1. 怎么设计出指令集（指令格式），指令集怎么实现？

指令格式举例（3）

某机指令字长度为16位，包括基本操作码4位和3个地址段，每个地址段长4位，其格式为：

OP	A_1	A_2	A_3

1） 4位基本操作码若全部用于表示三地址指令，则共有多少条？

2）若三地址指令15条，二地址指令最多可有多少条？

3）若三地址指令、二地址指令和一地址指令各有15条，零地址指令16条，则共有61条指令。

指令格式举例（3）

0 0 0 0 X X X X X X X X X X X X

0 0 0 1 X X X X X X X X X X X X

⋮ ⋮ ⋮ ⋮

1 1 1 0 X X X X X X X X X X X X

} 15条三地址指令

1 1 1 1 0 0 0 0 X X X X X X X X

1 1 1 1 0 0 0 1 X X X X X X X X

⋮ ⋮ ⋮ ⋮

1 1 1 1 1 1 1 0 X X X X X X X X

} 15条二地址指令

指令格式举例（3）

1111 1111 0000 XXXX
1111 1111 0001 XXXX
⋮ ⋮ ⋮ ⋮
1111 1111 1110 XXXX

15条一地址指令

1111 1111 1111 0000
1111 1111 1111 0001
⋮ ⋮ ⋮ ⋮
1111 1111 1111 1111

16条零地址指令

指令格式举例（4）

设某指令系统指令字长16位，每个地址码为6位。若二地址指令15条，一地址指令34条，则剩下零地址指令最多有多少条？

OP(4)	A1(6)	A2(6)

解:操作码按短到长进行扩展编码

二地址指令: (0000 – 1110) 共15条 (不扩展时为16条)

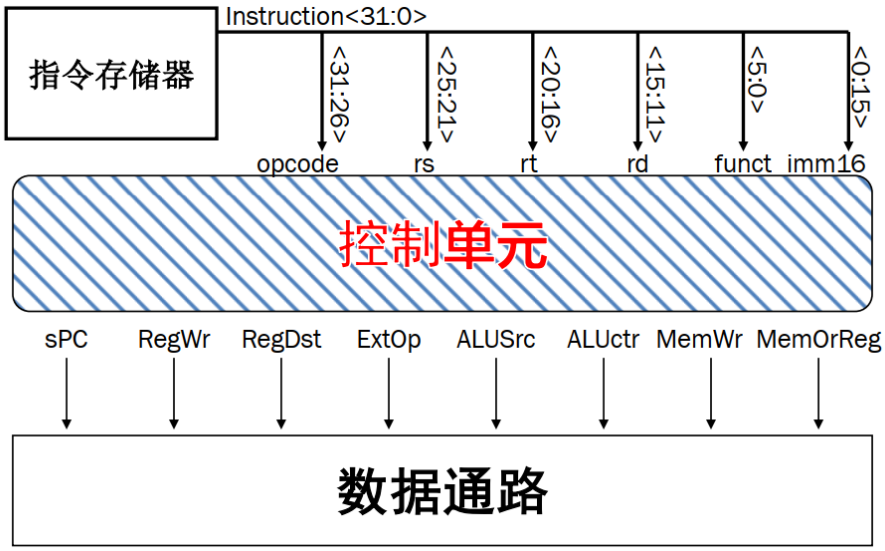
一地址指令: 1111 (000000 – 100001); (34条)

零地址指令: 1111 (100010 – 111111) (000000 – 111111)

(30种扩展标志)

