Register	ABI Name	Description	Saver
x0	zero	Hard-wire zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	thread pointer	—
x5-7	t0-2	Temporaries	Caller
x8	s0/fp	Saved Reg/frame pointer	Callee
x9	s1	Saved reg	Callee
x10-11	a0-1	Function arguments Return values	Caller
x12-17	a2-7	Function arguments	Caller
x18-27	s2-11	Saved reg	Callee
x28-31	t3-6	Temporaries	Caller

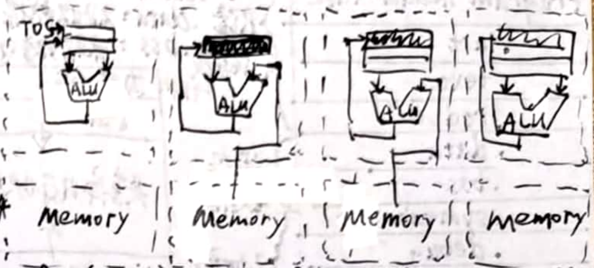
f... 省略

- Caller 在调用子函数时不会保存，会被重写（返回新值）
- Callee 在调用时会保存不变，无法返回新值。

	31–25	24–20	19–15	14–12	11–7	6–0
R	funct7	rs2	rs1	funct3	rd	opcode
I	imm[11:0]		rs1	funct3	rd	opcode
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
B	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
U	imm[31:12]				rd	opcode
J	imm[20|10:1|11|19:12]				rd	opcode

ISA { stack 无用操作数; accumulator 一个操作数; APR { Register-Memory (RM)：直接访问内存，Intel 8086; Load-Store (RR)：只有 ld, sd 载入，ARM, MIPS } }

slt(i)
slt x5, x6, x7
if(x6<x7) x5=1
else x5=0
sltu sltui



T_p: n=指令条数, m=depth　$T_p = \frac{n}{m\Delta t + (n-1)\Delta t}$　$T_{p max} = \frac{1}{\Delta t}$

单位时间流水线所完成指令或输出结果的数量

Sp: m段的流水线速度与等功能非流水线之比

$S_p = \frac{n \cdot m \Delta t}{m\Delta t + (n-1)\Delta t} = \frac{nm}{m+n-1}$　$S_{p max} = m$

Effiency: 各段处于工作时间的时空区/流水线中各段总的时空区

$\eta = (n \times m \times \Delta t_0)/(m+n-1)\Delta t_0 \times m = \frac{n}{(m+n-1)}$

冒险 { 结构 数据 控制 }

IF ID EX MEM WB

Target = PC + offset

```
int fact(int n)  -> x5
{ if(n<1) return f
  else return n*fact(n-1)
}
addi x2, x2, -8
sw   x1, x2, 4   save
sw   x5, x2, 0
slti x6, x5, 1   test
beq  x6, x0, L1
addi x17, x0, 1
addi x2, x2, 8
jalr x0, x1, 0
L1: addi x5, x5, -1
jal  x1, -36     recursive
lw   x1, x2, 0
lw   x1, x2, 4
addi x2, x2, 8
mul  x17, x5, x17
jalr x0, x1, 0
```

```
leaf(int g,h,i,j) in x10-x13
{                    in x7
  int f
  f=(g+h)-(i+j)
  return f
}
addi x2, x2, -4    save
sw   x7, x2, 0
add  x5, x10, x11  procedure
add  x6, x12, x13
sub  x7, x5, x6
add  x17, x7, x0
lw   x7, x2, 0     Restore
addi x2, x2, 4
jalr x0, x1, 0     Return
```

数据冒险条件：

1° EX/MEM.RegRd = ID/EX.Reg.Rs1
EX/MEM.RegRd = ID/EX.Reg.Rs2
MEM/WB.Reg.Rd = ID/EX.Reg.Rs1
MEM/WB.Reg.Rd = ID/EX.Reg.Rs2

