一、实验目标

（详细论述所设计作品的功能）

1. 深入了解 CPU 的原理。
2. 画出实现 54 条指令的 CPU 的通路图。
3. 学习使用 Verilog HDL 语言设计实现 54 条指令的 CPU。

二、总体设计

1. 作品功能设计及原理说明（作品总体设计说明）
   实现 54 条 MIPS 指令的单周期 CPU。
   指令集如下：

指令	指令说明	指令格式	OP 31-26	FUNCT 5-0	指令码 16 进制
addi	加立即数	addi rt, rs, immediate	001000		20000000
addiu	加立即数（无符号）	addiu rd, rs, immediate	001001		24000000
andi	立即数与	andi rt, rs, immediate	001100		30000000
ori	或立即数	ori rt, rs, immediate	001101		34000000
sltiu	小于立即数置 1（无符号）	sltiu rt, rs, immediate	001011		2C000000
lui	立即数加载高位	lui rt, immediate	001111		3C000000
xori	异或（立即数）	xori rt, rs, immediate	001110		38000000
slti	小于置 1（立即数）	slti rt, rs, immediate	001010		28000000
addu	加（无符号）	addu rd, rs, rt	000000	100001	00000021
and	与	and rd, rs, rt	000000	100100	00000024
beq	相等时分支	beq rs, rt, offset	000100		10000000
bne	不等时分支	bne rs, rt, offset	000101		14000000
j	跳转	j target	000010		08000000
jal	跳转并链接	jal target	000011		0C000000
jr	跳转至寄存器所指地址	jr rs	000000	001000	00000009
lw	取字	lw rt, offset(base)	100011		8C000000
xor	异或	xor rd, rs, rt	000000	100110	00000026
nor	或非	nor rd, rs, rt	000000	100111	00000027
or	或	or rd, rs, rt	000000	100101	00000025

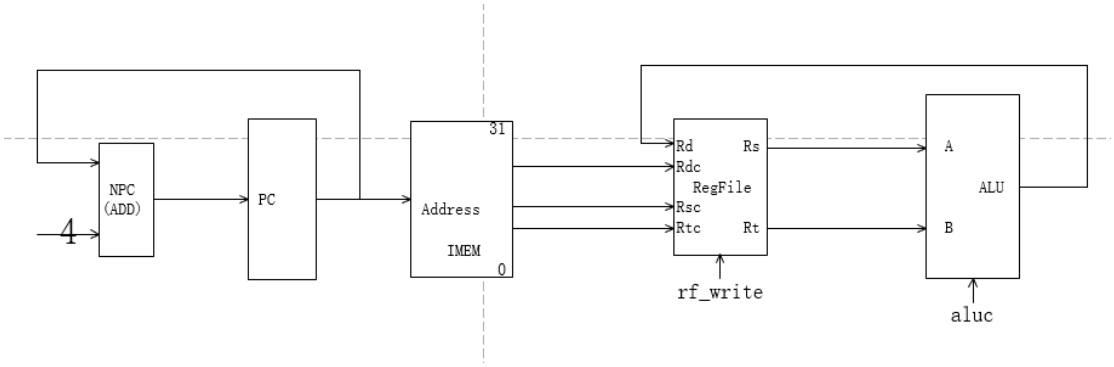
sll	逻辑左移	sll rd, rt, sa	000000	000000	00000000
sllv	逻辑左移（位数可变）	sllv rd, rt, rs	000000	000100	00000004
sltu	小于置 1（无符号）	sltu rd, rs, rt	000000	101011	0000002B
sra	算数右移	sra rd, rt, sa	000000	000011	00000003
srl	逻辑右移	srl rd, rt, sa	000000	000010	00000002
subu	减（无符号）	sub rd, rs, rt	000000	100010	00000022
sw	存字	sw rt, offset(base)	101011		AC000000
add	加	add rd, rs, rt	000000	100000	00000020
sub	减	sub rd, rs, rt	000000	100010	00000022
slt	小于置 1	slt rd, rs, rt	000000	101010	0000002A
srlv	逻辑右移（位数可变）	srlv rd, rt, rs	000000	000110	00000006
srav	算数右移（位数可变）	srav rd, rt, rs	000000	000111	00000007
clz	前导零计数	clz rd, rs	011100	100000	70000020
divu	除（无符号）	divu rs, rt	000000	011011	0000001B
eret	异常返回	eret	010000	011000	42000018

jalr	跳转至寄存器所指地址，返回地址保存在	jalr rs	000000	001001	00000008
lb	取字节	lb rt, offset(base)	100000		80000000
lbu	取字节（无符号）	lbu rt, offset(base)	100100		90000000
lhu	取半字（无符号）	lhu rt, offset(base)	100101		94000000
sb	存字节	sb rt, offset(base)	101000		A0000000
sh	存半字	sh rt, offset(base)	101001		A4000000
lh	取半字	lh rt, offset(base)	100001		84000000
mfc0	读 CP0 寄存器	mfc0 rt, rd	010000	000000	40000000
mfhi	读 Hi 寄存器	mfhi rd	000000	010000	00000010
mflo	读 Lo 寄存器	mflo rd	000000	010010	00000012
mtc0	写 CP0 寄存器	mtc0 rt, rd	010000	000000	40800000
mthi	写 Hi 寄存器	mthi rd	000000	010001	00000011
mtlo	写 Lo 寄存器	mtlo rd	000000	010011	00000013
mul	乘	mul rd, rs, rt	011100	000010	70000002
multu	乘（无符号）	multu rs, rt	000000	011001	00000019
syscall	系统调用	syscall	000000	001100	0000000C
teq	相等异常	teq rs, rt	000000	110100	00000034
bgez	大于等于 0 时分支	bgez rs, offset	000001		04010000
break	断点	break	000000	001101	0000000D
div	除	div rs, rt	000000	011010	0000001A

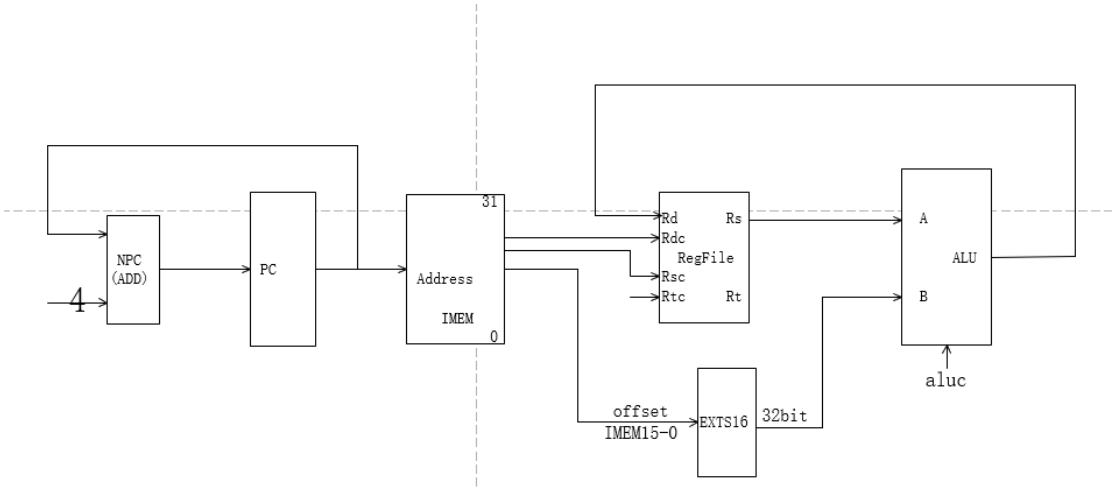
2. 硬件逻辑图

（除实验一外，均需给出硬件逻辑图，不得使用软件中自动生成的图，要求使用 visio 画，务必清晰明了）

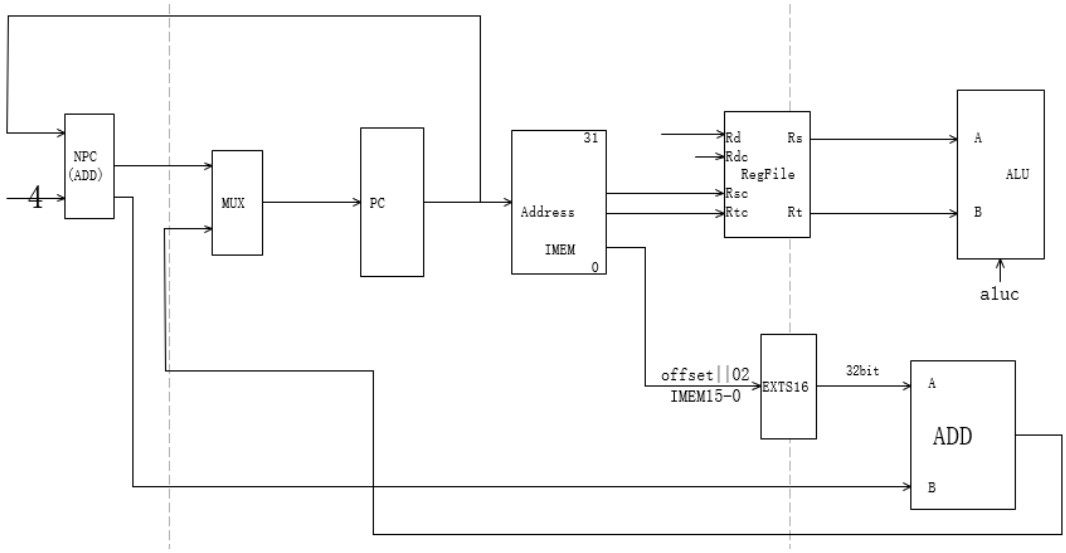
add/addu/and/nor/or/ sllv/slt/sltu/srav/srlv/sub/subu/xor



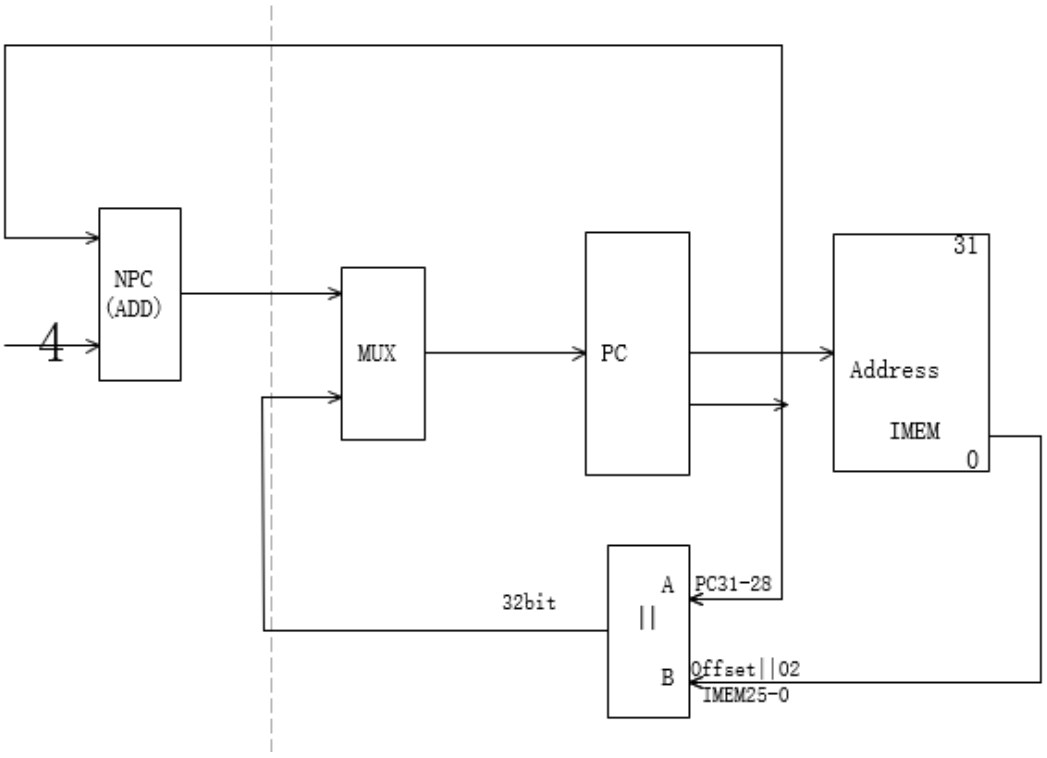
addi/addiu/andi/ori/ sll/slti/sltiu/sra/srl/xori



