§ Chap 1. Computer Abstractions and Technology.

1. Eight Great Ideas.

1. Design for Moore's Law.
2. abstraction
3. common case.
4. parallelism
5. pipelining
6. prediction
7. Hierachy
8. redundancy.

---

6. Limits of Processors.

- Memory Wall. Multiprocessors.
- Power Wall.

$$P = \frac{1}{2} C \cdot V^2 \cdot f$$

7. Fallacies and Pitfalls.

(1) Amdahl's Law: ($T = T_1 + T_2$).

$$T' = \frac{T_1}{\lambda} + T_2$$

(2) MIPS. same ISA. same program.

5. Performance.

- Execution Time.

$$\text{Performance} = \frac{1}{\text{Execution Time}}.$$

- CPU Time. CPU Clock Cycles. Clock Rate. Clock Time.

$$\text{Clock Rate} = \frac{1}{\text{Clock Time}}.$$

$$\text{CPU Time} = \text{CPU Clock Cycles} \cdot \text{Clock Time}$$

$$= \frac{\text{CPU clock cycles}}{\text{Clock Rate}}.$$

- Instruction Count (IC).

CPI (Clock cycles per instruction) (avg).

$$\text{CPU clock cycles} = IC \cdot CPI.$$

- $$MIPS = \frac{\text{Instruction Count} \times 10^{-6}}{\text{Execution time}} = \frac{\text{Clock Rate}}{CPI \times 10^6}.$$

(4) Making Decision.

Conditional

inst rs1, rs2, L1 ⤵

inst rs1, rs2, imm → jump to PC+imm. (地址)

经过的指令数为 imm/4.

beq. rs1=rs2 ? jmp.

bne. != ? jmp.

blt (u) < ? jmp.

bge (u). >= ? jmp.

Unconditioned.

jal reg, L1) caller

jal reg, imm (addr.

jalr reg, imm (base_reg). callee.

offset.

jal: jump and link. reg(x1) = PC + 4.

x0. (goto).

jalr: jump and link register;

4. Instruction Representations.

Chap 2. Language of the Machine.

1. Basics

ISA. RISC. CISC.

stored-program.

* Simplicity favors regularity.

Smaller is faster

Good design demands good compromises

Make common case fast

2. Registers.

RISC-V: 32. 64 bits. (doubleword.) * 32. x0 - x31.

detail ↩.

Spilling register.

3. Instructions.

(1) Arithmetic.

```
add  reg1, reg2, reg3. ; reg1 = reg2 + reg3.
addi reg1, reg2, const ; (subi).
sub  reg1, reg2, reg3.
```

(2) logical.

```
sll(i)
srl(i)
sra(i). ← 负数右移补1.
and (i)
or  (i)
xor (i).
```

not: xori reg1, reg2, $(11111111)_2$.

(3). Data Transfer.

通常按 doubleword 对齐. 可以不对齐但拼接

little-endian.

(byte).

load. reg, offset(mem_base_addr).

; reg = mem_base_addr [offset / width].

ld.	8	
lw. (u)	4	
lh. (u)	2	sign/zero extension
lb. (u)	1	

store. reg, offset(mem_base_addr).

; mem_base_addr [offset / width] = reg;

sd. sw. sh. sb.

从低位读.

(4) Making Decision.

Conditional

inst rs1, rs2, L1 ⤵

inst rs1, rs2, imm → jump to PC+imm. (把此处)

经过的指令数为 imm/4.

beq.	rs1=rs2	? jmp.
bne.	!=	? jmp.
blt (u)	<	? jmp.
bge (u).	>=	? jmp.

Unconditioned.

jal reg, L1 ↙ addr.) caller

jal reg, imm.

jalr reg, imm (base_reg).

offset. callee.

jal: jump and link. reg(x1) = PC + 4.

x0. (goto).

jalr: jump and link register;

4. Instruction Representations.

machine language, machine code

inst. 32bit (a word). field.

↙

opcode. funct3. funct7, rd, rs1. rs2. imm.

Name	Field.					
	7	5	5	3	5	7
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB-type	im[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ-type	imm [20.10:1,11,19:12]				rd	opcode
U-type	imm [31:12]				rd	opcode

R-type: arith & logic (inst rd. rs1, rs2)

I-type: immed arith & logic. (inst rd. rs1, imm)

ld, jalr. (inst, rd, imm(rs1))

S-type: store. (inst. rs2, imm(rs1))

SB-type: conditional (inst. rs1, rs2. imm[12:1])

(imm[0] = 0)

UJ-type: jal (inst rd. imm[20:1]).

U-type: lui (inst rd. imm[31:12]).

5. Addressing.

Addressing mode

immediate addressing

register addressing

base or displacement addressing

PC-relative addressing

Decoding Machine Language.

Table ↗.

