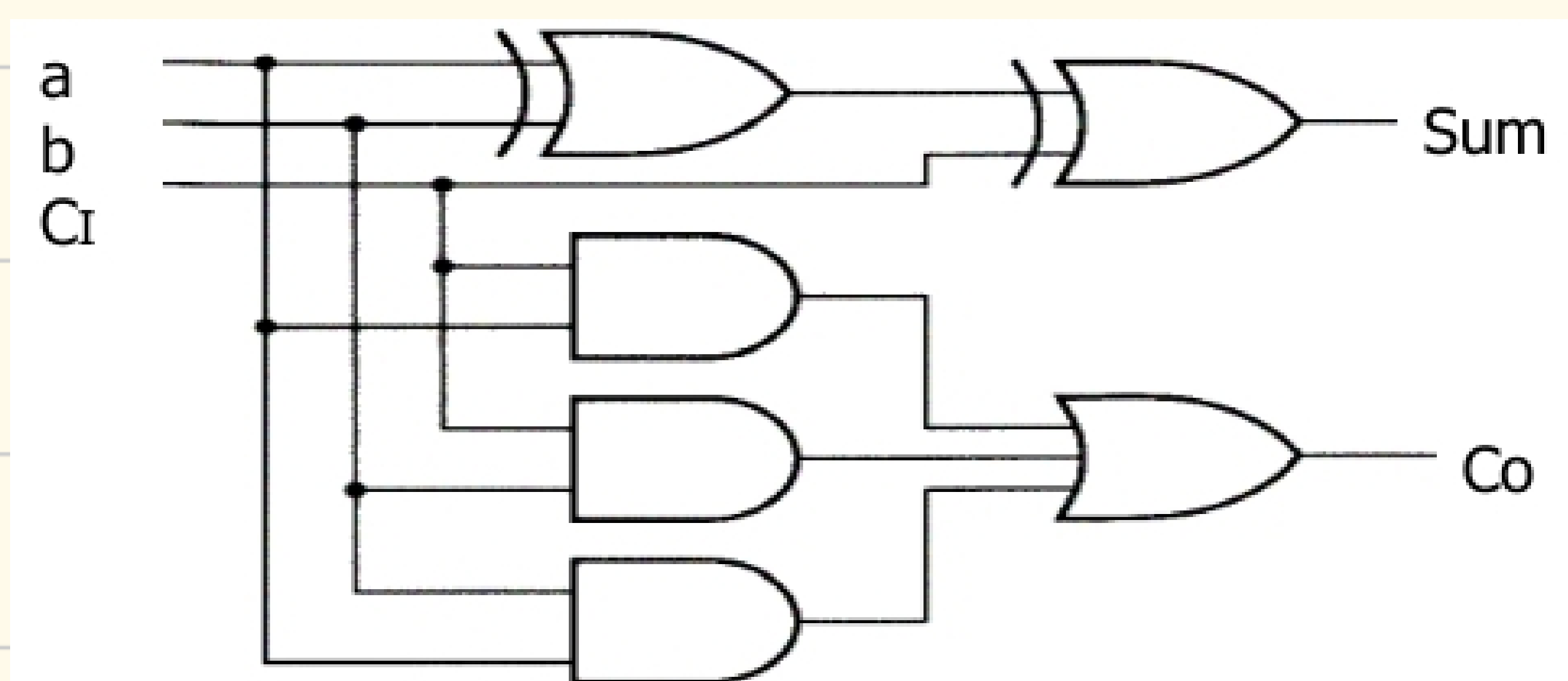
算术逻辑运算部件

全加器



$$Sum = a \oplus b \oplus C_{in}$$

$$C_{out} = ab + C_{in}(a+b)$$

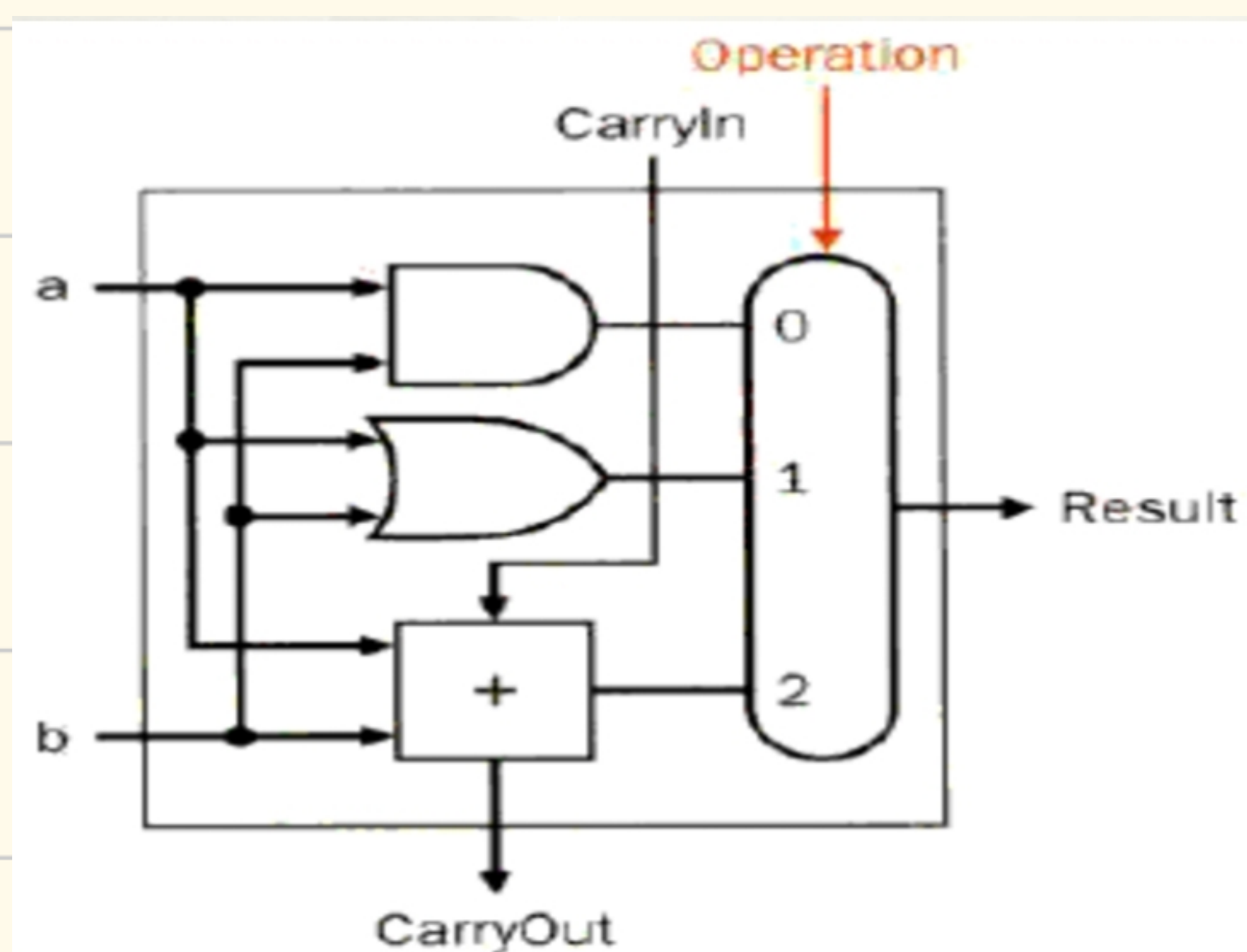


一位ALU

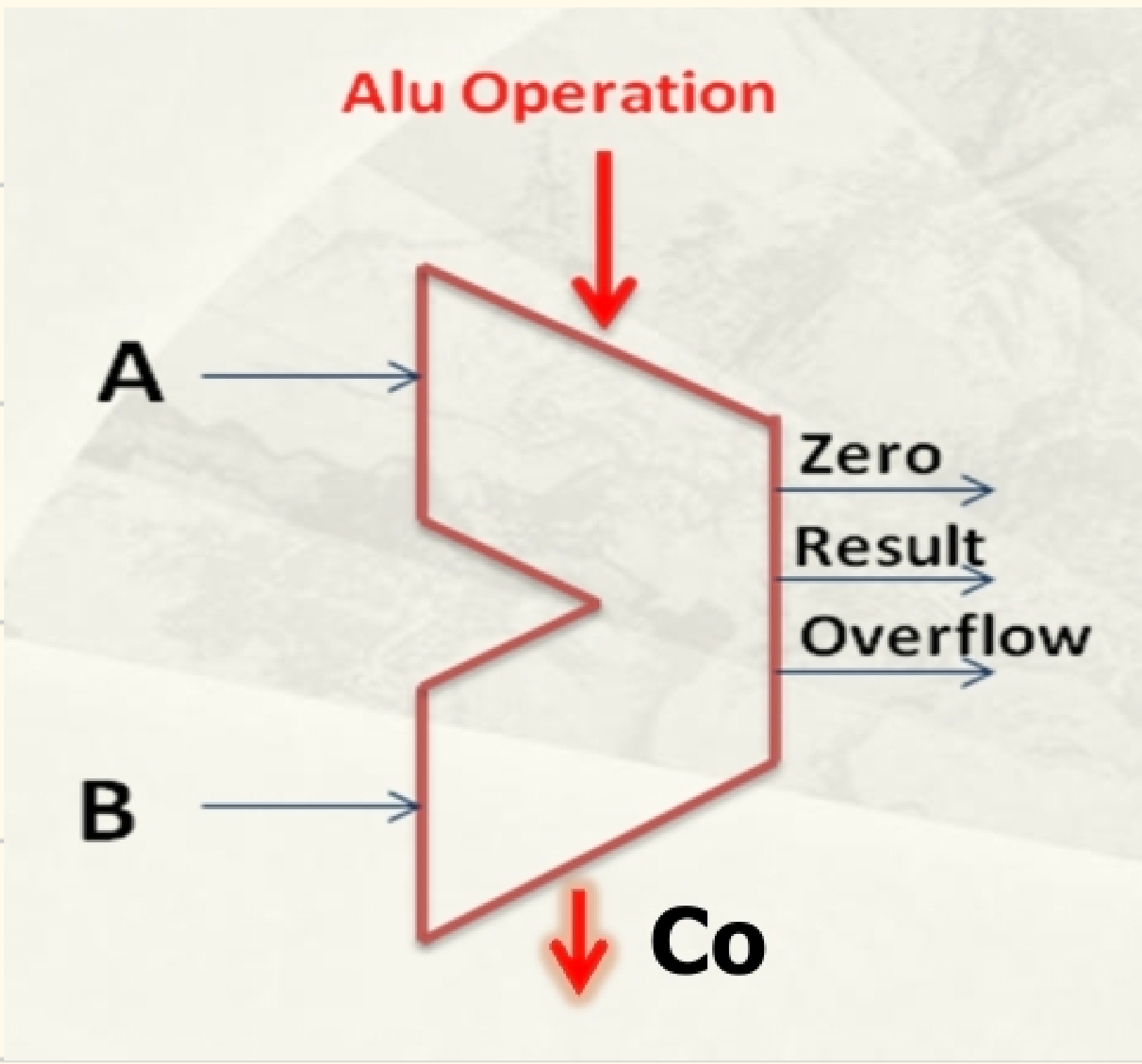
控制信号 00: AND

控制信号 01: OR

控制信号 10: ADD



ALU的符号与功能控制



功能控制信号

ALU 控制线 (Ainvert Bnegate Operation) 1位 1位 2位			功能
	0000		与
	0001		或
	0010		加
	0110		减
	0111		小于则置位
	1100		或非

并行加法器

进位链 $C_{i+1} = a_i b_i + (a_i + b_i) \cdot C_i$ ★

进位产生函数 $g_i = a_i b_i$

进位传递函数 $p_i = a_i + b_i$

$$C_{i+1} = \underbrace{g_i}_{\text{本地绝对进位}} + \underbrace{p_i \cdot C_i}_{\text{条件传递进位}}$$