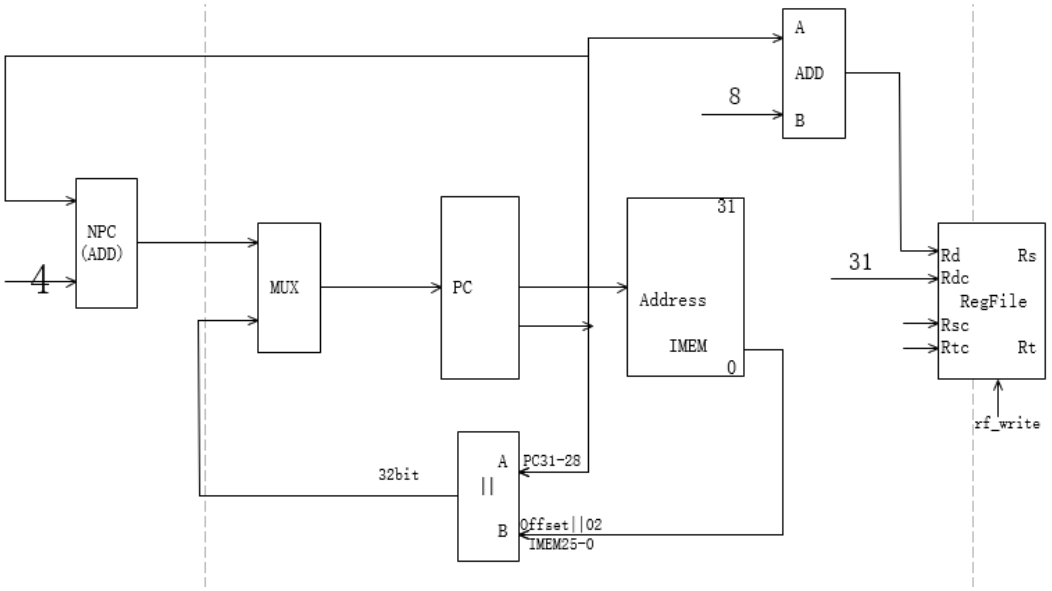
beq/bne



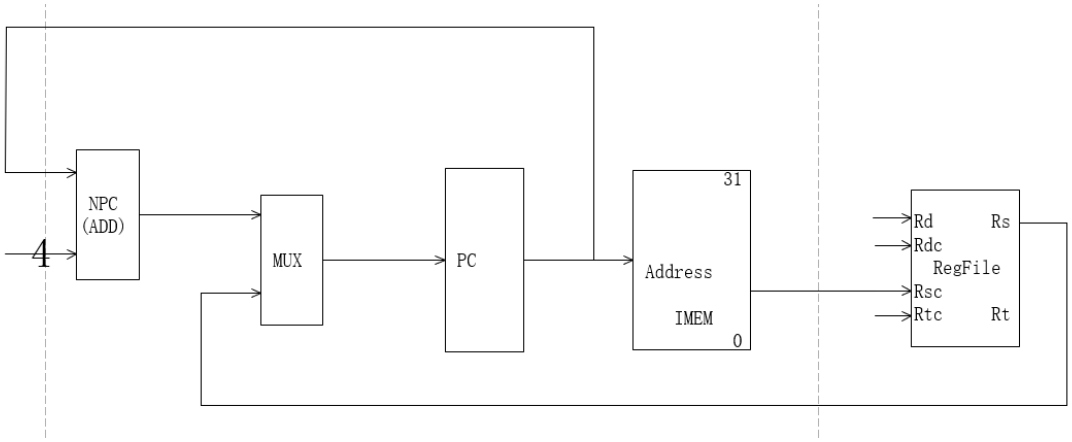
j


NPC
(ADD)
4
MUX
PC
31
Address
IMEM
0
A
||
B
PC31-28
32bit
Offset||02
IMEM25-0

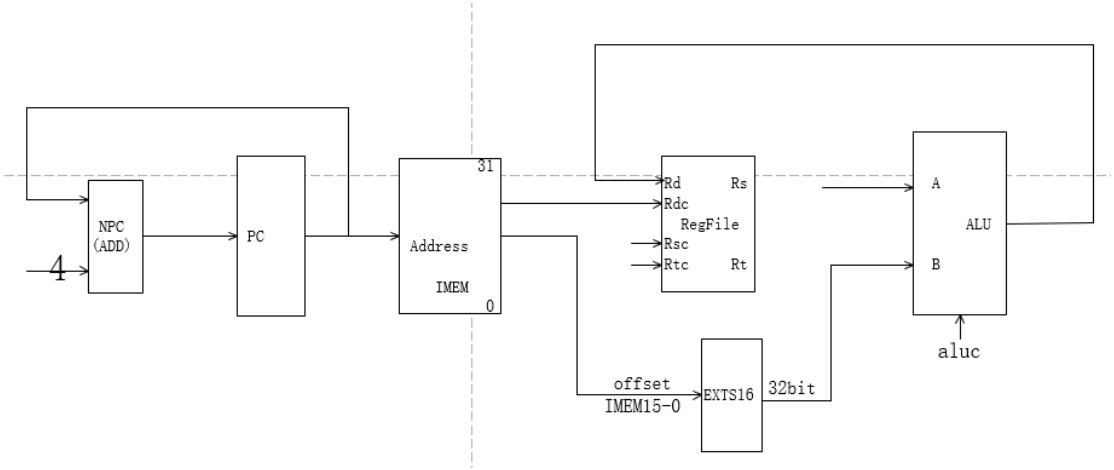
jal


A
ADD
8
B
NPC
(ADD)
4
MUX
PC
31
Address
IMEM
0
31
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
A
||
B
PC31-28
32bit
Offset||02
IMEM25-0

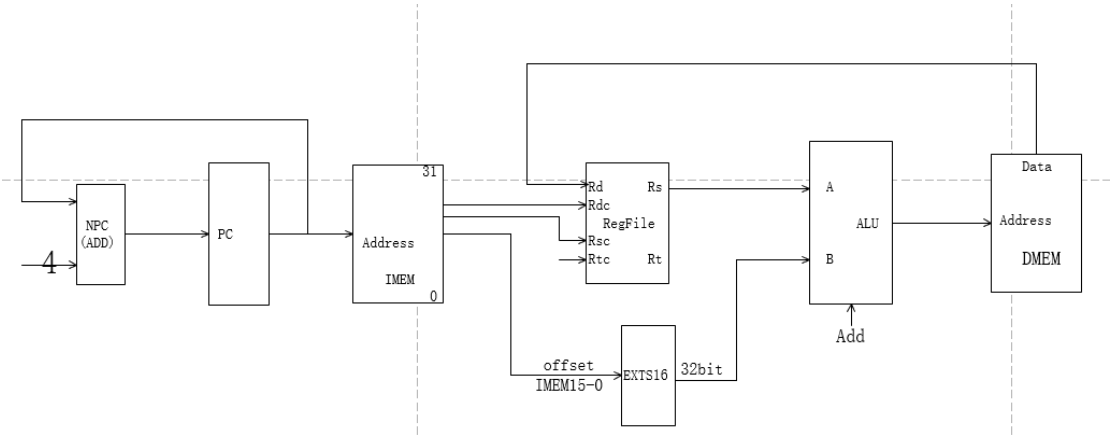
jr


NPC
(ADD)
4
MUX
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt

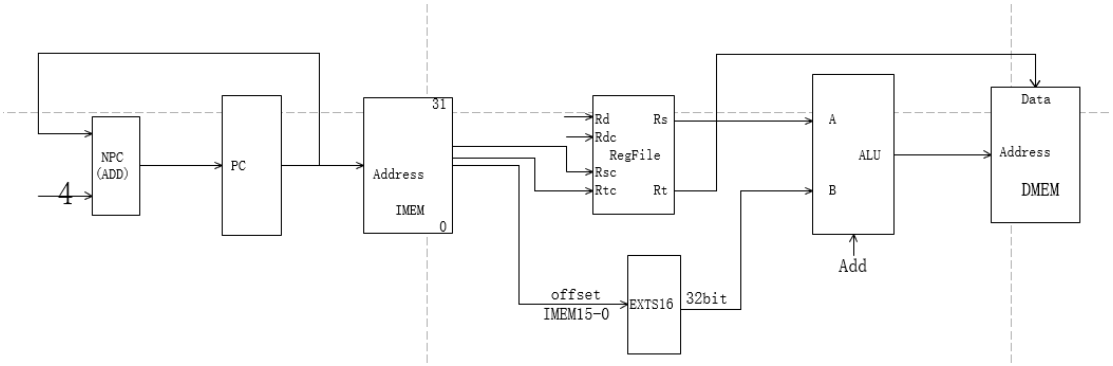
lui


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
A
ALU
B
aluc
offset
IMEM15-0
EXTS16
32bit

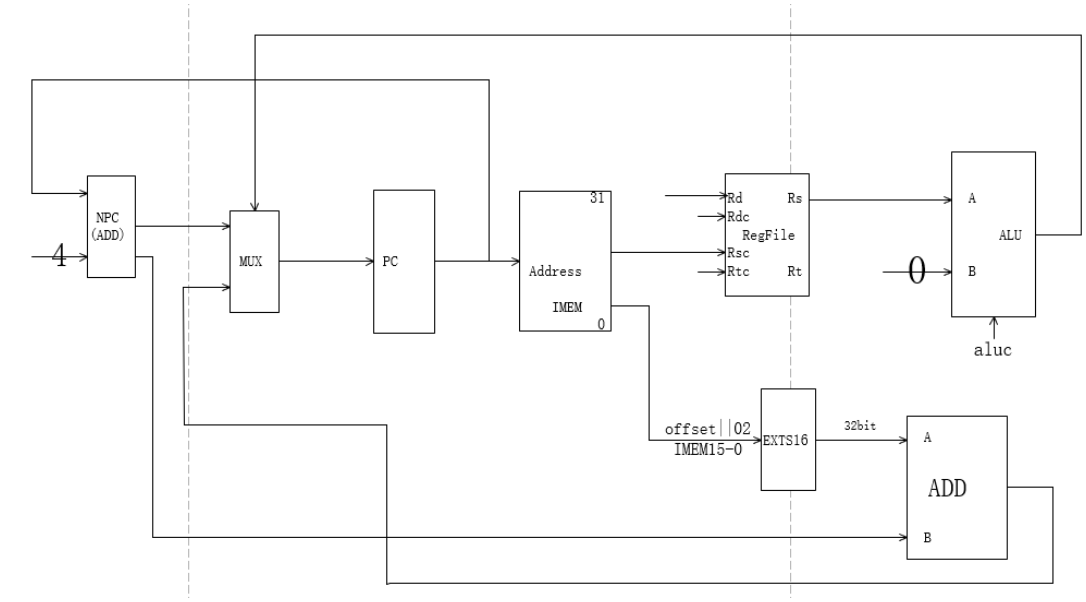
lw


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
A
ALU
B
Add
Data
Address
DMEM
offset
IMEM15-0
EXTS16
32bit