6. Procedures.

(1) procedure (subroutine)

1. ~~传~~ args → regs
2. jal
3. ld ...
4.
5. result → regs.
6. jalr

parameter regs: x10 ~ x17

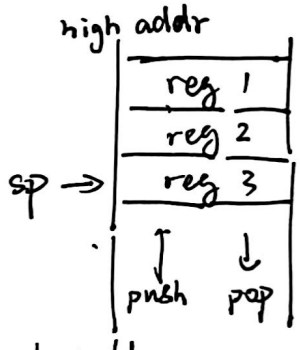
return value reg ← return ~~value~~ addr reg: x1 (ra).

PC. (mod 4).

procedure call inst: jal. jalr.

spilling reg.

(2) Stack



sp (x2).

push:

```
addi sp, sp, -8
sd   x5, 16(sp)
sd   x6, 8(sp)
sd   x7, 0(sp).
```

pop:

```
ld   x7   0(sp)
ld.  x6   8(sp)
ld   x5   16(sp)
addi sp, sp, 24.
```

Temporary registers caller 负责恢复

Saved registers. callee 负责恢复.

Preserved	Not Preserved
Saved reg: x8-x9, x18-x27	Temp reg: x5-x7, x28-x31
x2(sp)	Arg reg: x10-x17.
x8(fp)	
x1(ra)	
Stack above sp.	Stack below sp.

(3) Nested Procedures

leaf procedure.

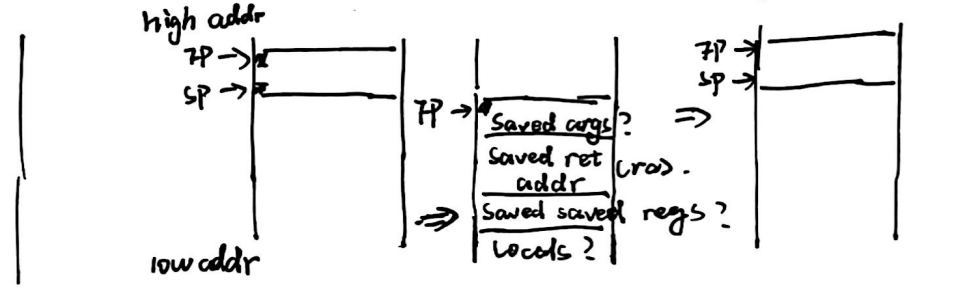
main → A → B. x1 push.
(x1) (x1')

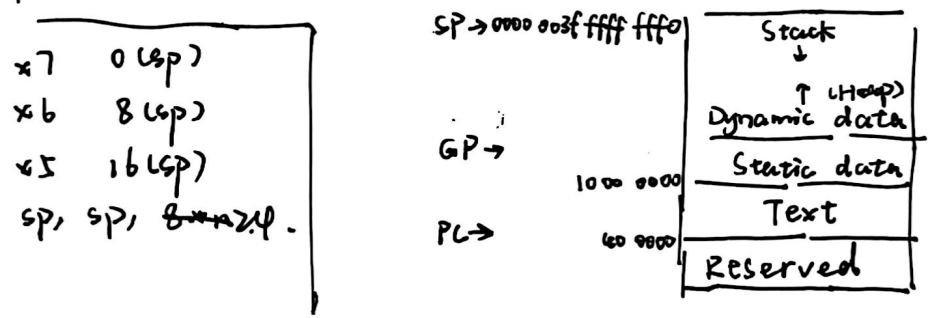
tail recursion → iteration

(4) Space Allocation

procedure frame / activation record.

x8(fp). frame pointer 帧(栈底),





§ chap 3. Arithmetic for Computer

overflow:

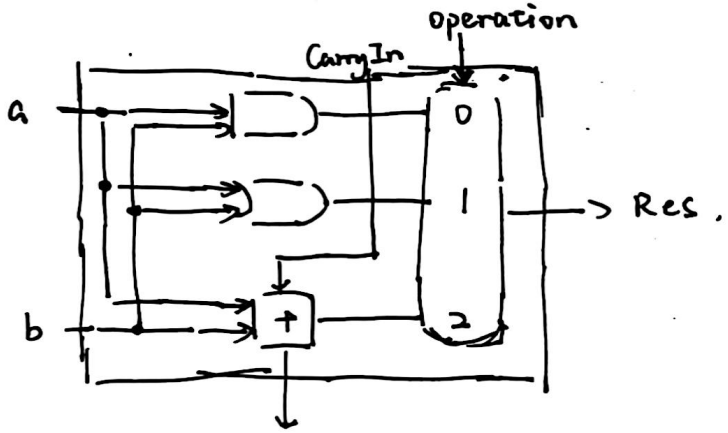
~~Un:Signed~~ Unsigned: carry/ borrow.