2° EX/MEM.RegWrite
MEM/WB.RegWrite

3° EX/MEM.RegRd ≠ 0
MEM/WB.RegRd ≠ 0

⇒ forward A/B { 00 ID/EX; 10 EX/MEM; 01 MEM/WB }

Double：优先 EX/MEM

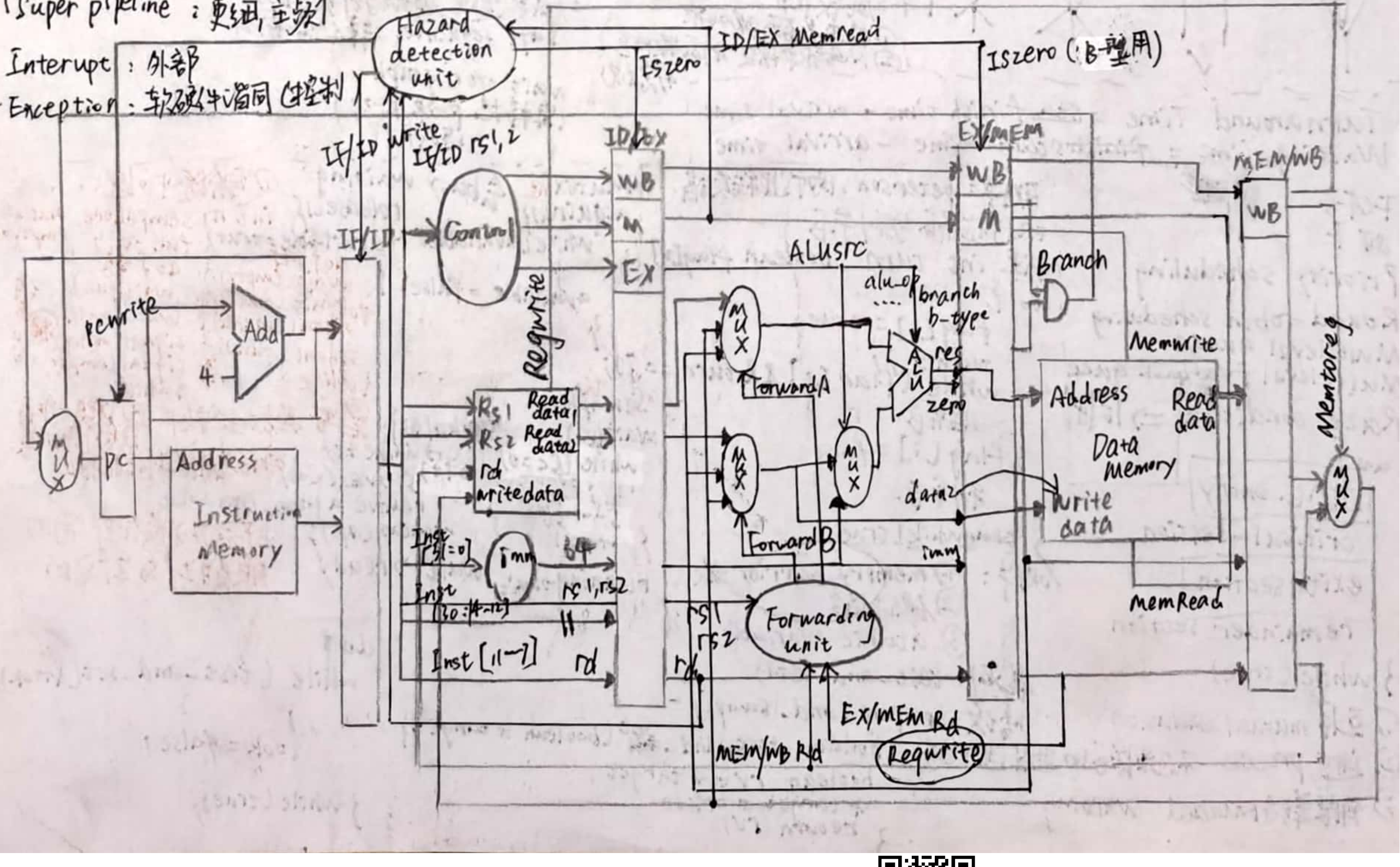
Load-use：在ID阶段检查，在MEM才输出 (ID/EX.MemRead && (ID/EX.RegRd = IF/ID.Reg.Rs))