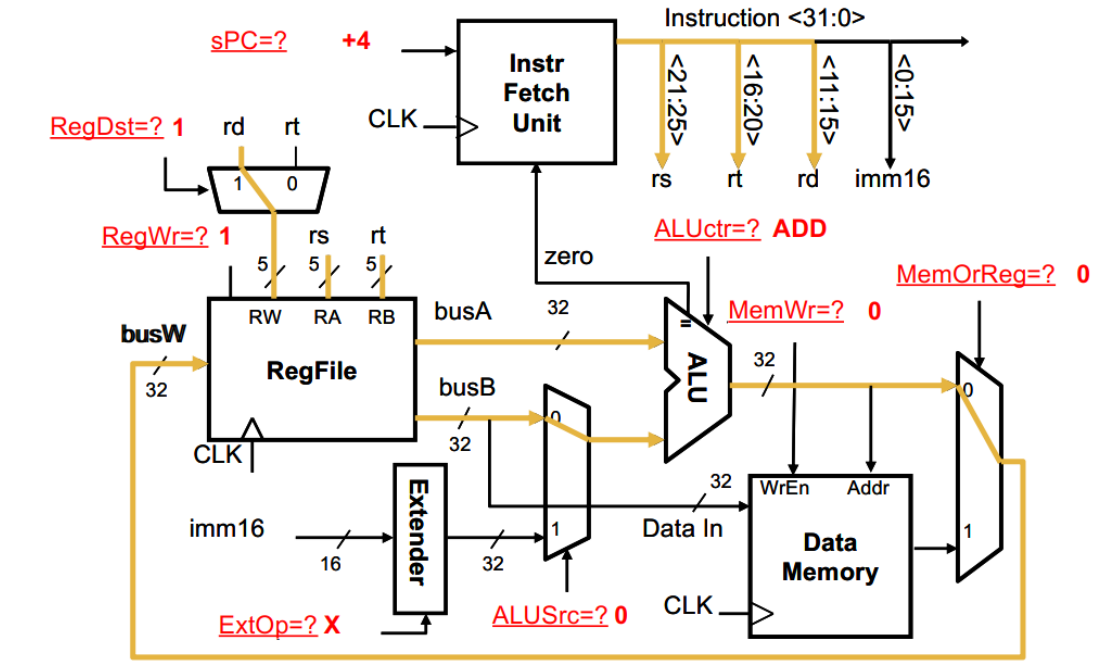
故零地址指令最多有$30\times2^6=15\times2^7$ 种

2. 取指、译码、执行、存储访问、写回，给几条指令，分析数据通路；控制器生成控制信号电路（与逻辑、或逻辑结合）

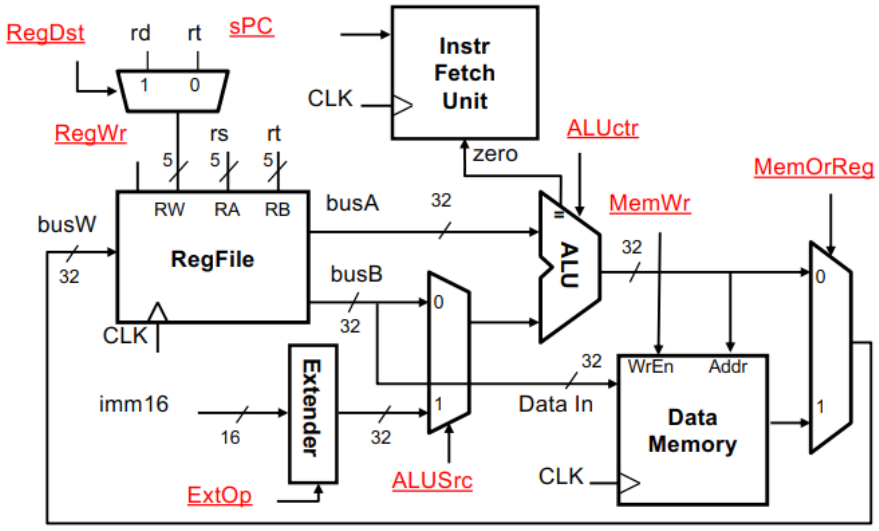


第 4 讲_部分 3-单周期 CPU 设计

R[rd]←R[rs]+R[rt];



- **ExtOp:** 0 → “zero”;1 → “sign”
- **ALUsrc:** 0 → busB; 1 → imm16
- **ALUctr:** “ADD”, “SUB”, “OR”
- **sPC:** 0 → +4; 1 → branch
- **MemWr:** 1 → write memory
- **MemOrReg:** 0 → ALU; 1 → Mem
- **RegDst:** 0 → “rt”; 1 → “rd”
- **RegWr:** 1 → write register



指令的控制信号

参考MIPS指令手册 → func / op

	ADD	SUB	ORI	LW	SW	BEQ
func	10 0000	10 0010	n/a			
op	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100
RegDst	1	1	0	0	X	X
ALUSrc	0	0	1	1	1	0
MemOrReg	0	0	0	1	X	X
RegWrite	1	1	1	1	0	0
MemWrite	0	0	0	0	1	0
sPC	0	0	0	0	0	1
ExtOp	X	X	0	1	1	X
ALUctr<1:0>	add	subtract	or	add	add	subtract