Signed:

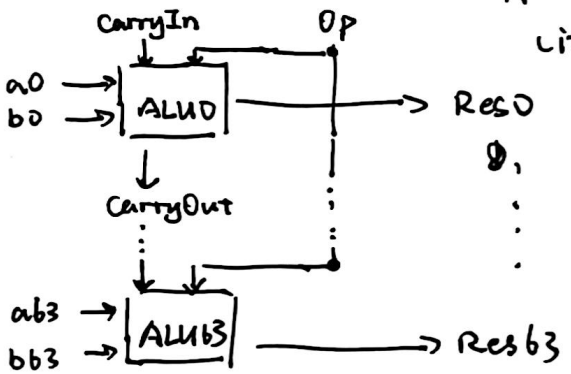
Op	A	B	Res indicating overflow.
A+B	≥0	≥0	<0
A+B	<0	<0	≥0
A-B	≥0	<0	<0
A-B	<0	≥0	≥0

ALU检测. 抛异常. EPC. OS例程.

constructing a Basic ALU



ripple carry adder (iterative circuit)



减法: 添加 Binvert. 将 carryIn 置 1. 视为 Bnegate

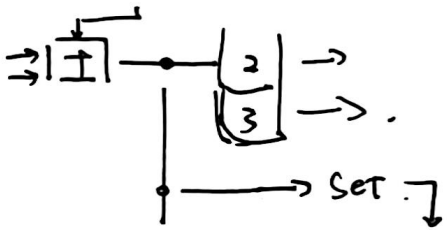
或非: 添加 Ainvert. 使用与运算. (同时可以与非)

slt: Less 当一mux[4]

↑ a<b a-b 的符号位.

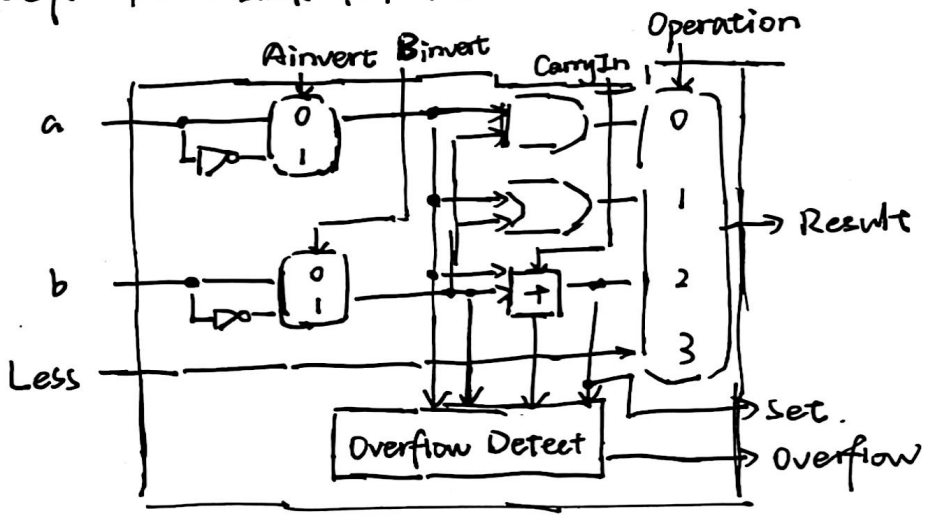
从最高位 ALU 加法器进位输出引出 Set,

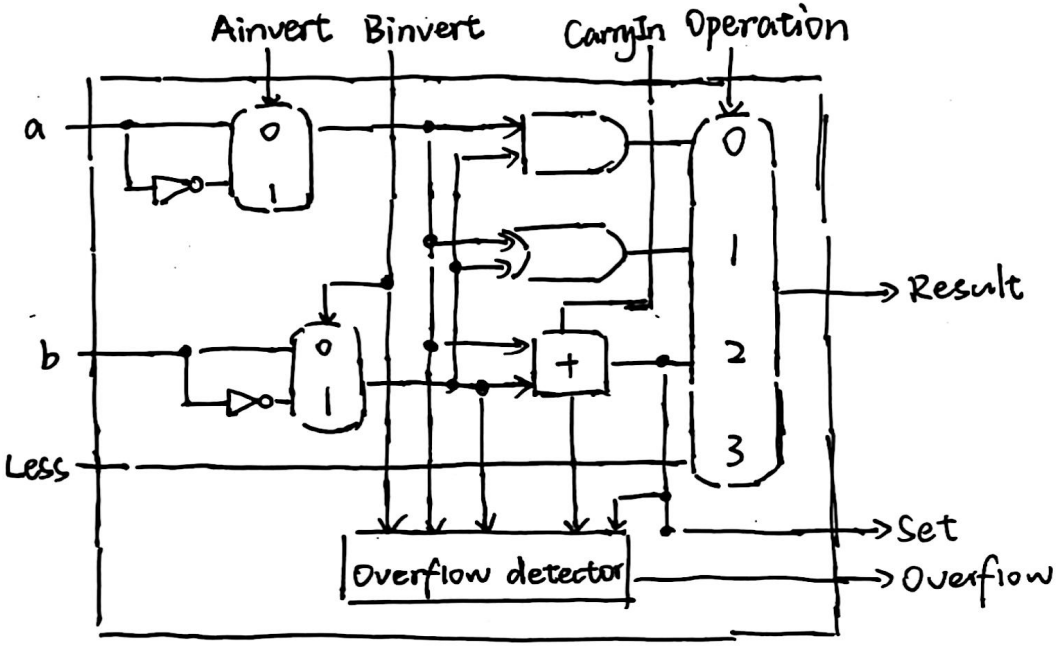
连到最低位ALU的输入脚.