串行进位、行波进位

并行进位、超前进位

$$C_{i+1} = g_i + p_i g_{i-1} + p_i p_{i-1} g_{i-2} + p_i p_{i-1} p_{i-2} g_{i-3} + \dots + p_i p_{i-1} \dots p_1 p_0 C_0$$

组内并行、组间并行

以16位为例

1) 第1组进位逻辑式

组内:

$$\begin{aligned} c_1 &= g_0 + p_0 c_0 \\ c_2 &= g_1 + p_1 g_0 + p_1 p_0 c_0 \\ c_3 &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \end{aligned}$$

组间:

$$c_4 = \underbrace{g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0}_{G_0} + \underbrace{p_3 p_2 p_1 p_0}_{P_0} c_0$$

所以 $\overset{P_0}{C_1} = \overset{G_0}{G_0} + P_0 c_0$

2) 第2组进位逻辑式

组内:

$$\begin{aligned} c_5 &= g_4 + p_4 c_1 \\ c_6 &= g_5 + p_5 g_4 + p_5 p_4 c_1 \\ c_7 &= g_6 + p_6 g_5 + p_6 p_5 g_4 + p_6 p_5 p_4 c_1 \end{aligned}$$

组间:

$$c_8 = \underbrace{g_7 + p_7 g_6 + p_7 p_6 g_5 + p_7 p_6 p_5 g_4}_{G_1} + \underbrace{p_7 p_6 p_5 p_4}_{P_1} c_1$$

所以 $\overset{P_1}{C_2} = \overset{G_1}{G_1} + P_1 C_1$

组间超前进位单元

$$\begin{aligned} C_1 &= G_0 + P_0 c_0 \\ C_2 &= G_1 + P_1 C_1 \\ &= G_1 + P_1 G_0 + P_1 P_0 c_0 \end{aligned}$$

$$\begin{aligned} C_3 &= G_2 + P_2 C_2 \\ &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0 \\ C_4 &= G_3 + P_3 C_3 \\ &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0 \end{aligned}$$