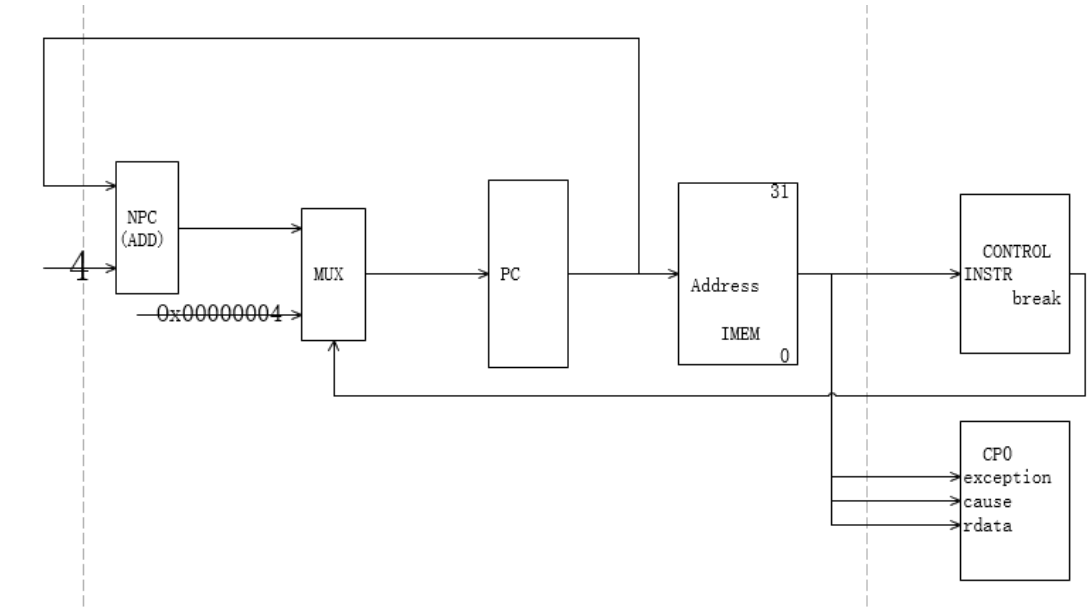
sw



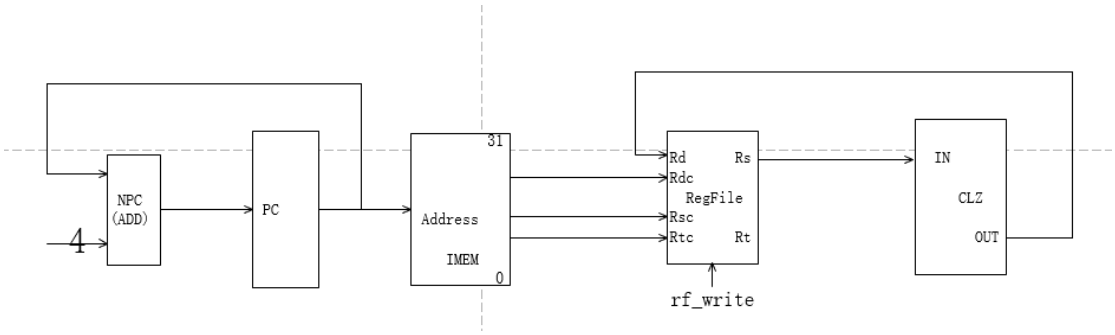
bgez



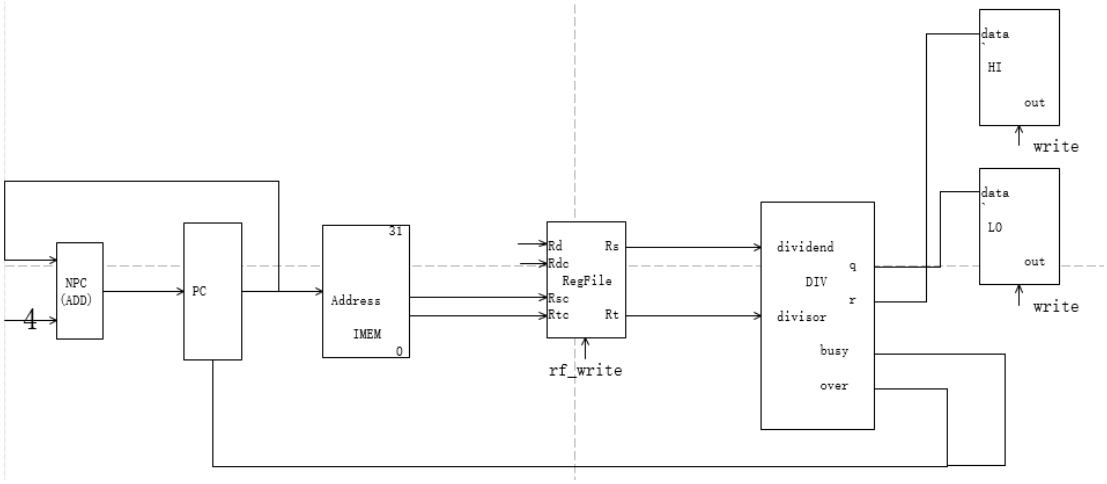
break/syscall



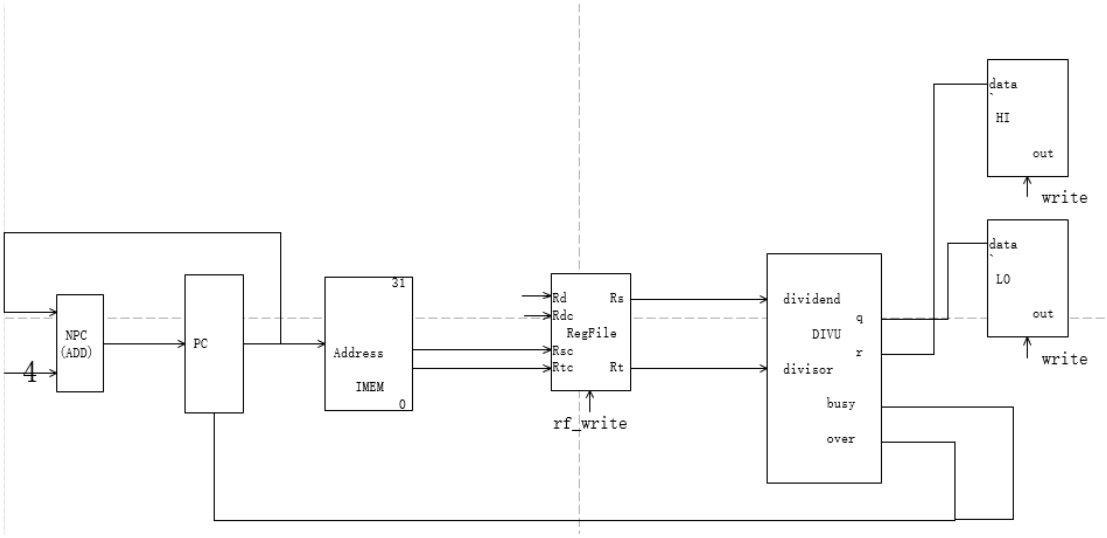
clz


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
IN
CLZ
OUT

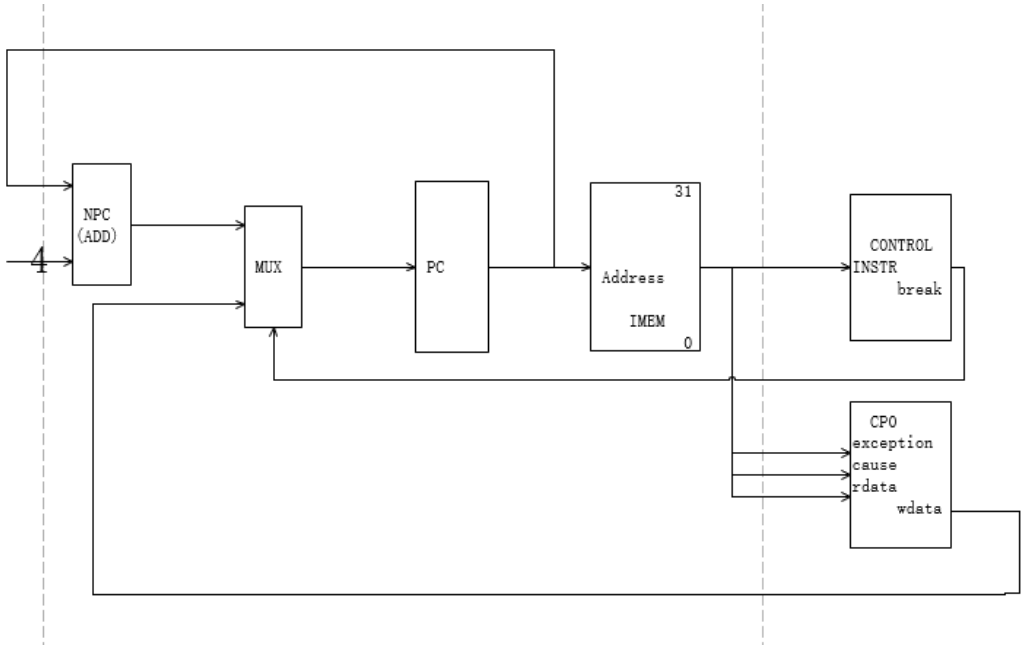
div


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
dividend
q
DIV
r
divisor
busy
over
data
HI
out
write
data
LO
out
write

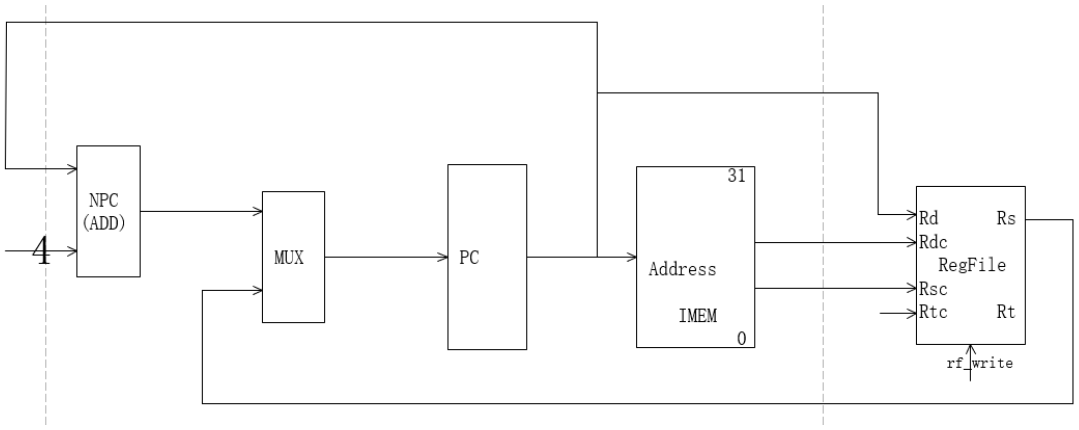
divu


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
dividend
q
DIVU
r
divisor
busy
over
data
HI
out
write
data
LO
out
write

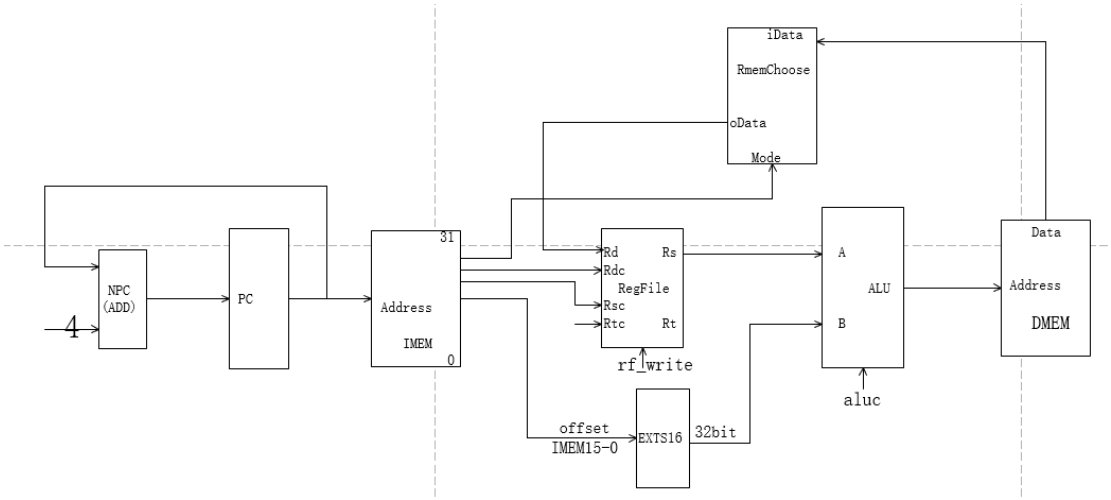
eret



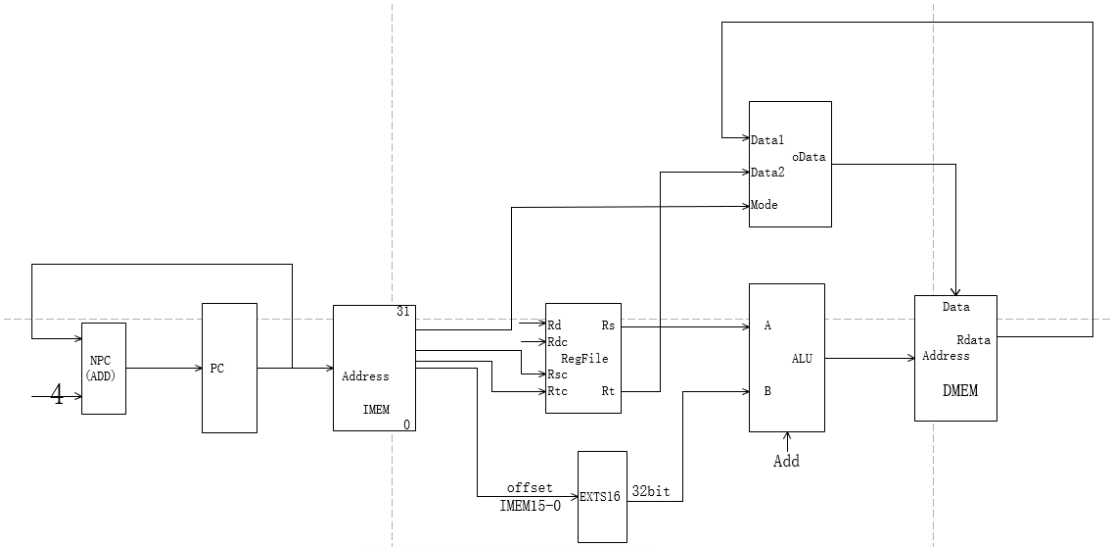
jalr



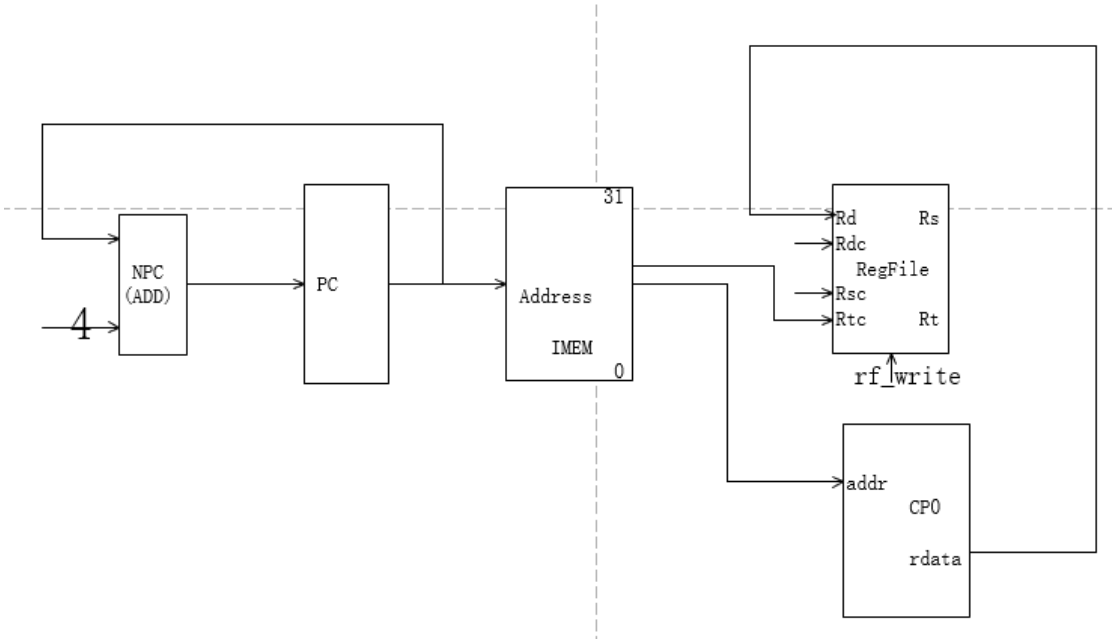
lb/lh/lbu/lhu


iData
RmemChoose
oData
Mode
NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_wrtie
A
ALU
B
aluc
Data
Address
DMEM
offset
IMEM15-0
EXTS16
32bit

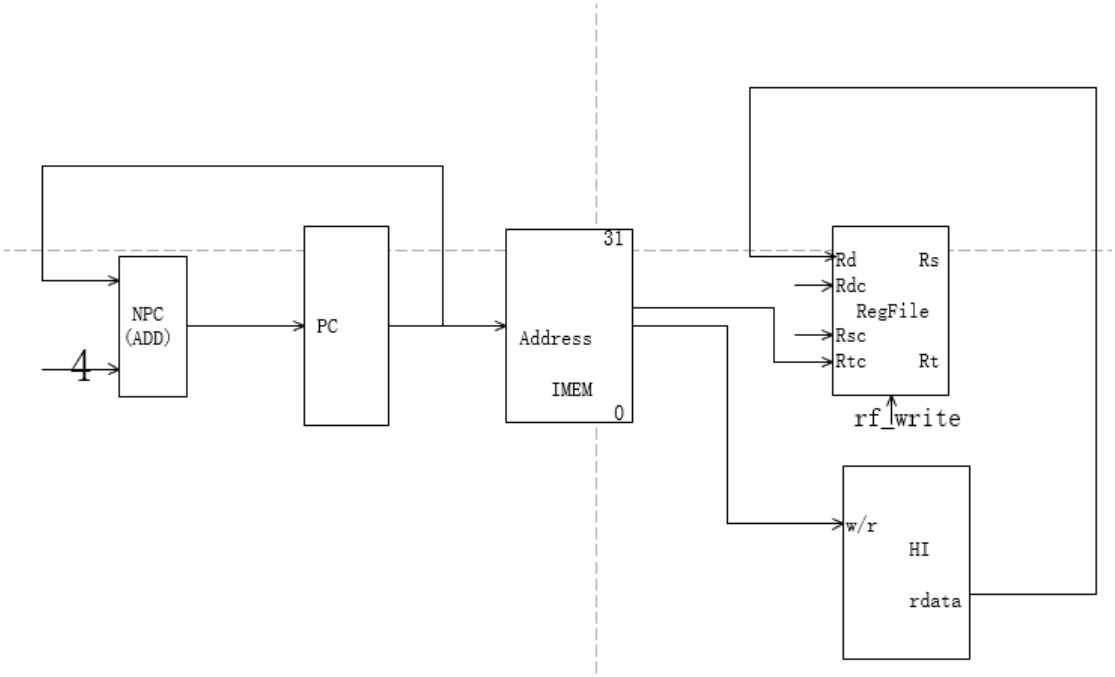
sb/sh


Data1
oData
Data2
Mode
NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
A
ALU
B
Add
Data
Rdata
Address
DMEM
offset
IMEM15-0
EXTS16
32bit