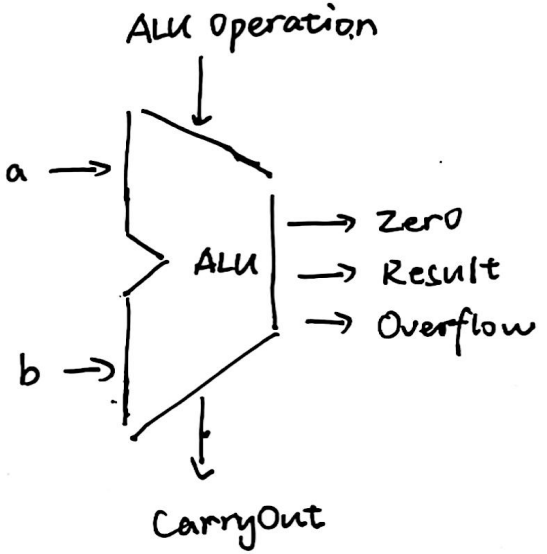
溢出: Overflow detection.

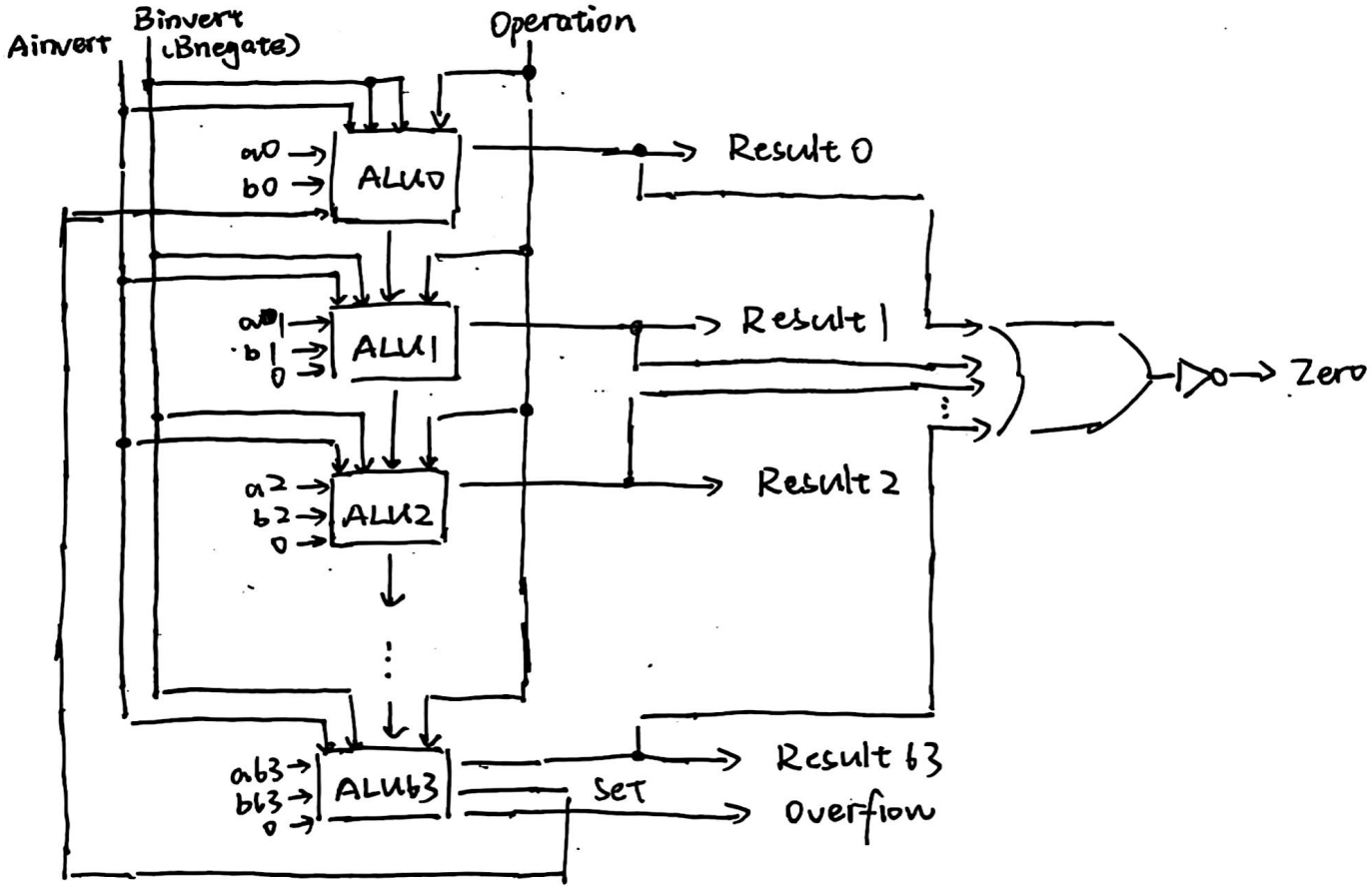
beq: 对 a-b 结果进行或非.