控制信号

ALU支持的操作）

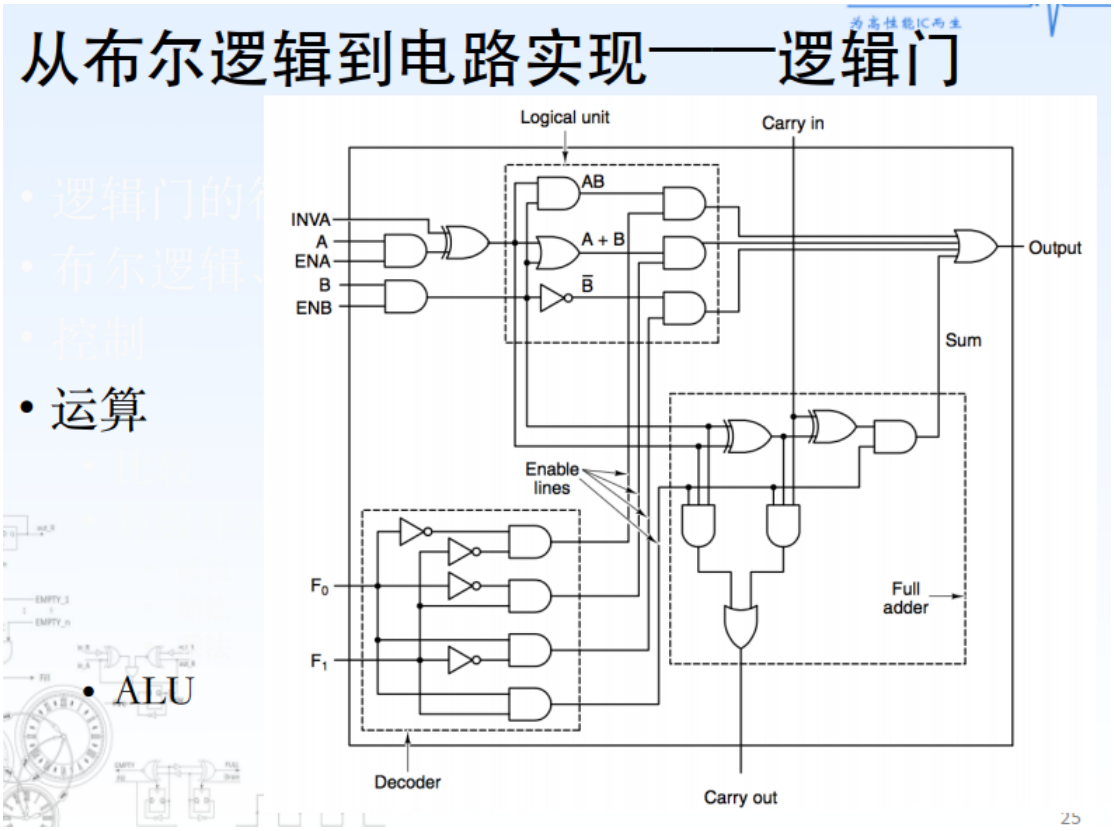
- 实现方式多种多样，这里进介绍一种简单直观方式
- 思路
  - 指令的OP和FUNC编码唯一确定指令——译码 译码（ID）阶段
  - 指令使能各控制信号
- 实现方式
  - 指令译码
    - I型和J型指令，只与OP相关
      - $BEQ = \overline{op[5]} \wedge \overline{op[4]} \wedge \overline{op[3]} \wedge op[2] \wedge \overline{op[1]} \wedge \overline{op[0]}$
    - R型指令
      - $Rtype = \overline{op[5]} \wedge \overline{op[4]} \wedge \overline{op[3]} \wedge \overline{op[2]} \wedge \overline{op[1]} \wedge \overline{op[0]}$
      - $ADD = Rtype \wedge (op[5] \wedge \overline{op[4]} \wedge \overline{op[3]} \wedge \overline{op[2]} \wedge \overline{op[1]} \wedge \overline{op[0]})$