Stall：强制 ID/EX.Reg = 0
锁死 pc，IF/ID Reg 不变

lw rd, offset(rs1)　x[rd] = sext(M(x[rs1]) + sext(off))
sw rs2, offset(rs1)　M[x[rs1] + sext(off)] = x[rs2]

{ nonlinear pipeling
Multiple issue { 静态 VLIW：静态 complier 调度；动态 Superscalar：硬件 cpu 调度（没那么理想）}
Super pipeline：更细，主频↑ }

Interrupt：外部
Exception：软硬件错误（控制）



syscall 改变mode (kernel entry 保存保护数据) system call table

用user的API接口 系统调用 index是syscall number 是用第一个寄存器传的, &eax ao。也将syscall返回值传回来 (ptr_regs[0])

进程管理 / 内存管理 / 文件存储管理 / I/O管理 / 保护

syscall: 进程控制 / 文件管理 / 设备管理 / 信息维护 / 通信 / 保护

传参方式: stack / reg / reg传指针

.c → .i → .S → .o —ld→ a.out ELF

ELF header
Program header table
.init
.text
.rodata
.data
.bss
.symtab
.debug
.line
.strtab
Section header table

只读代码段 / 读写数据段 / 不加载到内存的符号表

High
.text: code
.rodata: 初始化只读数据, const
.data: 初始化数据, 静态变量, 全局变量
.bss: 未初始化数据

High
内核内存
stack
共享库内存映射区
heap
读写段
只读代码段

argc, argv / 局部变量 / malloc / data上有空 分配空间

· 静态连接
  · 所有库被封装
  · 不需要动态loader
  · _start 直接执行
· 动态
  · 重用libraries
  · 共享库
  · 需要loader

IPC: shared memory · buffer
message passing
  · 直接, 间接
  · 同步, 异步
  · 自动, 显式
RPC: 远程, 分布式
socket: IP+端口(号)
pipe: fd[0]读 fd[1]写
  · 单/双
  · 父子关系
普通, 单向, fork.
命名, 双向, 无父子关系
背后是shared memory

Reg > Cache > memory > disk

PCB (进程控制块)
· state
· pc
· reg
· 调度信息
· 内存管理信息
· 记账信息
· I/O状态信息
· pid
调度保存pcB
linux (task_struct)
·指针 (指向其它pCB)

Process
· code/text
· data section (全局变量)
· pc
· stack (局部变量, ra, 函数参数, 临时变量)
· reg
· heap

high: stack / heap / data / text

new —admitted→ ready —scheduler dispatch→ running —exit→ terminated
running —interrupt→ ready
running —I/O or event wait→ waiting —I/O or event completion→ ready

Ready Queue: 马上运行
Waiting Queue: 等待I/O (有很多)

死锁 必要条件:
① mutual exclusion
② hold and wait
③ no preemption
④ circular wait

R₁, P₂ request, P₃ assignment, P₁, R₂, P₄

thread
有自己的[ID, pc, reg, stack 和存储区域 (DLL)] — context
share: code section, data section, heap, open files and signals

user — kernel

I/O bound: 大量短 cpu
cpu bound: 少量长 cpu

3) 死锁检测
① 检查是否出现死锁
② 从死锁中恢复
— 4) 忽视

处理方法:
1) 死锁预防, 确保四个条件之一不成立
· 互斥 → 共享 ×
· 持有并等待: 申请资源, 不占有其它资源
· 无抢占 → 抢占
· 循环等待 → 资源申请有序
2) 死锁避免: 需要额外的信息, 确保进程是安全状态
safe sequence, 银行家算法

wait-for graph
(直接把 p→R, R→p 连成 p→p)

Turnaround Time = finish time - arrival time
Waiting Time = start time - arrival time