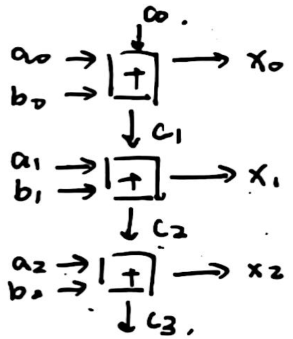
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	substract
0111	set less than
1100	NOR





Fast Addition: Carry Lookahead

Carry Lookahead Adder. $O(N) \to O(\log N)$



$$c_{i+1} = \underbrace{(a_i \cdot b_i)}_{g_i} + \underbrace{(a_i + b_i)}_{p_i} c_i$$

$$c_{i+1} = g_i + p_i c_i$$

e.g.:

$$c_5 = g_4 + p_4 g_3 + p_4 p_3 g_2 + p_4 p_3 p_2 g_1 + p_4 p_3 p_2 p_1 p_0 c_0$$

High-level abstraction:

$P_0 = p_3 \cdot p_2 \cdot p_1 \cdot p_0$ $\quad G_0 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0$

$P_1 = p_7 \cdot p_6 \cdot p_5 \cdot p_4$ $\quad G_1 = g_7$

$P_2 = p_{11} \cdot p_{10} \cdot p_9 \cdot p_8$ $\quad G_2 = g_{11}$

$P_3 = p_{15} \cdot p_{14} \cdot p_{13} \cdot p_{12}$ $\quad G_3 = g_{15}$

$C_1 = G_0 + P_0 \cdot c_0$

$C_2 = G_1 + P_1 \cdot G_0 + P_1 P_0 c_0$

$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 c_0$

$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 c_0$

To c_{15}.

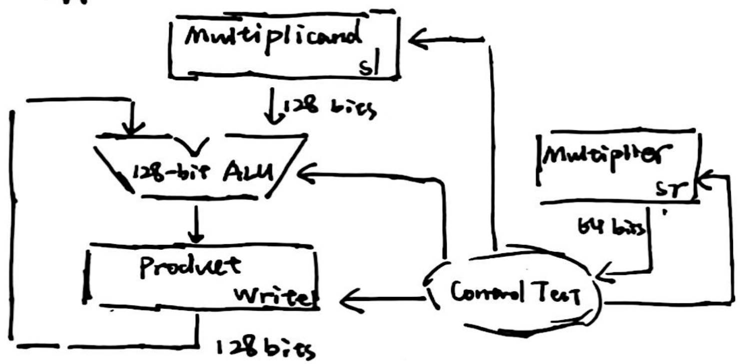
e.g.

	3	2	1	0	
A =	1111	1111	1111	1111	$c_0 = 0$
B =	0000	0000	0000	0001	
p	1111	1111	1111	1111	
g	0000	0000	0000	0001	parallel
P_i	1	1	1	1	
G_i	0	0	0	1	
	$C_4 = 1$	$C_3 = 1$	$C_2 = 1$	$C_1 = 1$	$c_0 = 0$

Multiplication.

```
     1011
  x  1101
     1011
    0
   1011
  1011
 10001111
```

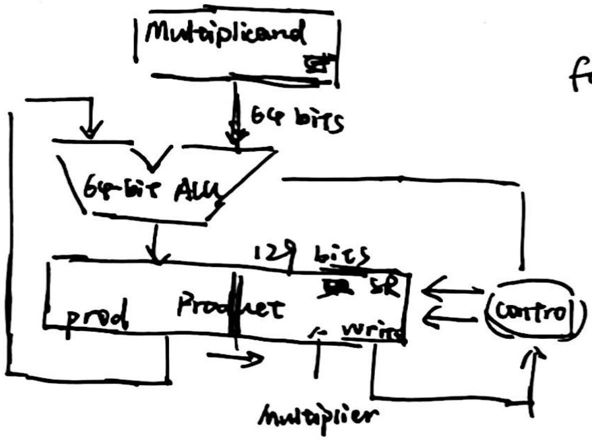
V1:



for i in range(64):
- a := LSB of Multiplier
- if a: product += multiplicand
- multiplicand <<= 1
- multiplier >>= 1

多: ALU Reduce (Adder Tree)

V2&3: 不移动 multiplicand. 移动 product



for i in range 64:
- a := LSB of Multiplier
- if a:
  - prod += multiplicand
  - Product >>= 1.

i64 × i64 → i128: mul, mulh.

mul. mulh. mulhu. mulhsu.

```
  0101
x 0111
 0 0111
  0101
 101011
2 101
  111011
  111101
3 01.01
  10001110
  1000110      → 100011
```

Signed Multiplication

Booth Algorithm.