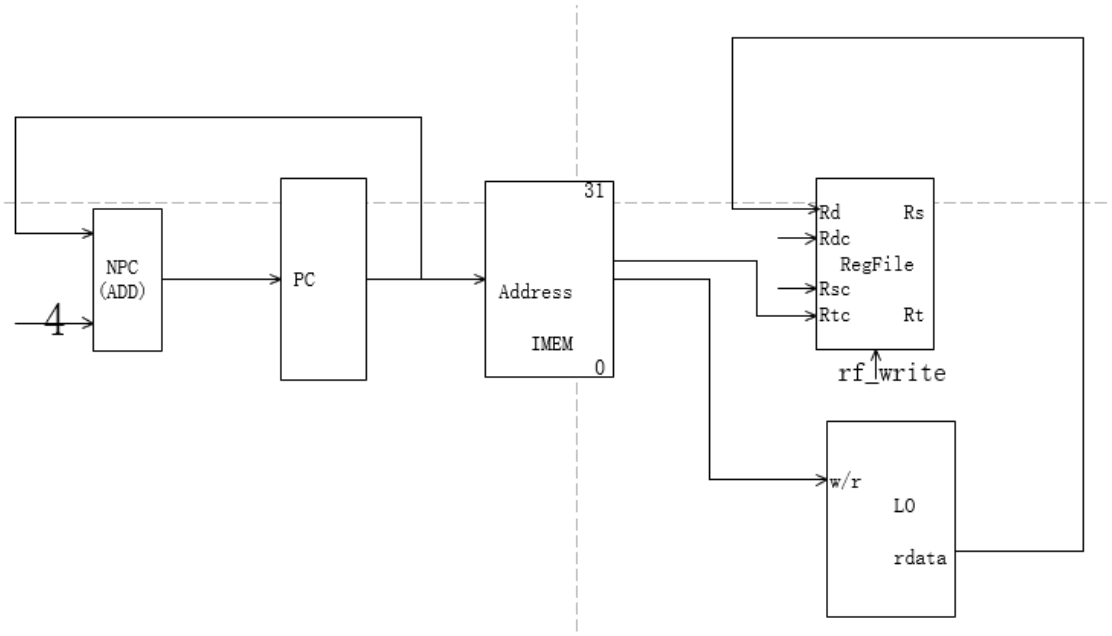
mfc0


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
addr
CP0
rdata

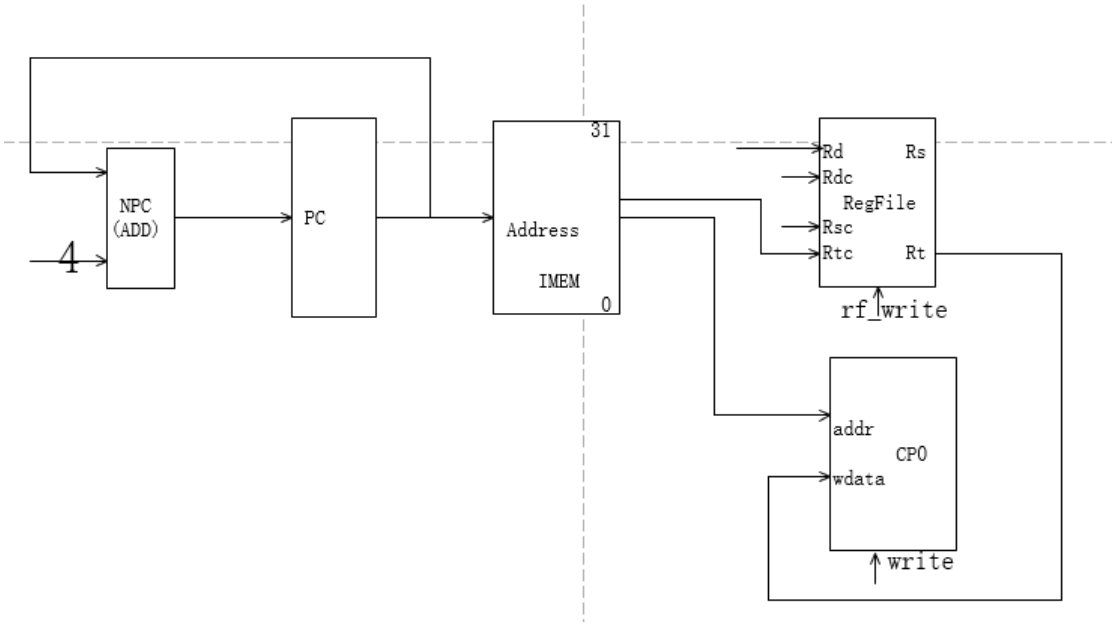
mfhi


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
w/r
HI
rdata

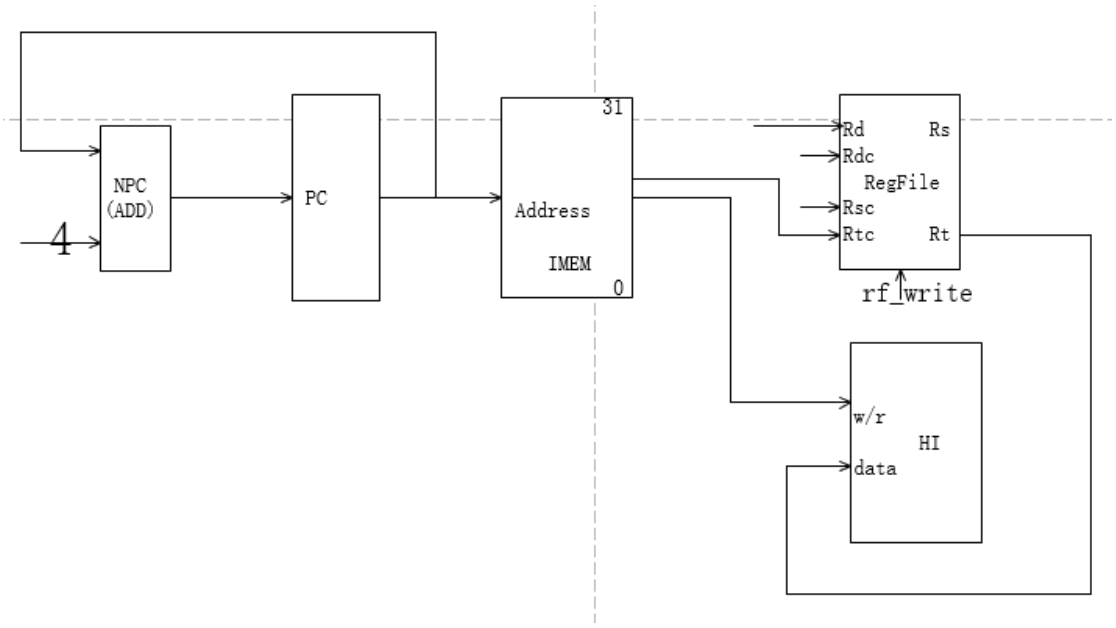
mflo


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
w/r
LO
rdata

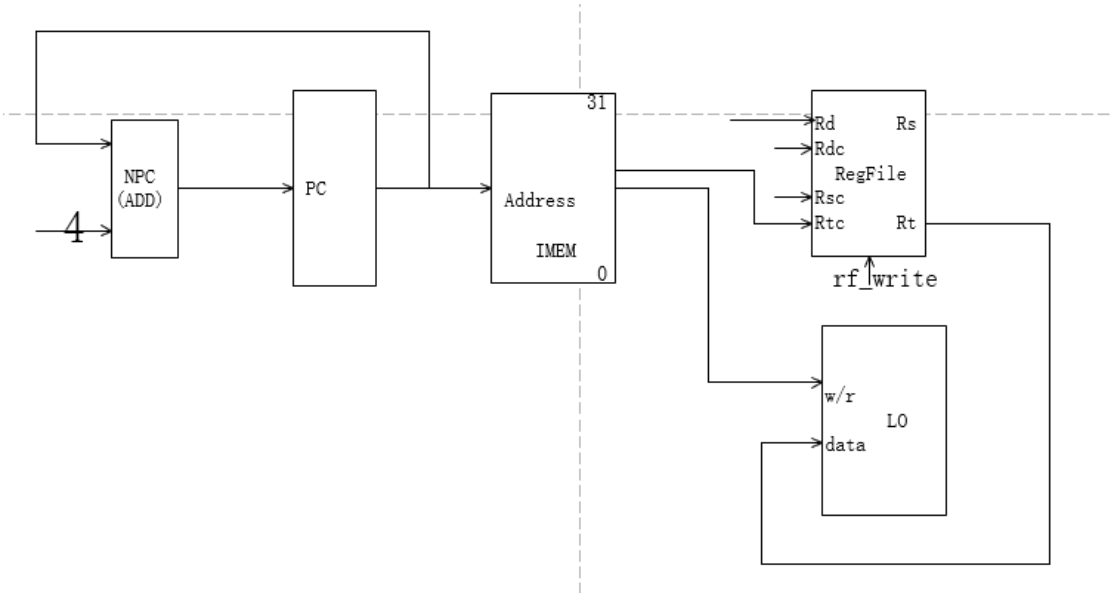
mtc0


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
addr
CP0
wdata
write

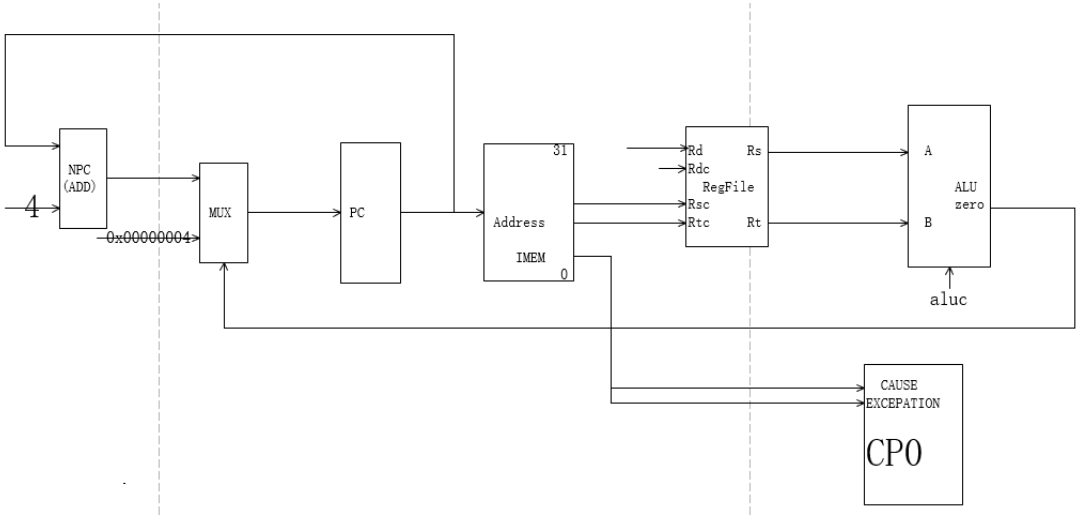
mthi


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
w/r
HI
data

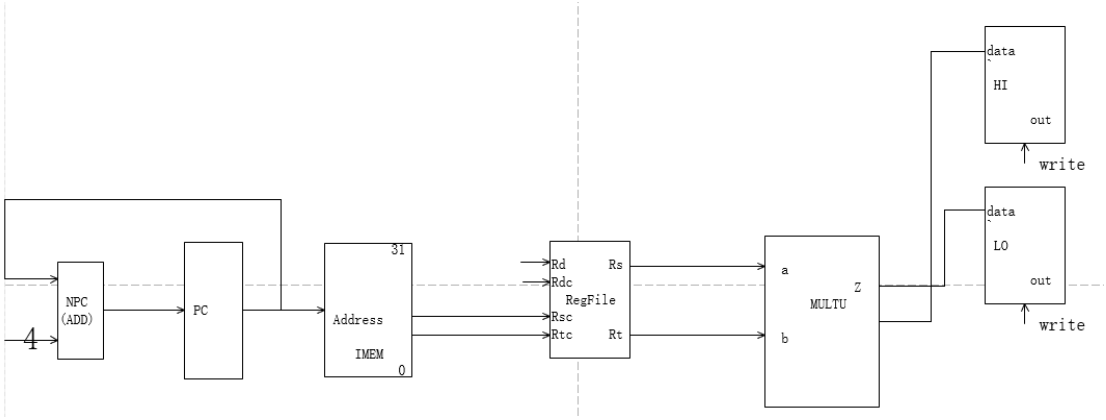
mtlo


NPC
(ADD)
4
PC
31
Address
IMEM
0
Rd
Rs
Rdc
RegFile
Rsc
Rtc
Rt
rf_write
w/r
LO
data

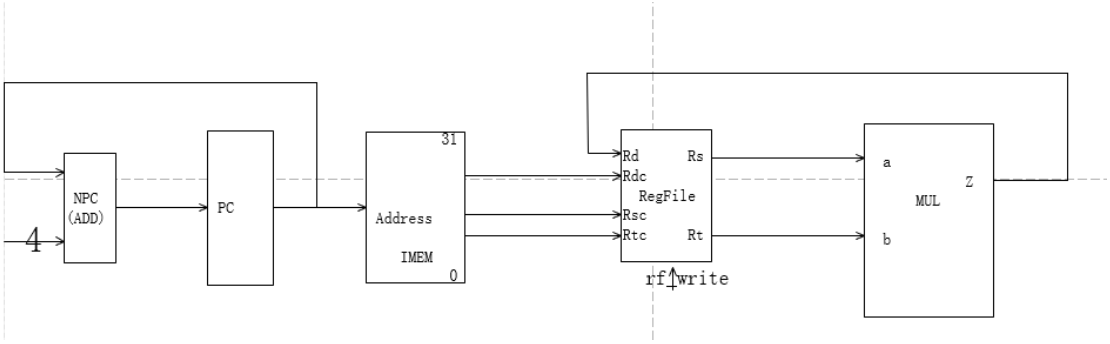
teq



multu



mul



信号：

add

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

addi

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	0	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

addiu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0000	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

and

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0100	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

addu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0000	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

andi

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0100	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

beq

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0011	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

bne

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0011	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

j/jal/jr

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

lui

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
100x	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	00				

lw

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	1	1	1	1	000
WMemMode	sign_lblh	RMemMode	rf_wa				
11	0	10	00				

nor

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0111	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

or

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0101	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

ori

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0101	1	0	0	0	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sll

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
111x	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sllv

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
111x	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

andi

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0100	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

slt

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1011	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

slti

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1011	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sltiu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1010	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sltu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1010	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sra

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1100	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

srav

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1100	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

srl

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1101	1	0	0	1	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

srlv

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
1101	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sub