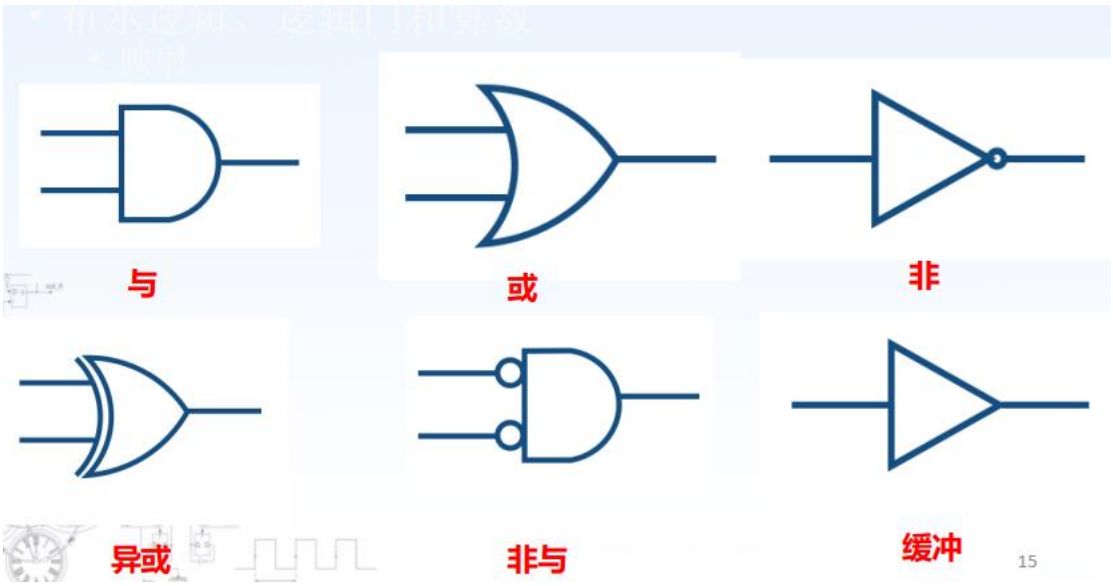
    与逻辑

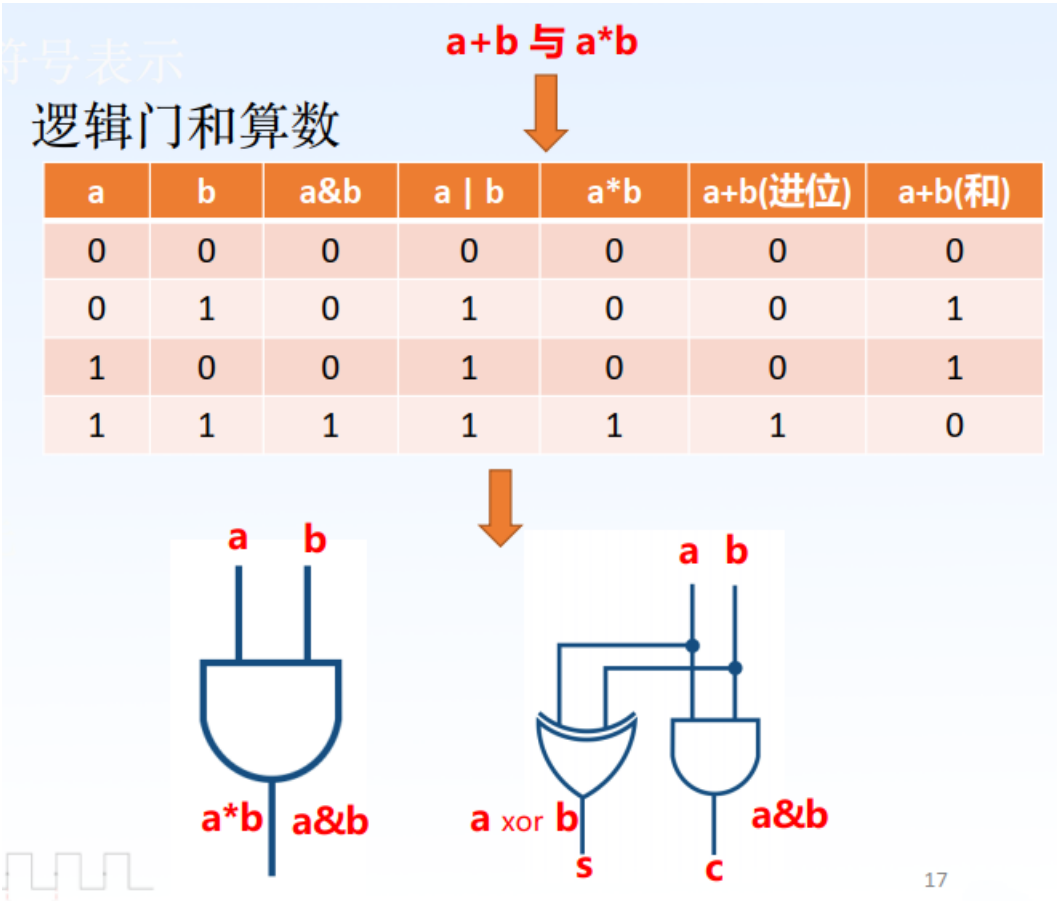
    为什么公式中不用考虑其他变量？
  - 指令控制
    - $MemWrite = SW$
    - $RegWrite = ADD \vee SUB \vee ORI \vee LW$