$Z = y \times 10111100$

$= y \times (2^7 + 2^6 - 2^2)$

$= y \times 2^7 + y \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + -y \times 2^2 + 0 \times 2^1 + 0 \times 2^0$.

(add: 01; shift: 11; sub: 1; shift: 00.)

10 : sub

11 : shift

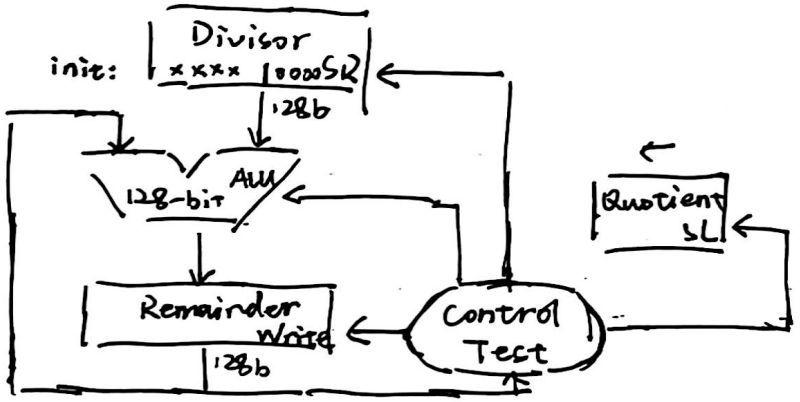
01 : add

0010 × 1101 = 1111 1010.

Iteration	step.	Multiplicand	Product
0	initial	0010	0000 1101 0
	1. -Mcand	0010	1110 1101 0
1	2. SR.	0010	1111 0110 1
2	1. +Mcand	0010	0001 0110 1
	2. SR.	0010	0000 1011 0
3	1. -Mcand	0010	1110 1011 0
	2. SR	0010	1111 0101 1
4.	SR	0010	1111 1010 1

Division

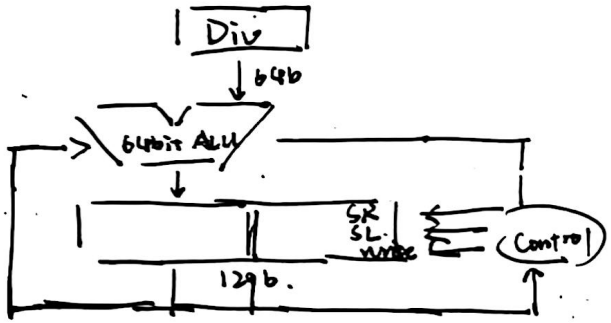
Trivial :



Divisor ← divisor 0000...0.

```
for i in range(65):
    R -= D.
    if R < 0:
        R += D, SLL Q, Q0 = 0.
    elif R >= 0
        SLL Q. Q0 = 1.
    D >>= 1.
```

Improved: (Structure like Mul).



//TODO.

Signed Division

符号位提取，执行无符号除，最后决定：同0异1.

余数符号：

同被除数. 绝对值不超过除数的绝对值.

Fast Division

SRT Division.

RISC-V.

div. divu. rem. remu.

5. Floating Point Numbers (IEEE-754)

$1.xxxxxxx \cdot 2^{yyyy}$.

Sign. Exponent. Fraction. $(-1)^S \times F \times 2^E$.

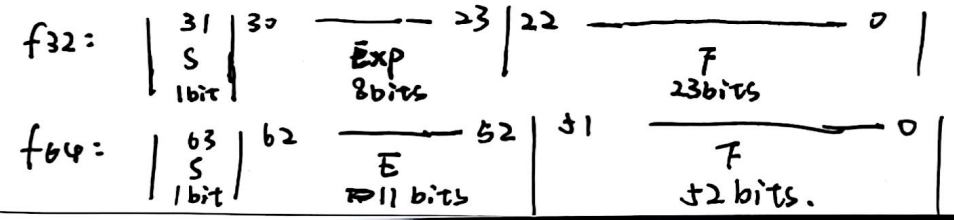
Hidden 1.

$(-1)^S \times (1+F) \times 2^E$.

Biased Notation. $Bias = 2^{(\text{Num of Exp bits} - 1)} - 1$.

f32: 127. f64: 1023.

$$(-1)^S \times (1+F) \times 2^{(E-Bias)}$$



Exception.

Overflow, Underflow.

NaN. (not a number).

NaN propagation. (NaN+5).

Indeterminate. ~~$\frac{0}{0}$~~ $\frac{\infty}{\infty}$ $\frac{\infty}{-\infty}$ $\frac{-\infty}{\infty}$ $\frac{-\infty}{-\infty}$.