FCFS
SJF
Priority scheduling
Round-robin scheduling
Multilevel queue
Multi-level feedback queue
Race condition ⇒ 同步

```
do {
  进入区 entry
  critical-section
  exit section
  remainder section
} while (true)
```

① 互斥 mutual exclusion
② 进步, process 不在剩余区的进程进入临界区
③ 有限等待 bounded waiting

软件: peterson, 两个进程交错执行临界区与剩余区
共享 int turn, boolean flag[2]

```
do {
  flag[i] = true;
  turn = j;
  while (flag[j] && turn == j);
    临界区
  flag[i] = false;
    剩余区
} while (true)
```

硬件: ① memory barrier 较
② 硬件指令
③ atomic variable

检测: test_and_set()
修改: compare_and_swap()

```
boolean test_and_set(boolean *target){
  boolean rv = *target;
  *target = true;
  return rv;
}
```

Mutex 短 会busy waiting

```
acquire(){
  while(available)
  ;
  available = false;
}
release(){
  available = true;
}
```

Semaphore 长

```
wait(S){
  S->value--;
  if (S->value<0){
    add this to S->list
    block();
  }
}
signal(S){
  S->value++;
  if (S->value <= 0){
    remove a process from S->list
    wakeup(p)
  }
}
```

running→wait   wait→ready

① 有界缓冲问题
int n; sempahone mutex=1, empty=n, full=0;

```
do {   // 生产者
  wait(empty)
  wait(mutex)   // 加
  signal(mutex)
  signal(full)
} while
```

```
do {   // 消费者
  wait(full);
  wait(mutex)
  signal(mutex)
  signal(empty)
} while
```

生产者为消费者生产满的
消费者为生产者生产空的

② 读者-作者
③ 哲学家就餐
需要其它加锁技术
将此等待限制在临界区

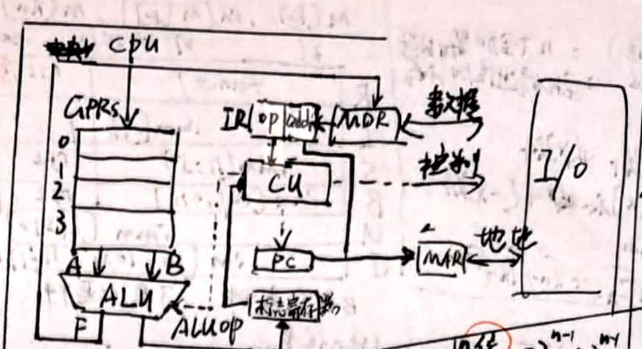
```
do {
  while (test_and_set(&lock))
  ;
  lock = false;
} while (true);
```

扫描全能王 创建

1. Computer System & Von-Neumann Model

- Applications
- Complier
- OS
- Instruction Set Architecture (ISA)
- Computer Organization (微体系结构)
- Register Transfer Level (RTL)
- Digital Logic Circuit (DLC)
- Device Technology

Soft ware ↓ / Hard ware



Radix

B O D H (0X)

整基取余，上低下高

乘基取整，上高下低

补码 $\sim X = -X-1$

$[X]_{补} = M + X_T \pmod M$

$[-128, 127]$ 7

$[-32768, 32767]$ 15

$[-65536, 65535]$ 16

$[-2147483648, \dots]$ 31

n 位 $-2^{n-1} \sim 2^{n-1}-1$

$0 \sim 2^n - 1$

int x = -1

unsigned u = 2^{31}

printf("x=%u=%d\n", x, x)

$x = 2^{32}-1 = -1$

printf("u=%u=%d\n", u, u)

$x = 2^{31} = -2^{31}$

$-2^{31} < 2^{31}-1$

C90: 无 ×

C99: 有 ✓

- 汇编器（重复）
- 解释器 (单个不多)
- 编译 (整个不多)

浮点数 IEEE 754

规格化 (左、右规)