    或逻辑

3. 分析 ALU 功能，进行的什么运算（逻辑，算数）（给一张图，输入一个编码）



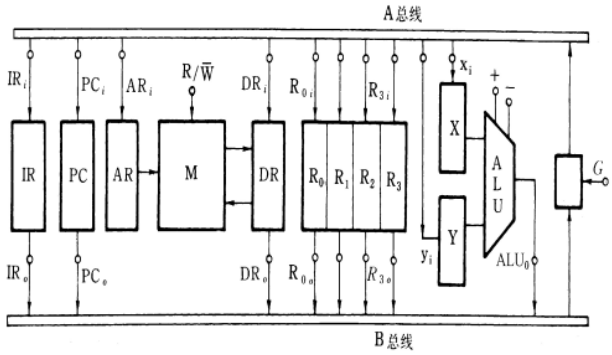




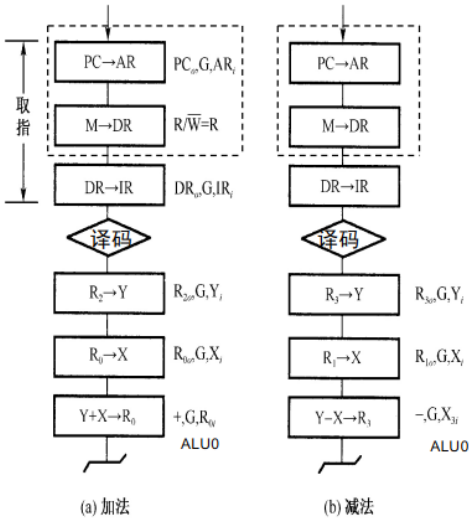
4. 总线型 CPU（单总线，双总线，多总线）一个周期内分布操作流程图。

总线型CPU

简单双总线CPU的数据通路



分步操作流程图



如果是单总线结构有何优劣？

41

5. 补码的计算

-128 的补码是 10000000。

一个正整数的补码和该数值的原码（即该数值的二进制形式）相同。

求负整数的补码的方法如下：

a）取该负整数的绝对值，并将该绝对值以二进制形式（原码）表示；

b）将上一步骤的二进制按位取反（包括符号位，0 变 1，1 变 0）；

c）对上一步骤的结果加 1。

例如，求-10 的补码的方法是：

a）取-10 的绝对值 10，并以二进制形式，即 10 的原码为 0000000000001010（此处按一个整数占 16 位计算，下同）；

b）对 1010 按位取反，得到二进制 1111111111110101；

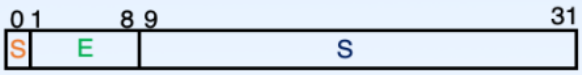
c）对上一步骤的结果再加 1，得到-10 的补码 1111111111110110。

6. 浮点数的计算

32位单精度浮点数（floating point）的规格化形式

- 第 0 位为符号位（数符），记为S
- 第 1～8 位为8位移码，表示指数，记为E
- 第 9～31 位为23位二进制原码小数表示的尾数M

浮点数：+/-1.xxxxxxxxxx$_{two}$ x $2^{Exponent}$



S：0表示正，1表示负

E：单精度数偏置量为127，双精度为1023

- 单精度规格化数的阶码范围为0000 0001 (-126) ~ 1111 1110 (127)
- 全0和全1用于表示特殊值

M：规格化尾数最高位总是1，所以隐含表示，省1位

- 尾数长度：1 + 23 bits （单精度）
- 尾数长度：1 + 52 bits （双精度）

值的范围：

- 单精度：$(-1)^S$ x (1 + Significand) x $2^{(Exponent-127)}$
- 双精度：$(-1)^S$ x (1 + Significand) x $2^{(Exponent-1023)}$

如何表示0？

- 指数：均为0
- 尾数：均为0
- 符号位：0/1均可
- +0：0 00000000 00000000000000000000000
- -0：1 00000000 00000000000000000000000

如何表示无穷大+∞/-∞？

- 指数：均为1
- 尾数：均为0
- 符号位：0/1均可
- +∞：0 11111111 00000000000000000000000
- -∞：1 11111111 00000000000000000000000

如何表示非数（NaN）？

- 指数：均为1
- 尾数：非0
- 符号位：0/1均可

Sqrt (- 4.0) = ?　　0/0 = ?