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0011	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

subu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0001	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

sw

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	1	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
11	0	10	01				

xor

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0110	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

xori

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0110	1	0	0	0	1	0	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

bgez

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	01				

break

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	BREAK			
10	0	10	01	1			

clz

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	110
WMemMode	sign_lblh	RMemMode	rf_wa	clz			
10	0	10	01	1			

div

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	div			
10	0	10	01	1			

divu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	divu			
10	0	10	01	1			

eret

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	eret			
10	0	10	01	1			

jarl

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	011
WMemMode	sign_lblh	RMemMode	rf_wa	jarl			
10	0	10	01	1			

lb

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	1	1	1	1	000
WMemMode	sign_lblh	RMemMode	rf_wa				
10	1	00	00				

lbu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	1	1	1	1	000
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	00	00				

lh

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	1	1	1	1	000
WMemMode	sign_lblh	RMemMode	rf_wa				
10	1	01	00				

lhu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	1	1	1	1	000
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	01	00				

mfc0

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	010
WMemMode	sign_lblh	RMemMode	rf_wa				
10	0	10	10				

mfhi

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	101
WMemMode	sign_lblh	RMemMode	rf_wa	mfhi			
10	0	10	01	1			

mflo

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	100
WMemMode	sign_lblh	RMemMode	rf_wa	mflo			
10	0	10	01	1			

mtc0

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	mtc0			
10	0	10	01	1			

mthi

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	mthi			
10	0	10	01	1			

mtlo

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	mtlo			
10	0	10	01	1			

mul

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	111
WMemMode	sign_lblh	RMemMode	rf_wa	mul			
10	0	10	01	1			

multu

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	1	0	0	1	1	1	111
WMemMode	sign_lblh	RMemMode	rf_wa	mul			
10	0	10	01	1			

sb

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	1	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
00	0	10	01				

sh

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	1	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa				
01	0	10	01				

syscall

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	syscall			
10	0	10	01	1			

teq

aluc	rf_write	DM_W	DM_R	sign_extend	ALU_a	ALU_b	rf_wd
0010	0	0	0	1	1	1	001
WMemMode	sign_lblh	RMemMode	rf_wa	teq			
10	0	10	01	1			

三、主要模块设计

（该部分要求分点描述，对总体设计原理图中的每个功能模块进行描述说明）

ALU:

ALU 是负责运算的电路。ALU 必须实现以下几个运算：加（ADD）、减（SUB）、与（AND）、或（OR）、异或（XOR）、置高位立即数（LUI）、逻辑左移与算数左移（SLL）、逻辑右移（SRL）以及算数右移（SRA）、SLT、SLTU 等操作。输出 32 位计算结果、carry(借位进位标志 位)、zero(零标志位)、negative(负数标志位)和 overflow(溢出标志位)。

CONTROL:

解析 IMEM 中传来的指令，发出控制信号，控制与协调 CPU 各个部件

Regfiles:

用于在内存与 CPU 运算部件之间暂存数据

CPU54:
CPU 的主体，协调控制器，寄存器堆，ALU 等各个内部部件

IMEM:
指令存储器，用于存放指令

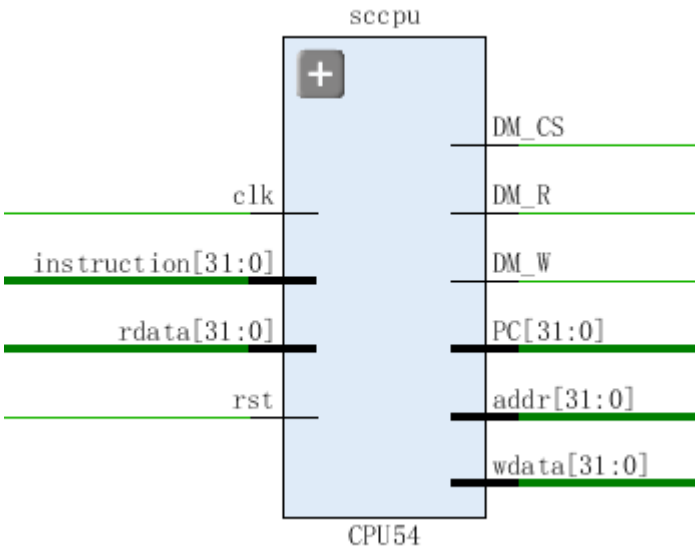
DMEM:
数据存储器，用于存放数据

CP0：
除了通常的运算功能之外，任何处理器都需要一些部件来处理中断、配置选项以及需要 某种机制来监控诸如片上高速缓存和定时器等功能。在 MIPS CPU 中，异常或者中断发生时的行为以及怎样处理都是由 CP0 控制寄存器和几条特殊指令来定义和控制的

四、应用程序

1. 应用程序流程图
2. 程序模块（对每个模块进行功能的说明，程序必须加注释）

CPU54




```
module CPU54(
input clk,
input rst,
input [31:0]instruction,//指令
input [31:0]rdata,//从内存中读取来的数据
output reg[31:0]PC,//指令的位置
output [31:0]addr,//在内存的中的位置
```

```
output reg[31:0]wdata,//存入内存的数据
output IM_R,//
output DM_CS,//内存片选有效
output DM_R,//内存读
output DM_W,//内存写
output intr,
output inta
      );
assign intr=1'b1;
assign inta=1'b1;
assign IM_R = 1'b1;
wire [3:0]aluc;
wire ALU_a;
wire ALU_b;
wire [31:0]NPC=PC+32'd4;
wire [31:0]alu_a;
wire [31:0]alu_b;
wire [31:0]alu_r;
wire zero;
wire rf_write;//寄存器堆写信号
wire [4:0]rs;
wire [4:0]rt;
wire [4:0]rd;
wire [4:0]rf_raddr1;
wire [4:0]rf_raddr2;
wire [4:0]rf_waddr;
wire [31:0]rf_rdata1;
wire [31:0]rf_rdata2;
reg [31:0]rf_wdata;
wire [2:0]rf_wd;
wire [1:0]rf_wa;
wire sign_extend;
wire [1:0]sb_r=addr[1:0];//判断 0 1 2 3
wire sh_r=addr[1];//判断  0 1
wire [1:0]lb_r=addr[1:0];
wire lh_r=addr[1];
wire  [31:0]MemDataS8  =  lb_r[1]?(lb_r[0]?{{24{rdata[31]}},  rdata[31:24]}:{{24{rdata[23]}},
rdata[23:16]}):(lb_r[0]?{{24{rdata[15]}}, rdata[15:8]}:{{24{rdata[7]}}, rdata[7:0]});
wire          [31:0]MemDataZ8          =          lb_r[1]?(lb_r[0]?{24'd0,          rdata[31:24]}:{24'd0,
rdata[23:16]}):(lb_r[0]?{24'd0, rdata[15:8]}:{24'd0, rdata[7:0]});
wire [31:0]MemDataS16 = lh_r?{{16{rdata[31]}}, rdata[31:16]}:{{16{rdata[15]}}, rdata[15:0]};
wire [31:0]MemDataZ16 = lh_r?{16'd0, rdata[31:16]}:{16'd0, rdata[15:0]};
wire jump_26;
wire beq;
```

```
wire bne;
wire jr;
wire bgez;
wire clz;
wire [31:0]clz_result;
wire div;
wire divu;
wire [31:0]q_div;
wire [31:0]r_div;
wire [31:0]q_divu;
wire [31:0]r_divu;
wire mfc0;
wire mtc0;
wire BREAK;
wire eret;
wire syscall;
wire teq;
wire [4:0]cause;
wire [31:0]rdata_cp0;
wire [31:0]exc_addr;
wire exception=BREAK|syscall|(teq&zero);
wire [31:0]status;
wire [15:0]instr_low16=instruction[15:0];
wire jump_16=(beq&zero)|(bne&(~zero))|(bgez&(~rf_rdata1[31]));
wire [25:0]instr_index=instruction[25:0];
wire [5:0]instr_op=instruction[31:26];
wire [31:0]EXTZ5={27'b0,instruction[10:6]};
wire [31:0]EXTZ16={16'b0,instr_low16};
wire [31:0]EXTS16={{16{instr_low16[15]}},instr_low16};
wire mthi;
wire mtlo;
wire jarl;
wire [1:0]RMemMode;
wire sign_lblh;
wire [1:0]WMemMode;
wire multu;
wire signed [31:0]mul_a=rf_rdata1;
wire signed [31:0]mul_b=rf_rdata2;
wire signed [31:0]mul_z = mul_a*mul_b;
wire [31:0]multu_a=rf_rdata1;
wire [31:0]multu_b=rf_rdata2;
wire [63:0]multu_z=rf_rdata1*rf_rdata2;
wire busy_div;
wire busy_divu;
```

```
reg start_div;
reg start_divu;
wire over_div;
wire over_divu;
wire busy=busy_div|busy_divu|(div&(~over_div))|(divu&(~over_divu));
always @(posedge clk or posedge rst) begin
    if (rst) begin
        start_div<=1'b0;
    end
    else if (!div) begin
        start_div<=1'b0;
    end
    else if (div&(~busy_div)&(~over_div)) begin
        start_div<=1'b1;
    end
    else begin
        start_div<=1'b0;
    end
end
always @(posedge clk or posedge rst) begin
    if (rst) begin
        start_divu<=1'b0;
    end
    else if (!divu) begin
        start_divu<=1'b0;
    end
    else if (divu&(~busy_divu)&(~over_divu)) begin
        start_divu<=1'b1;
    end
    else begin
        start_divu<=1'b0;
    end
end
always @(posedge clk or posedge rst) begin
    if (rst) begin
        PC<=32'h00400000;
    end
    else if (jump_16) begin
        PC<=NPC+{{14{instr_low16[15]}},instr_low16,2'b00};
    end
    else if (jump_26) begin
        PC<={PC[31:28],instr_index,{2'b00}};
    end
```

```
    else if (jr|jarl) begin
        PC<=rf_rdata1;
    end
    else if (exception| eret) begin
        PC<=exc_addr;
    end
    else if (busy) begin
        PC<=PC;
    end
    else begin
        PC<=NPC;
    end
end
reg [31:0]HI;
reg [31:0]LO;
always @(posedge clk or posedge rst) begin
    if(rst)begin
        HI<=32'h00000000;
    end
    else begin
        if (mthi)begin
            HI<=rf_rdata1;
        end
        else if (multu) begin
            HI<=multu_z[63:32];
        end
        else if (over_div) begin
            HI<=r_div;
        end
        else if (over_divu) begin
            HI<=r_divu;
        end
        else begin
            HI<=HI;
        end
    end
end

//LO  注意：上升沿修改。
always @(posedge clk or posedge rst) begin
    if(rst)begin
        LO<=32'h00000000;
    end
    else begin
```

```
        if (mtlo)begin
            LO<=rf_rdata1;
        end
        else if (multu) begin
            LO<=multu_z[31:0];
        end
        else if (over_div) begin
            LO<=q_div;
        end
        else if (over_divu) begin
            LO<=q_divu;
        end
        else begin
            LO<=LO;
        end
    end
end

assign rs=instruction[25:21];
assign rt=instruction[20:16];
assign rd=instruction[15:11];
wire rf_write_zero_check = rf_write & (rf_waddr==5'b0 ? 1'b0 :1'b1);
Regfiles
rfl(.clk(clk),.rst(rst),.wena(rf_write_zero_check),.raddr1(rf_raddr1),.raddr2(rf_raddr2),.waddr(rf_
waddr),.wdata(rf_wdata),.rdata1(rf_rdata1),.rdata2(rf_rdata2));
alu al(.a(alu_a),.b(alu_b),.aluc(aluc), .r(alu_r),.zero(zero),.carry(),.negative(),.overflow());
assign DM_CS = DM_W | DM_R;
assign rf_raddr1 = rs;
assign rf_raddr2 = rt;
assign rf_waddr = rf_wa[1]?(rf_wa[0]?5'd31:rt):(rf_wa[0]?rd:rt);
assign addr = rf_rdata1 + (sign_extend ? EXTS16:EXTZ16);
wire
[31:0]Wdata_sb=sb_r[1]?(sb_r[0]?{rf_rdata2[7:0],rdata[23:0]}:{rdata[31:24],rf_rdata2[7:0],rdata[
15:0]}):(sb_r[0]?{rdata[31:16],rf_rdata2[7:0],rdata[7:0]}:{rdata[31:8],rf_rdata2[7:0]});
wire [31:0]Wdata_sh= sh_r?{rf_rdata2[15:0],rdata[15:0]}:{rdata[31:16],rf_rdata2[15:0]};
reg [31:0]Wdata_sb;
always @(*) begin
    case(sb_r)
    2'b11:Wdata_sb={rf_rdata2[7:0],rdata[23:0]};
    2'b10:Wdata_sb={rdata[31:24],rf_rdata2[7:0],rdata[15:0]};
    2'b01:Wdata_sb={rdata[31:16],rf_rdata2[7:0],rdata[7:0]};
    2'b00:Wdata_sb={rdata[31:8],rf_rdata2[7:0]};
    default:Wdata_sb={rdata[31:8],rf_rdata2[7:0]};
    endcase
```

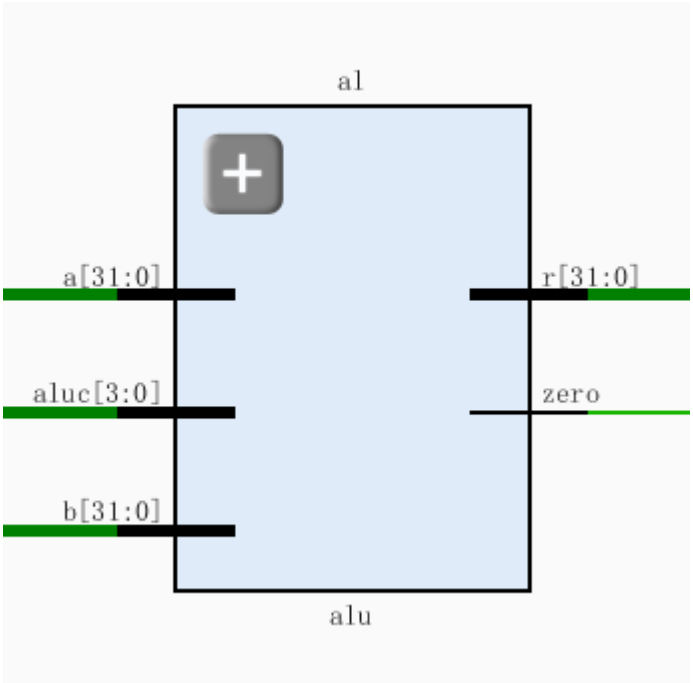
```
end
always @(*) begin
   case(WMemMode)
      2'b11:wdata=rf_rdata2;
      2'b10:wdata=rf_rdata2;
      2'b01:wdata=Wdata_sh;
      2'b00:wdata=Wdata_sb;
      default:wdata=rf_rdata2;

   endcase
end
always @(*) begin
   case(rf_wd)
      3'b111:rf_wdata=mul_z;
      3'b110:rf_wdata=clz_result;
      3'b101:rf_wdata=HI;
      3'b100:rf_wdata=LO;

      3'b010:rf_wdata=rdata_cp0;
      3'b011:rf_wdata=NPC;
      3'b000:
         case(RMemMode)
            2'b11:rf_wdata=rdata;
            2'b10:rf_wdata=rdata;
            2'b01:rf_wdata=sign_lblh?MemDataS16:MemDataZ16;
            2'b00:rf_wdata=sign_lblh?MemDataS8:MemDataZ8;
            default:rf_wdata=rdata;
         endcase
      3'b001:rf_wdata=alu_r;
      default:rf_wdata=alu_r;
   endcase
end

assign alu_a = ALU_a?rf_rdata1:EXTZ5;
assign alu_b = ALU_b?rf_rdata2:(sign_extend ? EXTS16:EXTZ16);
```