1 8 23 — s e f

1 11 52

对阶，

不满足加法结合律的原因

Exp = Bias + 阶

	32				64			
	符号	阶码	尾数	值				
正零	0	0	0	0	0	0	0	0
负零	1	0	0	0	1	0	0	0
+∞	0	255	0	∞	0	2047	0	∞
-∞	1				1			
NaN	0/1	255	≠0	NaN				
规范非零	0	0<e<255	f	$2^{e-127}(1.f)$	0	0<e<2047		2^{e-1023}
负	1	0<e<255		(-1)				
非规范	0	0	f≠0	$2^{-126}(0.f)$				
负	1				1			

assembler
interpreter
compiler

汇编：m → 机器程序

编 → 机指令

高 → asm/机器码

i == (int)(float) i ×

f == (float)(int) f ×

d+f-d == f ×

BCD码，余3 (8421)

ASCII 0 → 30H A → 41H, a → 61H

b → Byte → word → dword

1KB = 2^{10}B 1MB, 1GB, 1TB

little endian big endian 和顶用 reg 顺

字长：cpu内部用于整数运算 运算器/寄存器位数

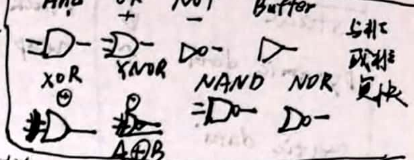
denormalized.

对折偶：以奇两位5-6X

$Y_2 = (X+\bar{X}) Y_2$

或 5

And OR NOT Buffer



$X+Y=Y+X$, $X \cdot Y = Y \cdot X$ $(X+Y)+Z = X+(Y+Z)$, $(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$

$X \cdot (Y+Z) = X \cdot Y + X \cdot Z$, $X + Y \cdot Z = (X+Y) \cdot (X+Z)$

$X + X \cdot Y = X$, $X \cdot (X+Y) = X$ 吸收定律 $X \cdot Y + X \cdot \bar{Y} = X$, $(X+Y)(X+\bar{Y}) = X$

$X \cdot Y + \bar{X} \cdot Z + Y \cdot Z = X \cdot Y + \bar{X} \cdot Z$, $(X+Y)(\bar{X}+Z) \cdot (Y+Z) = (X+Y) \cdot (\bar{X}+Z)$

$\overline{X_1 \cdots X_n} = \bar{X_1} + \bar{X_2} + \dots + \bar{X_n}$

$\overline{X_1 + X_2 + \dots + X_n} = \bar{X_1} \cdot \bar{X_2} \cdots \bar{X_n}$

对偶：· ↔ +，1/0 取反，变量不变

$F(x,y,z) = x\bar{y}\bar{z} + \bar{x}yz$

$G = \bar{F} = \overline{x\bar{y}\bar{z}} \cdot \overline{\bar{x}yz} = (\bar{x}+y+z)(x+\bar{y}+\bar{z})$

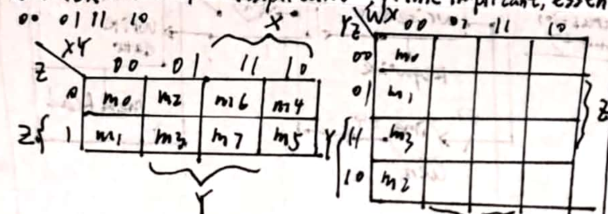
SOP (析取) POS (合取)

仅出现一次：最大项 最小项

全值=1 / 任取1 所有min项和=1 / 所有max项任取1

Σ ∏ (非是0)

K-Map · implicant prime implicant, essential prime implicant



module <模块名> (<端口列表>);

参数定义 (可选)

数据类型

持续赋值

always/initial 过程块

函数

延时说明

endmodule

缺省是 wire，module 定义 input 是 wire，默认 output 是 wire

always 里一定写 reg always@(*)

运算符

!! 注意位宽变化

位运算 (双目) 归约 (单目)