➢ 称为非数（NaN，Not a Number）

浮点数加减法基本要点

（假定：Xm、Ym分别是X和Y的尾数， Xe和Ye 分别是X和Y的阶码 ）

(1) 求阶差：Δe=Ye – Xe （若Ye > Xe，则结果的阶码为Ye)

(2) 对阶：将Xm右移Δe位，尾数变为 $Xm*2^{Xe-Ye}$（保留右移部分附加位）

(3) 尾数加减： $Xm*2^{Xe-Ye} \pm Ym$

(4) 规格化：

当尾数高位为0，则需左规：尾数左移一次，阶码减1，直到MSB为1

每次阶码减1后要判断阶码是否下溢（比最小可表示的阶码还要小）

当尾数最高位有进位，需右规：尾数右移一次，阶码加1，直到MSB为1

每次阶码加1后要判断阶码是否上溢（比最大可表示的阶码还要大）

阶码溢出异常处理：阶码上溢，则结果溢出；阶码下溢到无法用非规格化数表示，则结果为0

(5) 如果尾数比规定位数长（有附加位），则需考虑舍入（有4种舍入方式）

(6) 若运算结果尾数是0，则需要将阶码也置0。为什么？

尾数为0说明结果应该为0（阶码和尾数为全0）。

浮点数加法运算举例

例：用二进制浮点数形式计算 0.5 +(– 0.4375) = ?

0.4375=0.25+0.125+0.0625=0.0111B

解：$0.5 = 1.000 \times 2^{-1}$， $-0.4375 = -1.110 \times 2^{-2}$

对　阶: $-1.110 \times 2^{-2} \rightarrow -0.111 \times 2^{-1}$

加　减: $1.000 \times 2^{-1} + (-0.111 \times 2^{-1}) = 0.001 \times 2^{-1}$

左　规: $0.001 \times 2^{-1} \rightarrow 1.000 \times 2^{-4}$

判溢出: 无

结果为： $1.000 \times 2^{-4} = 0.0001000 = 1/16 = 0.0625$

问题：为何IEEE 754 加减运算右规时最多只需一次？

因为即使是两个最大的尾数相加，得到的和的尾数也不会达到4，故尾数的整数部分最多有两位，保留一个隐含的“1”后，最多只有一位被右移到小数部分。

1.计算X+Y，其中X=0.1010×2^{11}；Y=0.1101×2^{10}；要求：阶码、尾数以双符号位法表示，另阶码、尾数数值位分别取3位和6位，计算结果采用0舍1入法，尾数保留6位数字。

1解：

[X]补 = 00 011；00.101000　　[Y]补 = 00 010；00.110100

①对阶：

$[\triangle j]_{补} = [j_X]_{补} - [j_Y]_{补}$ = 00 011
\+ 11 101
= 1 00 001

阶差为+1，大于0

M_Y右移1位，E_Y+1

[Y]补 = 00 011；00.011010（0）

②尾数求和：

M_X = 00.101000
\+ M_Y = 00.011010（0）
01.000010（0）

尾数和溢出，需要右规

③右规：

[X+Y]补 = 00 011；01.000010（0）
= 00 100；00.100001（0）

即：X+Y = 00 100；00.100001（0）= 0.100001（0）×2^{100}

④舍入：0舍1入，保留6位

M_{X+Y} = 00.100001（0）= 00.100001

X+Y = 00 100；00.100001 = 0.100001×2^{100}

计算X-Y，其中X=（3/8）×2^{-3}；Y=（-5/8）×2^{-4}；要求：阶码、尾数以双符号位法表示，另阶码、尾数数值位分别取3位和4位，计算结果采用0舍1入法，尾数保留4位数字。