CONTROL

```
con_inst(.instruction(instruction),.aluc(aluc),.rf_write(rf_write),.DM_W(DM_W),.DM_R(DM_R),
.sign_extend(sign_extend),.ALU_a(ALU_a),.ALU_b(ALU_b),.rf_wd(rf_wd),
   .rf_wa(rf_wa),.jump_26(jump_26),.beq(beq),.bne(bne),.jr(jr),  .mfc0(mfc0),  .mtc0(mtc0),  .BRE
AK(BREAK), .eret(eret), .syscall(syscall), .teq(teq), .cause(cause), .mthi(mthi),
.mtlo(mtlo),.jarl(jarl),.RMemMode(RMemMode),.sign_lblh(sign_lblh),.WMemMode(WMemMo
de),.bgez(bgez),.clz(clz),.multu(multu),.div(div),.divu(divu));
```

```
CP0 cp0_inst(.clk(clk), .rst(rst), .mfc0(mfc0), .mtc0(mtc0), .pc(PC),
.Rd(rd),              //考虑   [2:0]   sel
.wdata(rf_rdata2),       // rt 中读取出来   rf_raddr2 = rt;
.exception(exception),
.eret(eret),
.cause(cause),
.intr(),
.rdata(rdata_cp0),           // Data from CP0 register for GP register
.status(status),
.timer_int(),
.exc_addr(exc_addr)        // Address for PC at the beginning of an exception
);

CLZ clz_inst(.in(rf_rdata1),.out(clz_result));
DIV
div_inst(.dividend(rf_rdata1),.divisor(rf_rdata2),.start(start_div),.clock(clk),.reset(rst),.q(q_div),.r(
r_div),.busy(busy_div),.over(over_div));
DIVU
divu_inst(.dividend(rf_rdata1),.divisor(rf_rdata2),.start(start_divu),.clock(clk),.reset(rst),.q(q_divu
),.r(r_divu),.busy(busy_divu),.over(over_divu));
endmodule
```

ALU




```
module alu(
input [31:0] a, //32  位输入，操作数 1
```

```
input [31:0] b, //32 位输入，操作数 2
input [3:0] aluc, //4 位输入，控制 alu 的操作
output reg [31:0] r, //32 位输出，由 a、b 经过 aluc 指定的操作生成
output reg zero, //0 标志位
output reg carry, // 进位标志位
output reg negative, // 负数标志位
output reg overflow // 溢出标志位
);

reg signed [31:0] alg;
reg [32:0] temp;
always @(*)
begin
casex(aluc)
4'b0000://Addu
    begin
        temp=a+b;
        r=temp;
        if(r==0)
            zero=1;
        else
            zero=0;
        if(temp[32])
            carry=1;
        else
            carry=0;
        if(r[31])
            negative=1;
        else
            negative=0;
        overflow=0;
    end
4'b0010://Add
    begin
        r=a+b;
        if(r==0)
            zero=1;
        else
            zero=0;
        if(!a[31]&&!b[31]&&r[31]||a[31]&&b[31]&&!r[31])//(正+负和负+正不会溢出),正+
正||负+负
            overflow=1;
        else
            overflow=0;
```

```
            if(r[31])
                negative=1;
            else
                negative=0;
            carry=0;
        end
    4'b0001://Subu
        begin
            r=a-b;
            if(a<b)
                carry=1;
            else
                carry=0;
            if(r==0)
                zero=1;
            else
                zero=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            overflow=0;
        end
    4'b0011://Sub
        begin
            r=a-b;
            if(!a[31]&&b[31]&&r[31]||a[31]&&!b[31]&&!r[31])//（负-负和正-正不会溢出），正-负||负-正可能溢出
                overflow=1;
            else
                overflow=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            if(r==0)
                zero=1;
            else
                zero=0;
            carry=0;
        end
    4'b0100://And
        begin
        r=a&b;
```

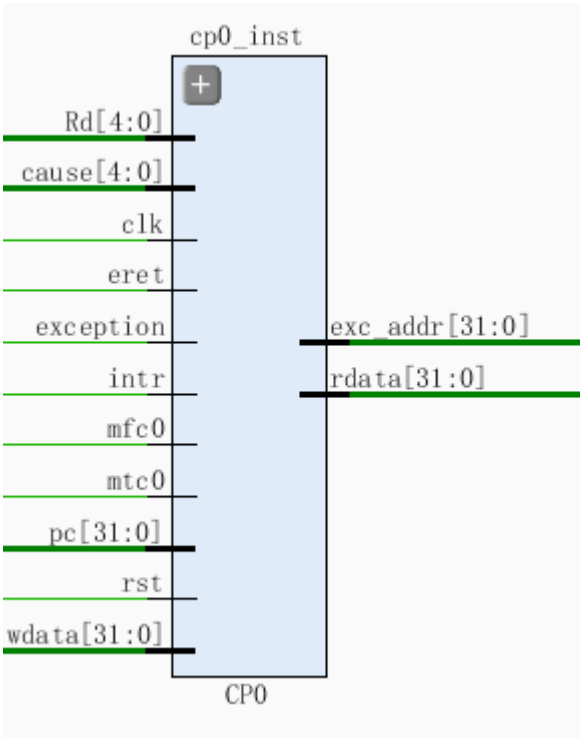
```
        if(r==0)
            zero=1;
        else
            zero=0;
        if(r[31])
            negative=1;
        else
            negative=0;
        carry=0;
        overflow=0;
        end
    4'b0101://Or
        begin
        r=a|b;
        if(r==0)
            zero=1;
        else
            zero=0;
        if(r[31])
            negative=1;
        else
            negative=0;
        carry=0;
        overflow=0;
        end
    4'b0110://Xor
        begin
            r=a^b;
            if(r==0)
                zero=1;
            else
                zero=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            carry=0;
            overflow=0;
        end

    4'b0111://Nor
        begin
            r=~(a|b);
            if(r==0)
```

```
                zero=1;
            else
                zero=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            carry=0;
            overflow=0;
        end
4'b100x://Lui
        begin
            r={b[15:0],16'b0};
            if(r==0)
                zero=1;
            else
                zero=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            carry=0;
            overflow=0;
        end
4'b1011://Slt
        begin
            if(a[31]&&!b[31]||!a[31]&&!b[31]&&a<b||a[31]&&b[31]&&a[30:0]<b[30:0])// 负 < 正 ||
正<正||负<负  (正恒>负)
                r=1;
            else
                r=0;
            if(a==b)
                zero=1;
            else
                zero=0;
            negative=r;
            carry=0;
            overflow=0;
        end
4'b1010://Sltu
        begin
            if(a<b)
                begin
                    carry=1;
```

```
                r=1;
            end
        else
            begin
                carry=0;
                r=0;
            end
        zero=a==b?1:0;
        negative=0;
        overflow=0;
    end
4'b1100://Sra
    begin
        alg=b;
        r=alg>>>a[4:0];//调试的时候注意一下这个地方是否有问题
        if(a<=32&&a>0)
            carry=b[a-1];
        else
            carry=b[31];
        if(r==0)
            zero=1;
        else
            zero=0;
        if(r[31])
            negative=1;
        else
            negative=0;
        overflow=0;
    end
4'b111x://Sll//Slr
    begin
        r=b<<a[4:0];
        if(a<=32&&a>0)
            carry=b[32-a];
        else
            carry=0;
        if(r==0)
            zero=1;
        else
            zero=0;
        if(r[31])
            negative=1;
        else
            negative=0;
```

```
            overflow=0;
        end
4'b1101://Srl
        begin
            r=b>>a[4:0];
            if(a<32&&a>0)
                carry=b[a-1];
            else
                carry=0;
            if(r==0)
                zero=1;
            else
                zero=0;
            if(r[31])
                negative=1;
            else
                negative=0;
            overflow=0;
        end
endcase
end
endmodule
```

CP0：

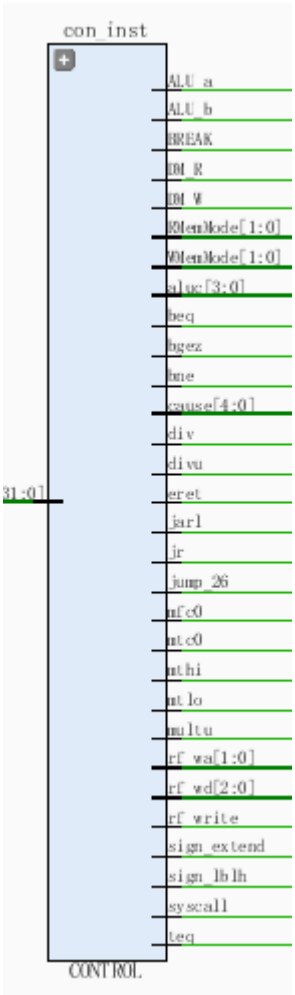



```
module CP0(
input clk,
```

```
input rst,
input mfc0,                    // CPU instruction is Mfc0
input mtc0,                    // CPU instruction is Mtc0
input [31:0]pc,
input [4:0] Rd,               // Specifies Cp0 register
input [31:0] wdata,         // Data from GP register to replace CP0 register
input exception,
input eret,                    // Instruction is ERET (Exception Return)
input [4:0]cause,
input intr,
output [31:0] rdata,           // Data from CP0 register for GP register
output [31:0] status,
output reg timer_int,
output [31:0]exc_addr        // Address for PC at the beginning of an exception
);

//syscall>break>teq>eret
reg [31:0]CP0_array_reg[31:0];
assign status=CP0_array_reg[12];
assign exc_addr=eret?CP0_array_reg[14]:32'd4;
assign rdata=mfc0?CP0_array_reg[Rd]:32'b0;


wire syscall =(cause==5'b01000)?1'b1:1'b0;
wire break =(cause==5'b01001)?1'b1:1'b0;
wire teq =(cause==5'b01101)?1'b1:1'b0;

wire exception_excute= exception & CP0_array_reg[12][0] & ((CP0_array_reg[12][1] & syscall) |
(CP0_array_reg[12][2] & break) | (CP0_array_reg[12][3] & teq));
reg sll_5;

integer i,j;
always @(posedge clk or posedge rst) begin
   if (rst) begin
      for(i=0;i<11;i=i+1)begin
         CP0_array_reg[i]<=32'b0;
      end
      for(j=13;j<32;j=j+1)begin
         CP0_array_reg[j]<=32'b0;
      end
      CP0_array_reg[12]<=32'h0000000f;
      sll_5<=1'b0;
   end
   else begin
```

```
        if (exception_excute & (~sll_5))begin
            CP0_array_reg[12]<=CP0_array_reg[12]<<5;
            CP0_array_reg[13][6:2]<=cause;
            CP0_array_reg[14]<=pc;
            sll_5<=1'b1;
        end
        if(eret & sll_5)begin
            CP0_array_reg[12]<=CP0_array_reg[12]>>5;
            sll_5<=1'b0;
        end
        if (mtc0) begin
            CP0_array_reg[Rd]<=wdata;
        end
    end
end
endmodule
```

CONTROL:




```
module CONTROL(
```

```
input [31:0]instruction,
output reg[3:0]aluc,
output rf_write,
output DM_W,
output DM_R,
output sign_extend,
output ALU_a,
output ALU_b,
output reg[2:0]rf_wd,
output reg[1:0]rf_wa,
output jump_26,
output beq,
output bne,
output jr,
output mfc0,
output mtc0,
output BREAK,
output eret,
output syscall,
output teq,
output [4:0]cause,
output mthi,
output mtlo,
output jarl,
output reg[1:0]RMemMode,
output sign_lblh,
output reg[1:0]WMemMode,
output bgez,
output clz,
output multu,
output div,
output divu
);

wire [5:0]instr_op=instruction[31:26];
wire [5:0]instr_func=instruction[5:0];
wire sw=instr_op==6'b101011;
wire lw=instr_op==6'b100011;
wire ori=instr_op==6'b001101;
wire sll=(instr_op==6'b000000) & (instr_func==6'b000000);
wire srl=(instr_op==6'b000000) & (instr_func==6'b000010);
wire sra=(instr_op==6'b000000) & (instr_func==6'b000011);
wire addu=(instr_op==6'b000000) & (instr_func==6'b100001);
wire subu=(instr_op==6'b000000) & (instr_func==6'b100011);
```