=== : 不定态

!== : 高阻态

{a, {2{c,d}}}

默认 32 位

2'b10

高位在左 超位会丢失

非阻塞 时序电路

不可重复驱动，回路不可

begin-end

阻塞：组合，=

非：时序，<=

允许在一个 always 里对 reg 赋值，但不可在多个 always 里对 reg 赋值

! ~

* / %

+ -

<< >>

< <= > >=

== != === !==

& ~&

^ ^~ 或 ~^

| ~|

&&

||

?:

归约 位

Combinational

仅依赖 input，无回路，不存在一个节点两输入，非法值，High impedance

非 > 与，与非 > 异或，同或 > 或，或非

Fan in / Fan out

gate delay

2-level ~ multi-level · Encoder · decoder

$2^n \to n$ $n \to 2^n$

multiplexer：Mux，$\lceil \log_2 k \rceil$ 控制

从 n 个选一个

demultiplexer：DeMux

把一个分配给 n 个中的一个

· Race Hazard

half adder

$A \oplus B = F$

Cout = A · B

full adder

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

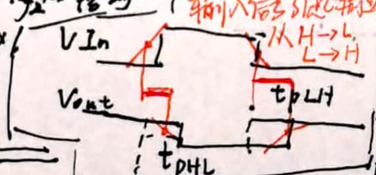
$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$

T_{pd}：传输延迟

T_{cd}：最小延迟

输入信号引起输出信号

从 H → L, L → H



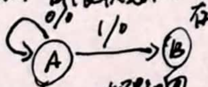
① t_{setup}：clk 来到 D 稳定

② t_{hold}：clk 后到 D 不变

③ clk-to-Q：clk 之后到 Q 改变

传输延迟

FSM 有限状态机 回路 存储



mealy：都影响

moore：无 input

D flip flop 时钟 主从结构 边沿触发

level-triggered：电平 ⇒ 锁存器 latch

edge-triggered：边沿 ⇒ flip-flop

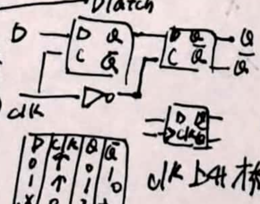
synchronous：同步 有一个时钟，只在时钟发生

Asynchronous：异步 无目标中心时钟

latch 双稳态

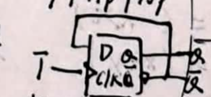
flip-flop

D latch



clk 上升沿

T flip flop



$Q_{n+1} = T \oplus Q_n$

T=1 翻转

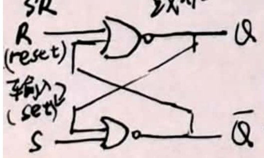
T=0 不变

相当于把 J 和 K 连到一起

Bistable circuit 双稳态电路

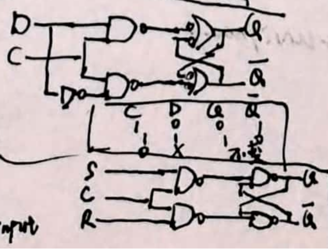
· Latches (level-trigger) 触发延迟

SR 或非



S	R	Q	Q̄
0	0		不变
0	1	0	1
1	0	1	0
1	1		禁止

D latch



SR clocked

有一个 control input

$SR: Q^* = S + \bar{R}Q$

$JK: Q^* = J\bar{Q} + \bar{K}Q$

$T: Q^* = T\bar{Q} + \bar{T}Q$

$D: Q^* = D$

时序电路

扫描全能王 创建

ALU.

CRA (行波进位) : n个全加器相连
carry lookahead : 全先行进位加法器
算术运算 logic

[x+y]补 = [x]补 + [y]补
[x-y]补 = [x]补 + [-y]补

OF: $C_{n-1} \oplus C_n$
$= X_{n-1}Y_{n-1}\bar{F}_{n-1} + \bar{X}_{n-1}\bar{Y}_{n-1}F_{n-1}$

add, addi

lui, x1, 42 ; 42放在x1高20位, 低12位0.