2解：

[X]补 = 11 101；00.0110　[Y]补 = 11 100；11.0110

①对阶：

$[\triangle j]_{补} = [j_X]_{补} - [j_Y]_{补}$ = 11 101
\+ 00 100
= 1 00 001

阶差为+1，大于0

M_Y右移1位，E_Y+1

[Y]补 = 11 101；11.1011（0）

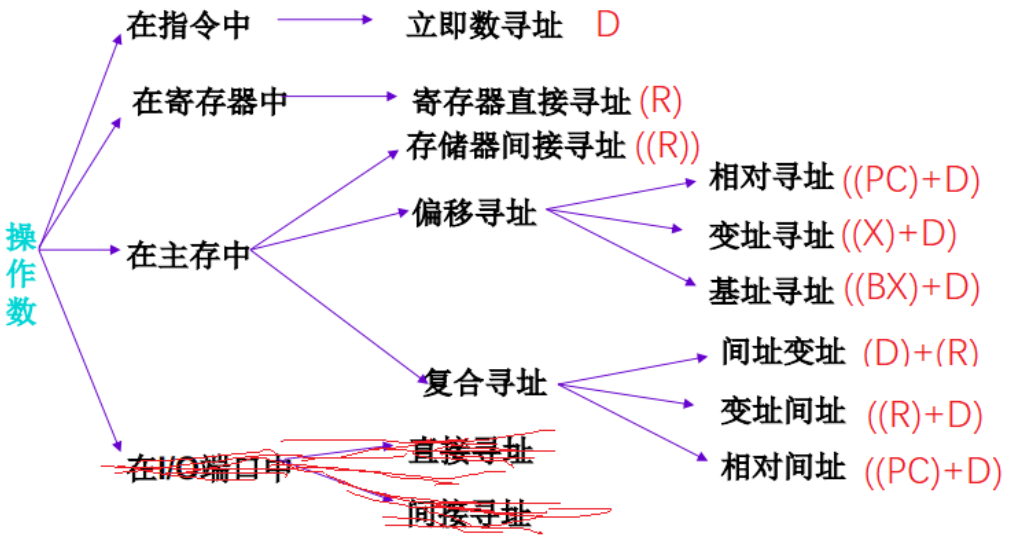
②尾数求差：

$[-M_Y]_{补}$ = 00.0101（0）
M_X = 00.0110
\+ $-M_Y$ = 00.0101（0）
00.1011（0）

尾数和未溢出，且无需规格化

7. 操作数寻址

第 4 讲_部分 2-指令系统.pdf



操作数寻址实例（1）

主存数据分布如图所示,若A为单元地址 (A)为A的内容,求

((7)) - (N) + ((N))+ (((N)))的值

0	9
1	11
2	22
3	53
4	44
5	3
6	2
7	0
	.
N	5

((7)) = 9

(N)= 5

((N)) = 3

(((N)))=53

结果 = 60

操作数寻址实例（2）

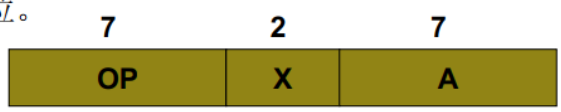
字长16位，主存64K，指令单字长单地址码，80条指令。寻址方式有直接、间接、相对、变址。请设计指令格式

解： 80条指令 ⇒ OP字段需要7位(2^7=128)

4种寻址方式 ⇒ 寻址方式特征位2位

16位字长指令还余7位作为地址位，由于是单地址指令，所以地址位的长度也是7位。

指令格式如下：



PC为16位　　变址寄存器16位

- 相对寻址　E=（PC）+D ,寻址范围为：64K
- 变址寻址　E=（R）+D，　寻址范围为：64K
- 直接寻址 E = D，寻址范围为128
- 间接寻址 E = （D），　寻址范围为64K

52

8. CPU 访存实例，给地址变换，怎么映射
   作业三
9. 命中关系，判断机制，是否都可命中

CPU访存：访存次数

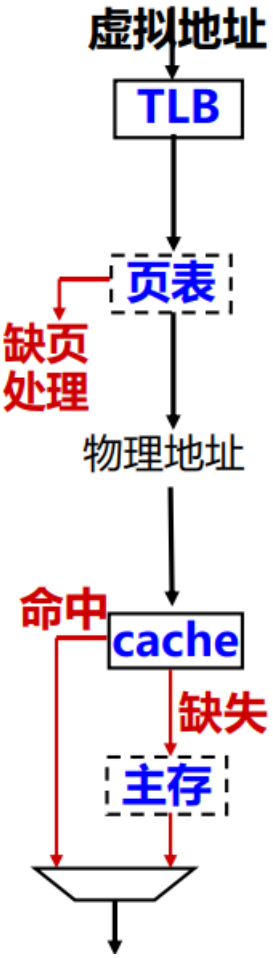
TLB	Page table	Cache	Possible? If so, under what circumstance?
hit	hit	miss	
miss	hit	hit	
miss	hit	miss	
miss	miss	miss	
hit	miss	miss	
hit	miss	hit	
miss	miss	hit	

最好的情况是hit、hit、hit，此时，访问主存几次？　不需要访问主存！

以上组合中，最好的情况是？ hit、hit、miss和miss、hit、hit　访存1次

以上组合中，最坏的情况是？ miss、miss、miss　需访问磁盘、并访存至少2次

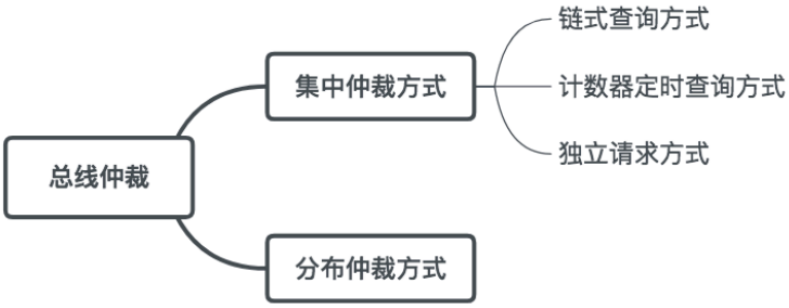
介于最坏和最好之间的是？ miss、hit、miss　不需访问磁盘、但访存至少2次