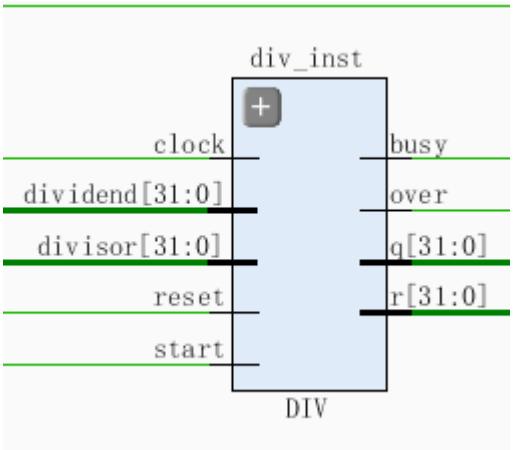
```
wire add=(instr_op==6'b000000) & (instr_func==6'b100000);
wire sub=(instr_op==6'b000000) & (instr_func==6'b100010);
wire AND=(instr_op==6'b000000) & (instr_func==6'b100100);
wire OR=(instr_op==6'b000000) & (instr_func==6'b100101);
wire XOR=(instr_op==6'b000000) & (instr_func==6'b100110);
wire NOR=(instr_op==6'b000000) & (instr_func==6'b100111);
wire slt=(instr_op==6'b000000) & (instr_func==6'b101010);
wire sltu=(instr_op==6'b000000) & (instr_func==6'b101011);
wire sllv=(instr_op==6'b000000) & (instr_func==6'b000100);
wire srlv=(instr_op==6'b000000) & (instr_func==6'b000110);
wire srav=(instr_op==6'b000000) & (instr_func==6'b000111);
wire addi=instr_op==6'b001000;
wire addiu=instr_op==6'b001001;
wire andi=instr_op==6'b001100;
wire xori=instr_op==6'b001110;
wire slti=instr_op==6'b001010;
wire sltiu=instr_op==6'b001011;
wire lui=instr_op==6'b001111;
wire jal=instr_op==6'b000011;
assign bgez=instr_op==6'b000001;
assign div=(instr_op==6'b000000) & (instr_func==6'b011010);
assign divu=(instr_op==6'b000000) & (instr_func==6'b011011);
assign mthi=(instr_op==6'b000000) & (instr_func==6'b010001);
assign mtlo=(instr_op==6'b000000) & (instr_func==6'b010011);
wire mfhi=(instr_op==6'b000000) & (instr_func==6'b010000);
wire mflo=(instr_op==6'b000000) & (instr_func==6'b010010);
assign jarl=(instr_op==6'b000000) & (instr_func==6'b001001);
assign clz=(instr_op==6'b011100) & (instr_func==6'b100000);
wire mul=(instr_op==6'b011100) & (instr_func==6'b000010);
assign multu=(instr_op==6'b000000) & (instr_func==6'b011001);
wire lb=instr_op==6'b100000;
wire lh=instr_op==6'b100001;
wire lbu=instr_op==6'b100100;
wire lhu=instr_op==6'b100101;
wire sb=instr_op==6'b101000;
wire sh=instr_op==6'b101001;
//CP0 -SIGNAL
assign mfc0=instruction[31:21]==11'b01000000000;
assign mtc0=instruction[31:21]==11'b01000000100;
assign syscall=(instr_op==6'b000000) & (instr_func==6'b001100);
assign BREAK=(instr_op==6'b000000) & (instr_func==6'b001101);
assign teq=(instr_op==6'b000000) & (instr_func==6'b110100);
assign eret=(instr_op==6'b010000) & (instr_func==6'b011000);//ERETNC 重复了
```

```
    assign jr=(instr_op==6'b000000) && (instr_func==6'b001000);
    assign jump_26=(instr_op==6'b000010)||jal;//j    jal
    assign beq =instr_op==6'b000100;
    assign bne =instr_op==6'b000101;
    assign cause = syscall?5'b01000:(BREAK?5'b01001:(teq?5'b01101:5'b00000));
    assign ALU_a=sll|srl|sra?0:1;
    assign ALU_b=ori|addi|addiu|andi|xori|slti|sltiu|lui?0:1; //1-rf_rd2    0-s_z_extend
    // assign RMemMode = lw?2'b11:(lb|lbu?2'b00:(lh|lhu?2'b01:2'b10));
    always @(*) begin
        if(lw)
            RMemMode=2'b11;
        else if(lb|lbu)
            RMemMode=2'b00;
        else if (lh|lhu)
            RMemMode=2'b01;
        else
            RMemMode=2'b10;
    end
    // assign WMemMode = sw?2'b11:(sb?2'b00:(sh?2'b01:2'b10));
    always @(*) begin
        if(sw)
            WMemMode=2'b11;
        else if(sb)
            WMemMode=2'b00;
        else if (sh)
            WMemMode=2'b01;
        else
            WMemMode=2'b10;
    end

    assign rf_write = lw|lb|lh|lbu|lhu|addu|subu|sll|ori|add|sub|AND|OR|XOR|NOR|slt|sltu|srl|sra|sllv|srlv|srav|addi|addiu|andi|xori|slti|sltiu|lui|jal|mfc0|mfhi|mflo|jarl|clz|mul?1:0;
    assign DM_W = sw|sb|sh?1:0;
    assign DM_R = lw|lb|lh|lbu|lhu?1:0;
    always @(*) begin
        if (mul)
            rf_wd=3'b111;
        else if(clz)
            rf_wd=3'b110;
        else if(mfhi)
            rf_wd=3'b101;
        else if(mflo)
```

```
            rf_wd=3'b100;
        else if(mfc0)
            rf_wd=3'b010;
        else if(jal|jarl)
            rf_wd=3'b011;
        else if(lw|lb|lh|lbu|lhu)
            rf_wd=3'b000;
        else
            rf_wd=3'b001;
    end

    always @(*) begin
        if(jal)
            rf_wa=2'b11;
        else if(mfc0)
            rf_wa=2'b10;
        else if(lw|lb|lh|lbu|lhu|ori|addi|addiu|andi|xori|slti|sltiu|lui)
            rf_wa=2'b00;
        else
            rf_wa=2'b01;
    end

    //                assign                rf_wa                =
jal?(2'b11):(mfc0?2'b10:(lw|lb|lh|lbu|lhu|ori|addi|addiu|andi|xori|slti|sltiu|lui?2'b00:2'b01));



    //0 零扩展   1 符号扩展
    assign sign_extend = ori|andi|xori?0:1;
    assign sign_lblh = lb|lh?1:0;



    //处理 aluc 信号
    always @(*) begin
        case(instruction[31:26])
            6'b000000:
                case(instruction[5:0])
                    6'b100000:aluc=4'b0010;//add
                    6'b100001:aluc=4'b0000;//addu
                    6'b100010:aluc=4'b0011;//sub
                    6'b100011:aluc=4'b0001;//subu
                    6'b100100:aluc=4'b0100;//and
```

```
                    6'b100101:aluc=4'b0101;//or
                    6'b100110:aluc=4'b0110;//xor
                    6'b100111:aluc=4'b0111;//nor
                    6'b101010:aluc=4'b1011;//slt
                    6'b101011:aluc=4'b1010;//sltu


                    6'b000000:aluc=4'b111x;//sll
                    6'b000010:aluc=4'b1101;//srl
                    6'b000011:aluc=4'b1100;//sra
                    6'b000100:aluc=4'b111x;//sllv
                    6'b000110:aluc=4'b1101;//srlv
                    6'b000111:aluc=4'b1100;//srav

                    6'b110100:aluc=4'b0011;//teq
                    default:aluc=4'b0010;
                endcase
            6'b001000:aluc=4'b0010;//addi
            6'b001001:aluc=4'b0000;//addiu
            6'b001100:aluc=4'b0100;//andi
            6'b001101:aluc=4'b0101;//ori
            6'b001110:aluc=4'b0110;//xori
            6'b000100:aluc=4'b0011;//beq
            6'b000101:aluc=4'b0011;//bne
            6'b001010:aluc=4'b1011;//slti
            6'b001011:aluc=4'b1010;//sltiu
            6'b001111:aluc=4'b100x;//lui
            default:aluc=4'b0010;
        endcase
    end
endmodule
```

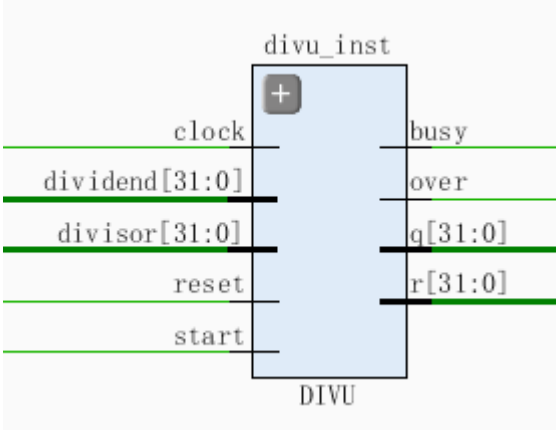
DIV:



```
module DIV(
    input signed [31:0]dividend,//被除数
    input signed [31:0]divisor,//除数
    input start,//启动除法运算
    input clock,
    input reset,
    output [31:0]q,//商
    output reg [31:0]r,//余数
    output reg busy,//除法器忙标志位
    output reg over
);
reg[5:0]count;
reg signed [31:0] reg_q;
reg signed [31:0] reg_r;
reg signed [31:0] reg_b;
reg r_sign;

wire [32:0] sub_add = r_sign?({reg_r,q[31]} + {1'b0,reg_b}):({reg_r,q[31]} - {1'b0,reg_b});//加、减法器

// assign q = reg_q;
// wire signed[31:0] tq=(dividend[31]^divisor[31])?(-reg_q):reg_q;
assign q = reg_q;
always @ (posedge clock or posedge reset)
begin
    if (reset)
        begin//重置
            count <=0;
            busy <= 0;
            over<=0;
        end
    else
        begin
            if (start)
                begin//开始除法运算，初始化
                    reg_r <= 0;
                    r_sign <= 0;
                    count <= 0;
                    busy <= 1;
                    if(dividend<0)
                        reg_q <= -dividend;
                    else
                        reg_q <= dividend;
                    if(divisor<0)
```

```
                    reg_b <= -divisor;
                else
                    reg_b <= divisor;
            end
        else if (busy)
            begin
                if(count<=31)
                    begin
                        reg_r <= sub_add[31:0];//部分余数
                        r_sign <= sub_add[32];//如果为负，下次相加
                        reg_q <= {reg_q[30:0],~sub_add[32]};//上商
                        count <= count +1;//计数器加一
                    end
                else
                    begin
                        if(dividend[31]^divisor[31])
                            reg_q<=-reg_q;
                        if(!dividend[31])
                            r<=r_sign? reg_r + reg_b : reg_r;
                        else
                            r<=-(r_sign? reg_r + reg_b : reg_r);
                        busy <= 0;
                        over <= 1;
                    end
            end
        else
        over<=0;
    end
end
endmodule
```

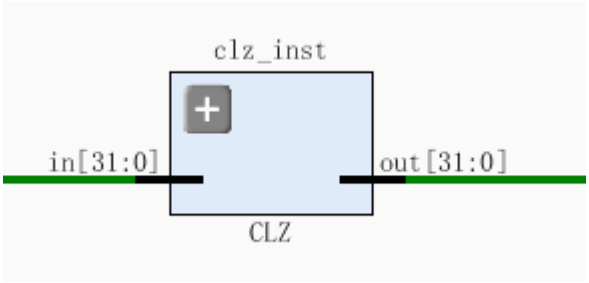
DIVU:



```
module DIVU(
    input [31:0]dividend,          //被除数
    input [31:0]divisor,           //除数
    input start,                   //启动除法运算
    input clock,
    input reset,
    output [31:0]q,                //商
    output [31:0]r,                //余数
    output reg busy,                    //除法器忙标志位
    output reg over
    );

    reg [4:0]count;
    reg [31:0] reg_q;
    reg [31:0] reg_r;
    reg [31:0] reg_b;
    reg r_sign;
    wire [32:0] sub_add = r_sign?({reg_r,q[31]} + {1'b0,reg_b}):({reg_r,q[31]} - {1'b0,reg_b});
//加、减法器
    assign r = r_sign? reg_r + reg_b : reg_r;
    assign q = reg_q;


    always @(posedge clock or posedge reset) begin
        if (reset) begin
            busy<=0;
            count<=0;
            over<=0;
        end
        else begin
            if (start) begin
                reg_q<=dividend;
                reg_b<=divisor;
                reg_r<=32'b0;
                count<=0;
                busy<=1;
                r_sign<=0;
            end
            else if (busy) begin
                reg_r<=sub_add[31:0];
                reg_q<={reg_q[30:0],~sub_add[32]};
                r_sign<=sub_add[32];
                count<=count +5'b1;
                if(count == 5'd31)begin
```

```
                busy<=0;
                over<=1;
            end
        end
        else begin
            over<=0;
        end
    end

    end
endmodule
```

CLZ:




```
module CLZ(
input [31:0]in,
output reg[31:0]out
    );

always @(*) begin
    if (in[31])
        out = 32'd0;
    else if (in[30])
        out = 32'd1;
    else if(in[29])
        out = 32'd2;
    else if(in[28])
        out = 32'd3;
    else if(in[27])
        out = 32'd4;
    else if(in[26])
        out = 32'd5;
    else if(in[25])
        out = 32'd6;
    else if(in[24])
        out = 32'd7;
    else if(in[23])
```

```
        out = 32'd8;
else if(in[22])
        out = 32'd9;
else if(in[21])
        out = 32'd10;
else if(in[20])
        out = 32'd11;
else if(in[19])
        out = 32'd12;
else if(in[18])
        out = 32'd13;
else if(in[17])
        out = 32'd14;
else if(in[16])
        out = 32'd15;
else if(in[15])
        out = 32'd16;
else if(in[14])
        out = 32'd17;
else if(in[13])
        out = 32'd18;
else if(in[12])
        out = 32'd19;
else if(in[11])
        out = 32'd20;
else if(in[10])
        out = 32'd21;
else if(in[9])
        out = 32'd22;
else if(in[8])
        out = 32'd23;
else if(in[7])
        out = 32'd24;
else if(in[6])
        out = 32'd25;
else if(in[5])
        out = 32'd26;
else if(in[4])
        out = 32'd27;
else if(in[3])
        out = 32'd28;
else if(in[2])
        out = 32'd29;
else if(in[1])
```

```
            out = 32'd30;
        else if(in[0])
            out = 32'd31;
        else
            out = 32'd32;
end

endmodule
```

运用小程序：

1. vga

```
module vga(
input clk_in,//50M
input clk_in_25,
input rst_in,
input[31:0]i_data,
input we,
output hsync,
output vsync,
output [3:0]vga_r,
output [3:0]vga_g,
output [3:0]vga_b,
output intr,
output [31:0]o_data
        );

wire clk=clk_in_25;
wire rst=rst_in;

wire vga_valid;
wire[31:0]show_data;
wire restart;

reg[31:0]data2;

// data2
//cpu 传来写信号  50M
always @(posedge clk_in or posedge rst) begin
    if (rst) begin
//        data2<=32'd0;
          data2 <=32'hffffffff;
    end
    else if (we) begin
        data2<=i_data;//CPU 写入 I/O
```

```
    end
    else begin
        data2<=data2;
    end
end

assign o_data={{31{1'b0}},restart};

VGA_640x480 vga_6_4_inst(
.clk(clk),
.rst(rst),
.data2(data2),
.HS(hsync),
.VS(vsync),
.valid(vga_valid),
.o_data(show_data),
.intr(intr),
.rstart(restart));


VGA_Color vc_inst(
.clk(clk),
.rst(rst),
.valid(vga_valid),
.data(show_data),
.red(vga_r),
.green(vga_g),
.blue(vga_b));

Endmodule
```

2. vga_color

```
module VGA_Color(
input clk,
input rst,
input valid,
input [31:0]data,
output reg[3:0]red,
output reg[3:0]green,
output reg[3:0]blue
        );

reg [4:0]count;//at most of 31
always @(posedge clk or posedge rst) begin
```

```
        if (rst) begin
            // reset
            count<=0;
        end
        else if (valid) begin
            count<=count+1;
        end
        else begin
            count <=0;
        end
end

always @(*) begin
        if (valid) begin
            if(data[31-count]==1)begin
                red=4'b0000;green=4'b0000;blue=4'b0000;//black
            end
            else begin
                red=4'b1111;green=4'b1111;blue=4'b1111;//white
            end
        end
        else begin
             red=4'b0000;green=4'b0000;blue=4'b0000;
        end
end
endmodule
```