sll, slt, sltu, srl, sra

ld x1, 80(x2) regs[x1] ← mem[80+regs[x2]]
sd x2, 400(x3) mem[400+regs[x3]] ← regs[x2]

jal x1, offset regs[x1] ← pc+4 ; pc ← pc+(offset<<1)
jalr, x1, x2, offset regs[x1] ← pc+4, pc ← regs[x2]+offset
bne, blt, bge, bltu, bgeu 偏移量×2

stack ↓
dynamic data ↑
static data
pc.

stack
heap 堆

IF → ID → EX → MEM → WB, 调pc.
取指 译码 执行 访存 写回寄存器

datapath, control

m[k], m[m[k]], m(Rn), m[Rm+Rn], m[Rm+X], m(Rn+Rm+@X)

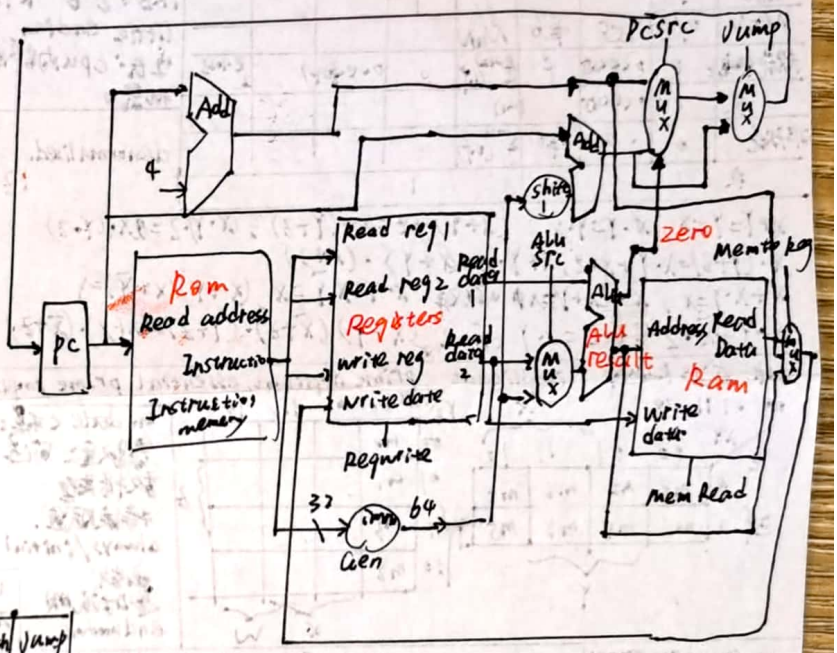
	31	27	25	24	20	19	15	14	12	11	7	6	0
R	funct7		rs2		reg	rs1		funct3		rd		opcode	
I	imm[11:0]									rd			
S	imm[11:5]		rs2							imm[4:0]			
B	imm[12|10:5]		rs2							imm[4:1|11]			
U	imm[31:12]									rd			
J	imm[20|10:1|11|19:12]									rd			

B是S变种 J是U变种

R: 寄存器操作数
I: 短立即数/Load
S: 存数
B: 条件跳转
U: 长立即数
J: 无条件跳转

目的

R-type	funct7	funct3
add	0	000
sub	1	000
and	0	111
or	0	110



R.	opcode	ALUSrc	MemtoReg	reg write	mem read	mem write	Branch	Jump
ld (I)	0000011	0	01	1	1	0	0	0
sd (S)	0100011	1	X	0	0	1	0	0
beq (B)	1100011	1	X	0	0	0	1	0
jal (J)	1101111	X	10	1	0	0	0	1

pc, Rom, Ram
registers 是在 write register writedata 是在 时序

mux, ALU, Imm Add 组合

(x<0) == (-x>0)
~x+~y < -(x+y)
unsigned 0
size of 是 unsigned
-x = ~x+1
unsigned int → float 会出错
-(unsigned)(-x-y) = unsigned x + unsigned y.

1ns = 1000MHz
1us = 1MHz

$t_{clk} > t_{ffpd} + t_{comb} + t_{setup}$
$t_{hold} < t_{ffpd} + t_{comb}$

Thold
t_{pd}
t_{setup}

扫描全能王 创建