10. 描述题（系统互连）总线 工作原理、查询方式、仲裁、同步定时、异步定时

    连接到总线上的功能模块有主动和被动两种形态，其中主方可以启动一个总线周期，而从方只能响应主方请求。

    每次总线操作，只能有一个主方，但是可以有多个从方。



https://blog.csdn.net/qq_41587740

总线周期的四个阶段

1）申请分配阶段：由需要使用总线的主模块（或主设备）提出申请，经总线仲裁机构决定将下一传输周期的总线使用权授予某一申请者。也可将此阶段细分为**传输请求**和**总线仲裁**两个阶段。

2）寻址阶段：获得使用权的主模块通过总线**发出**本次要访问的从模块的**地址**及有关**命令**，启动参与本次传输的从模块。

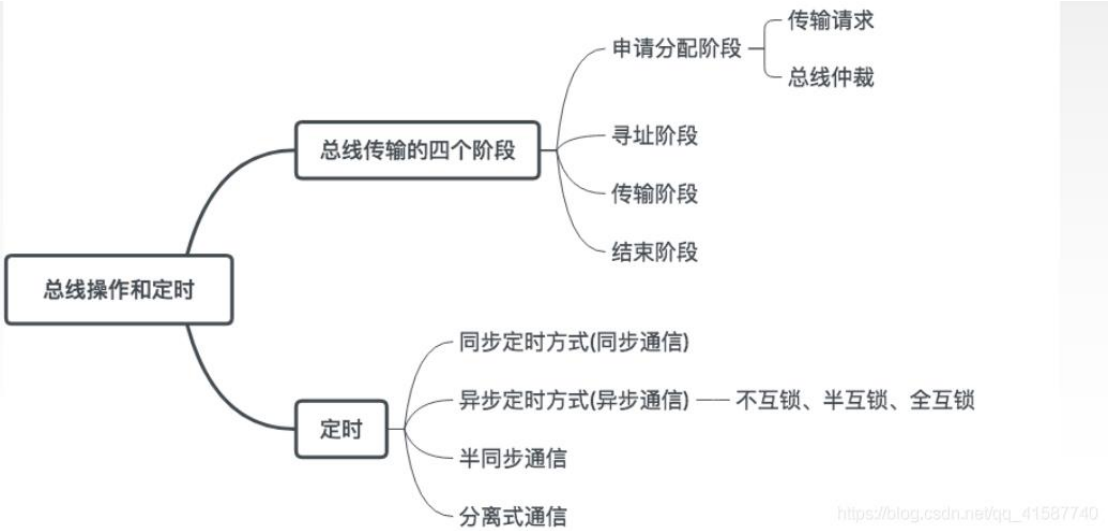
3）传输阶段：主模块和从模块进行**数据交换**，可单向或双向进行数据传送。

4）结束阶段：主模块的**有关信息**均从系统总线上**撤除**，让出总线使用权。

总线定时是指总线在双方交换数据的过程中需要时间上配合关系的控制，这种控制称为总线定时，它的实质是一种协议或规则

- 同步通信(同步定时方式)　由 统一时钟 控制数据传送
- 异步通信(异步定时方式)　采用 应答方式，没有公共时钟标准
- 半同步通信　同步、异步结合
- 分离式通信　充分 挖掘 系统 总线每瞬间 的 潜力

https://blog.csdn.net/qq_41587740



https://blog.csdn.net/qq_41587740

https://blog.csdn.net/qq_41587740/article/details/109183032
https://blog.csdn.net/qq_41587740/article/details/109184575

11. I/O 子系统（机制） 中断（多重中断屏蔽字，程序执行顺序（作业四）） DMA（简答 DMA 过程）

DMA 过程：

（1）外设可通过 DMA 控制器向 CPU 发出 DMA 请求：

（2）CPU 响应 DMA 请求，系统转变为 DMA 工作方式，并把总线控制权交给 DMA 控制器；

（3）由 DMA 控制器发送存储器地址，并决定传送数据块的长度；

（4）建立虚拟通道执行 DMA 传送；

（5）DMA 操作结束，发出 DMA 结束中断，把总线控制权交还 CPU。

12. 第六题，从这学期学的内容判断分析某种 CPU 结构的好处，有无缺陷（怎样让 CPU 更快）开放题