Div. $\frac{0}{0}$ $\frac{\infty}{\infty}$ $\frac{\infty}{-\infty}$ $\frac{-\infty}{\infty}$ $\frac{-\infty}{-\infty}$.

Mod. ~~$a\%0$~~ $\infty\%a$.

Mul. $0 \times \infty$

Add. $\infty + (-\infty)$ $\infty - \infty$.

Exp. 0^0. ∞^0. 1^∞. $\infty^{-\infty}$.

Out of domain. $\sqrt{-1}$. $\log(-1)$.

Inf: $+\infty$ 0 11111111 000 $\rightarrow$
$-\infty$ 1 11111111 000 $\rightarrow$ 0.

NaN: ? 11111111 ??? $\rightarrow$?

Float Addition. // TODO.

1. Alignment
2. Addition (significands)
3. Normalization
4. Rounding: (maybe needs norm).

Float Multiplication.

1. $E = E_1 + E_2 - Bias$.
2. Mul significands.
3. Norm (& Overflow Test).
4. Rounding. (maybe needs norm)
5. Set sign. $(S_1 \otimes S_2)$.

Accurate Arithmetic.

Guard. Round. (Sticky).

1. $m_1\ m_2 \cdots m_p$ | G R $\underbrace{s_1 s_2 \cdots s_n}$, OR.

$$S = \sum_{i=1}^{n} s_i$$

Round to nearest even

向上取整. 向下取整. 截断取整.

Chap 4: The processor.

1. Single Cycle. ISA. compiler. CPU!

Perf: Instruction Count. Clock Cycle Time. CPI.

Single cycle CPU. 1. datapath. 2. control unit.

Datapath.

memory-ref inst: ld. sd. —.

arith-logical inst: add. sub. and. or. —.

conditional branch inst: beq. —.

(I. unconditional. ...).

取指(IF) PC fetch.

译码(ID). 识别指令类型,(读取寄数据).

执行(Ex).

访存(MEM). 写回(WB).

memref:
- ld: MEM+WB. (ld 五步, 最长)
- sd: MEM. (最快)

arith-logical: WB ALU.

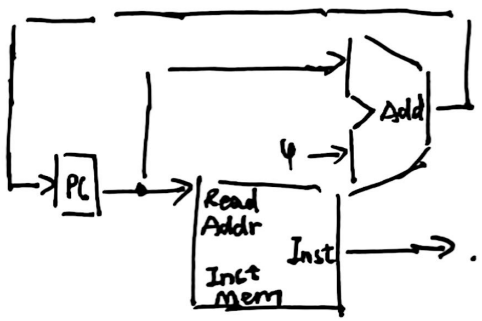
conditional: ∅.

unconditional: WB(x1).

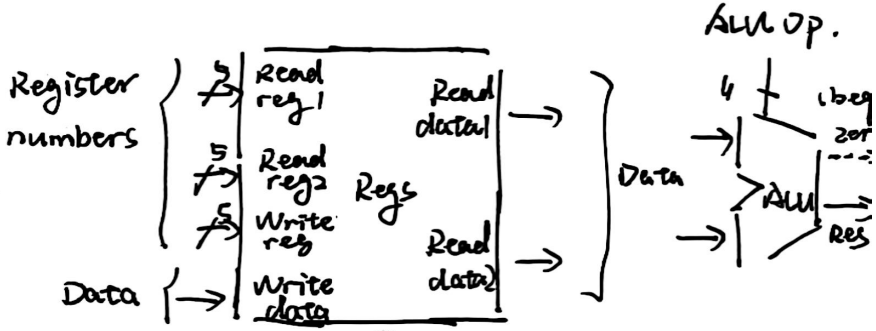
v1 线路有汇合. + MUX.

(时序电路 上有沿触发).

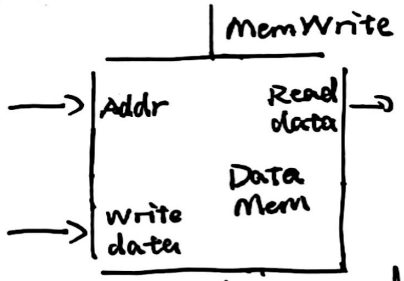
IF



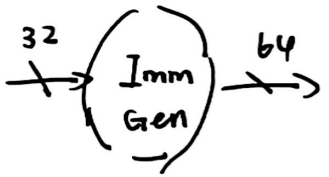
ID.



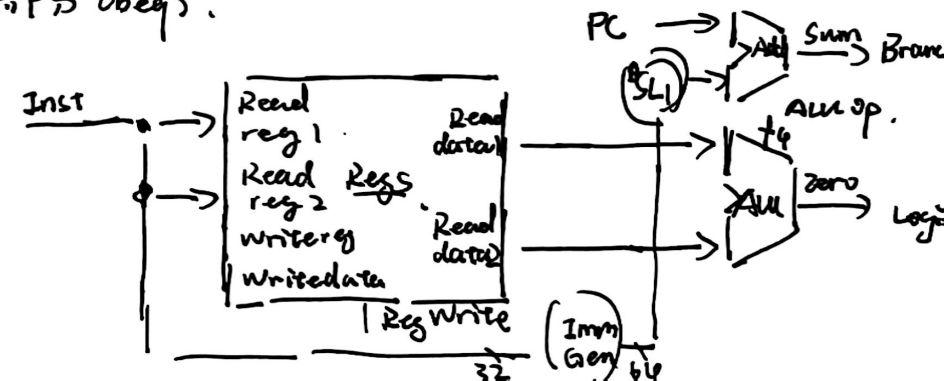
MEM.



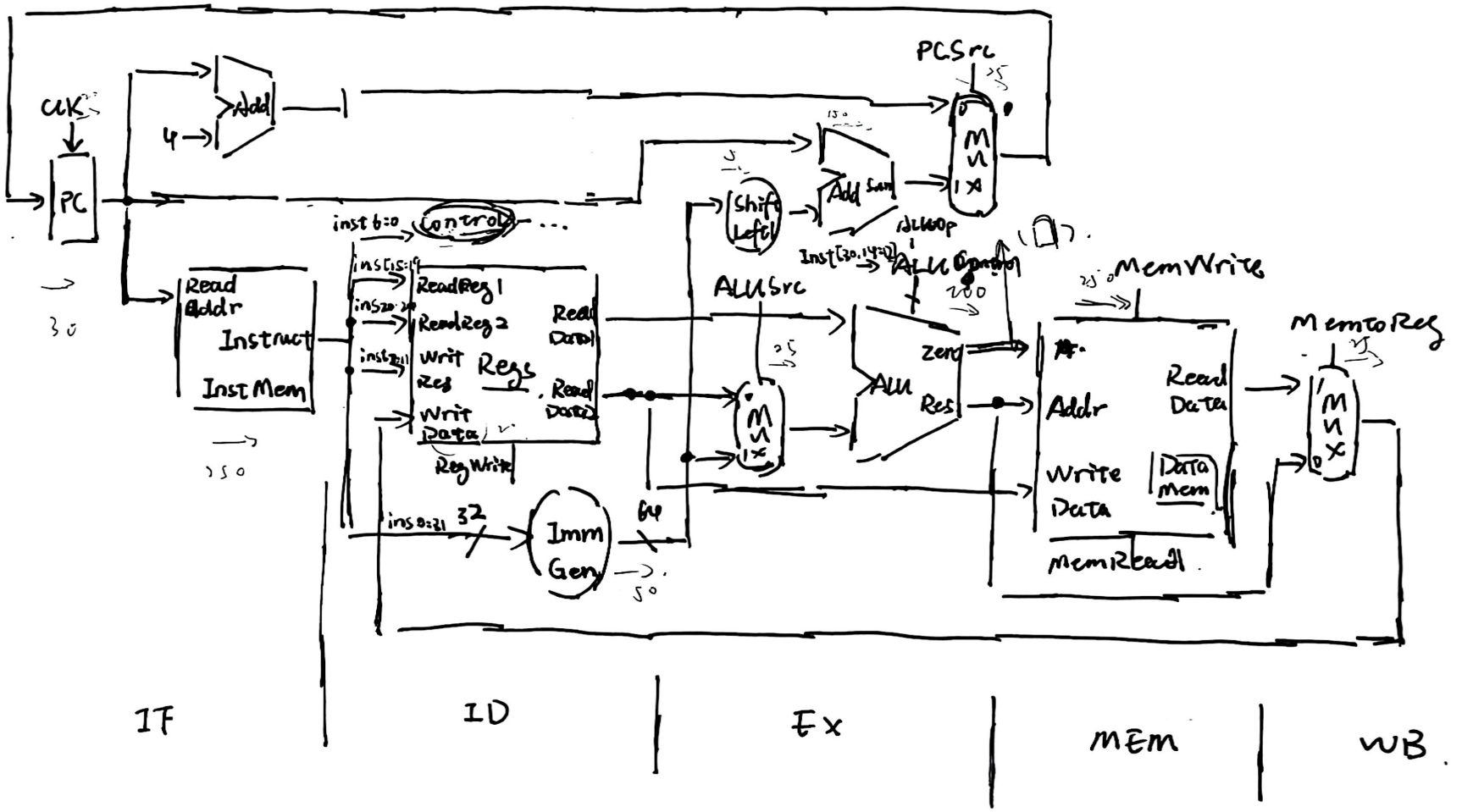
★ 符号扩展.



判条转移 (beq).



Composition.



Control Unit.

The ALU Control.

4 bit. funct3. funct7. ALUOp..
(2)

00: load & save.
01: condi branch
10: R

Main Control Unit

RegWrite. MemRead. MemWrite.

ALUSrc. PCSrc. Mem to Reg (2)..