3. vga_640x480

```
module VGA_640x480(
        input wire clk,//分频的时钟，频率为 25mhz
        input wire rst,
        input [31:0] data2,
        output HS,
        output VS,
        output valid,
        output [31:0] o_data,
        output intr,
        output rstart
        );

        reg [31:0]h_count;
        reg [31:0]v_count;
        wire[9:0]xpos;
        wire[9:0]ypos;
```

```
localparam vga_x=640;
localparam vga_y=480;
localparam pict_x=640;
localparam pict_y=480;

localparam pict_num=1;
localparam play_time_num=4;// every 5 pictures is played. the display time between two pictures is less than 0.1
// localparam addr_num=pict_x/32*pict_y;

reg[14:0]addr;
reg[3:0]pict_count;
reg[4:0]play_time_count;
wire[31:0]tdata;

reg[31:0]data1;
reg req;//32 位的数据用完，向 cpu 请求数据
reg ini;//初始化，向 cpu 请求数据
reg flag;
reg restart;
// reg flag_2_to_1;//rst 之后 把 data2 给 data1
assign rstart = restart;

// assign o_data = ((xpos<pict_x) && (ypos < pict_y))?tdata:32'd0;
// assign o_data = (play_time_count>0)?32'd0:(((xpos<pict_x) && (ypos < pict_y))?data1:32'd0);
assign o_data = (play_time_count>0)?32'd0:(((xpos<pict_x) && (ypos < pict_y))?((xpos%32==0)?data2:data1):32'd0);




//ini flag
always @(posedge clk or posedge rst) begin
    if (rst) begin
        ini<=0;
        flag<=0;
    end
    else if (h_count== 10'd96 && flag==0) begin
        ini<=1;
        flag<=1;
    end
    else begin
```

```
            ini<=0;
        end
    end


    assign intr = ini|req;




    // req
    // data1
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            req<=0;
            data1<=32'hffffffff;
//          data1<=32'h0;
        end
        else if (valid) begin
            if (xpos % 32 == 0 && (xpos<pict_x) && (ypos <pict_y)) begin//进一步判读是否需要发出 req
                if ((ypos == pict_y - 1) && (xpos == pict_x - 32)) begin//在一帧图像的结尾
                    if(play_time_count==play_time_num-1)begin//一帧图播放到最后一次
                        req<=1;
                        data1<=data2;
                    end
                    else begin
                        req<=0;
                        data1<=data2;
                    end
                end
                else begin
                    if (play_time_count == 0) begin
                        req<=1;
                        data1<=data2;
                    end
                    else begin
                        req<=0;
                        data1<=data1;//  play_time_count>0  的时候  上面已经操作过使得o_data=0
                    end
                end
            end
            else begin
```

```
                req<=0;
                data1<=data1;
            end
        end
        else begin
            req<=0;
            data1<=data1;
        end
    end

    // restart
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            restart<=0;
        end
        else if (valid && (xpos % 32 == 0) && (ypos == pict_y - 1) && (xpos == pict_x - 32)
&& (play_time_count==play_time_num-1) &&(pict_count == pict_num - 1)) begin
            restart<=1;
        end
        else if(v_count == 0)begin
            restart<=0;
        end
        else begin
            restart<=restart;
        end
    end




    // // addr
    // always @(posedge clk or posedge rst) begin
    //     if (rst) begin
    //         addr<=0;
    //     end
    //     else if (valid) begin
    //         if (xpos % 32 == 31 && (xpos<pict_x) && (ypos <pict_y)) begin
    //             if ((ypos == pict_y - 1) && (xpos == pict_x -1)) begin
    //                 if(play_time_count==play_time_num-1)begin
    //                     if(pict_count == pict_num - 1)begin
    //                         addr<=0;
    //                     end
    //                     else begin
    //                         addr<=addr+1;
```

```
//                              end
//                          end
//                          else begin
//                              addr<=addr;//
//                          end
//                      end
//                      else begin
//                          if (play_time_count == 0) begin
//                              addr<=addr+1;
//                          end
//                          else begin
//                              addr<=addr;
//                          end

//                      end
//              end
//      end
//      else begin
//              addr<=addr;
//      end
// end

//play_time_count
always @(posedge clk or posedge rst) begin
        if (rst) begin
                // reset
                play_time_count<=0;
        end
        else if ((ypos == vga_y - 1) && (xpos == vga_x -1)) begin
                if(play_time_count == play_time_num-1)begin
                        play_time_count<=0;
                end
                else begin
                        play_time_count<=play_time_count+1;
                end
        end
        else begin
                play_time_count<=play_time_count;
        end
end

// pict_count
always @(posedge clk or posedge rst) begin
        if (rst) begin
```

```
            pict_count<=0;
        end
        else if ((ypos == vga_y - 1) && (xpos == vga_x -1) && (play_time_count == play_time_num-1)) begin
            if(pict_count == pict_num - 1)begin
                pict_count<=0;
            end
            else begin
                pict_count<=pict_count+1;
            end
        end
        else begin
            pict_count<=pict_count;
        end
    end



    //行计数：h_count(0-639+8+8+96+40+8 = 799)
    always@(posedge clk or posedge rst)begin
        if (rst) begin
            h_count<=0;
        end
        else if(h_count == 10'd799)
            h_count <= 10'h0;
        else
            h_count <= h_count + 10'h1;
    end


    //帧计数：v_count(0-524)
    always@(posedge clk or posedge rst)begin
        if (rst) begin
            v_count<=0;
        end
        else if(h_count == 10'd799)begin
            if(v_count == 10'd524)v_count <= 10'h0;
            else v_count <= v_count + 10'h1;
        end
    end

    assign xpos = valid?(h_count - 10'd143):0;
    assign ypos = valid?(v_count - 10'd35):0;
```

```
        assign VS = (v_count >= 10'd2);
        assign HS = (h_count >= 10'd96);
        assign valid = (((h_count >= 10'd143)&&(h_count < 10'd783)) && ((v_count >= 10'd35)
&& (v_count < 10'd515)));
endmodule
```

4. io_sel

```
module io_sel(
        input [31:0] addr,
          input cs,
          input sig_w,
          input sig_r,
          output dmem_cs,
          output vga_cs
        );

// assign seg7_cs = (addr == 32'h10010000 && cs == 1 && sig_w == 1) ? 1 : 0;
// assign switch_cs = (addr == 32'h10010010 && cs == 1 && sig_r == 1) ? 1 : 0;
        assign vga_cs = ((addr == 32'h10810000 | addr == 32'h10810004)&& cs == 1) ? 1 : 0;
        assign dmem_cs = cs & (~vga_cs);
endmodule
```

5. sscpu_dataflow

```
module sccomp_dataflow(
input clk_in,
input reset,
input [15:0] sw,
output [7:0] o_seg,
output [7:0] o_sel,
output hsync,
output vsync,
output [3:0]vga_r,
output [3:0]vga_g,
output [3:0]vga_b
        );
wire locked;
wire exc;
wire [31:0]status;

wire [31:0]rdata;
wire [31:0]wdata;
wire IM_R,DM_CS,DM_R,DM_W;
wire [31:0]inst,pc,addr;
wire inta,intr;
```

```
wire clk;
wire [31:0]data_fmem;
wire [31:0]data_fvga;
wire rst=reset|~locked;
wire [31:0]ip_in;
wire seg7_cs,switch_cs;

assign ip_in = pc-32'h00400000;

wire dmem_cs;
wire vga_cs;
wire clk_vga;

clk_wiz_0 clk_inst
     (
       // Clock out ports
       .clk_out1(clk),          // output clk_out1
       .clk_out2(clk_vga),          // output clk_out2
       // Status and control signals
       .reset(reset), // input reset
       .locked(locked),              // output locked
     // Clock in ports
       .clk_in1(clk_in));

//clk_div #(3)cpu_clk(clk_in,clk);

/*地址译码*/
// io_sel io_mem(addr, DM_CS, DM_W, DM_R, seg7_cs, switch_cs);
io_sel io_mem(
     .addr(addr),
     .cs(DM_CS),
     .sig_w(DM_W),
     .sig_r(DM_R),
     .dmem_cs(dmem_cs),
     .vga_cs(vga_cs)
       );
```

Mips 源程序：

```
j _start
sll $0, $0, 0
j _exceptions
sll $0, $0, 0
```

```
_start:
lui $a0,0x1001 #a0 为 dmem 取数首地址
lui $a1,0x1081 #a1 为 vga in  地址

lui $a2,0x1081
addiu $a2,$a2,0x0004 #a2 为 vga status  地址

lui $a3,0x1001 #a3 为数据地址
ori $s1,$0,0x8000 #s1 用来判断数据是否压缩
ori $s2,$0,0x4000 #s1 用来判断数据发生压缩时，是 0 还是 1
lui $s6,0x8000 #s6  恒等于 0x8000_0000
sra $s7,$s6,31        # s7=ffff_ffff
ori $t8,$0,0x0020 # t8 恒为 32
ori $t9,$0,0x001f # t9 恒为 31
ori $t7,$0,0x3fff # t7  恒为    0011_1111_1111_1111



cycle:
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
ori $v1,$0,0x0020
j cycle

_exceptions:
#先选 VGA 状态寄存器地址，查看是否需要把数据地址置为 0
lw $t0,($a2)
and $s5,$0,$s0
beq $t0,$0,no_restart #等于 0 的时候不需要重置，否则重置
#重新播放图片  寄存器全部重置
addu $a3,$a0,$0
and $v0,$0,$0



no_restart:
bne $v0,$0,no_fatch #先判断是否有剩余位
fatch_num:
lhu $s0,($a3) #此时要取数    半字无符号扩展
addiu    $a3,$a3,0x0002 #地址+2
and $t1,$s1,$s0 #判断是否压缩
sltu $s3,$t1,$s1 #0000000 -1      , 0000_8000 - 0

beq $s3,$0,label3
```

```
#不压缩
and $s0,$s0,$t7 #留下 14 位有效
addiu $v0,$0,0x000e # v0 =14
j no_fatch

label3:
#压缩
and $t1,$s2,$s0 #判断压缩 0/1
sltu $s4,$t1,$s2 #0000000 -1      , 0000_4000 - 0
and $v0,$s0,$t7 #留下 14 位有效



no_fatch:
bne $s3,$0,no_compress

compress:
sltu $t0,$v0,$v1 # v0<v1 -- 1      , v0>=v1 ---0
bne $t0,$0,cp_2

#v1<=v0
sllv $s5,$s5,$v1    #把输出结果左移
and $t2,$0,$0 #置 0，保证压缩 0 的时候也能得到正确结果
bne $s4,$0,label1
or $t2,$0,$s7
beq $v1,$t8,label1 #v1=32 时的特殊处理
#对 1 的压缩

subu $t1,$t9,$v1
srav $t2,$s6,$t1
xor $t2,$t2,$s7

label1:
or $s5,$s5,$t2 #拼装出最后结果
subu $v0,$v0,$v1
j output

cp_2:
# v0<v1
sllv $s5,$s5,$v0    #把输出结果左移
and $t2,$0,$0 #置 0，保证压缩 0 的时候也能得到正确结果
bne $s4,$0,label2
#对 1 的压缩
subu $t1,$t9,$v0
srav $t2,$s6,$t1
```

```
xor $t2,$t2,$s7

label2:
or $s5,$s5,$t2 #拼装出最后结果
subu $v1,$v1,$v0
j fatch_num


no_compress:
sltu $t0,$v0,$v1 # v0<v1 -- 1      , v0>=v1 ---0
beq $t0,$0,no_cp_1

# v0<v1
sllv $s5,$s5,$v0    #把输出结果左移
or $s5,$s5,$s0    #拼装出 S5
subu $v1,$v1,$v0 #修改剩余  需要的拼装位数
j fatch_num

no_cp_1:
#v1<=v0
sllv $s5,$s5,$v1    #把输出结果左移
subu $v0,$v0,$v1
srlv $t2,$s0,$v0 #右移两个差  位
or $s5,$s5,$t2 #拼装出最后结果
#s0 保留  v0 位
subu $t2,$t8,$v0
sllv $s0,$s0,$t2
srlv $s0,$s0,$t2

output:
sw $s5,($a1) #把数据送到 vga
#j _epc_plus4

#_epc_plus4:
#sll $0, $0, 0
mfc0 $k0, $14
addi $k0, $k0, 0x4
mtc0 $k0, $14
eret
```

五、测试/调试过程

（分别描述硬件模块的测试及调试实验程序的详细过程）

31 条**前仿真测试：**

数据来源于 mips246 网站上的 31 条 CPUtest 文件。

在 tb 中使用

$readmemh("L:\\instr.txt", CPU_inst.IMEM.memo); - - 初始化数据存储器

$fdisplay(file_output,"instr: %h",CPU_inst.instruction); -- 输出指令

$fdisplay(file_output,"pc: %h",CPU_inst.PC); -- 输出 pc

file_output=$fopen("L:\\result_my.txt"); - - 把结果输出到指定 txt 上

把输出结果与网站上的标准答案用 notepad++进行对比，发现 addu 指令部分结果与答案不同，查看 Mips 指令与 ALU 中寄存器的状态，发现是 0 号寄存器也发生改动导致错误，于是对所有写入寄存器堆的数据地址进行筛选，如果是要写入 0 号寄存器，则设置‘写’信号无效，保证 0 号寄存器不会发生改变。

31 条后仿真测试：

在前仿真完成之后，由于在下板的时候 tb 中的 initial 指令是无法综合的，因此需要把指令存储器更换成 ip 核。根据教程配置了 block memory generator，选择使用 COE 文件。然后进行后仿真功能测试，发现指令存储器存在两个周期的读延迟，打开 bmg ip 核的 summary 进行查看，果然存在延迟，于是在配置中去掉输出寄存器，依旧存在一个周期的读延迟。对于不涉及跳转的指令，延迟并不会造成什么大的影响，但是对于跳转指令，都要以当前 PC 为跳转基准，因此，读延迟会造成严重的错误。为了解决这个问题，就更换 ip 核类型，换成 distributed memory generator，选择 ROM,使用 COE 文件，直接生成。紧接着继续进行测试，发现读延迟消失了，指令的功能后仿真正常通过。最后再进行指令的后仿真时序分析，发现延迟很严重，最坏情况的延迟是 inf，后来一查，原来代表的是 infinite，吓的打开具体的时序分析报告，仔细查看延迟严重的部分在什么地方。报告中说，数据存储器中的延迟十分严重，于是把自己写的 dmem 换成 distributed memory generator 类型的 ip 核。然后再进行时序仿真，延迟降低到正常范围。然后可以开始进行下板测试了。

31 条下板测试：

下板后进行教程中的操作后，发现只有数码管的最后一位数字发生改变，于是回去查看.v 文件。发现是在顶层模块中从 CPU 中输出地 32 位的 wdata 在顶层模块中接收端口的宽度被设置成 1 位，进行修改后重新下板，进行相应操作后得到了预期的结果。

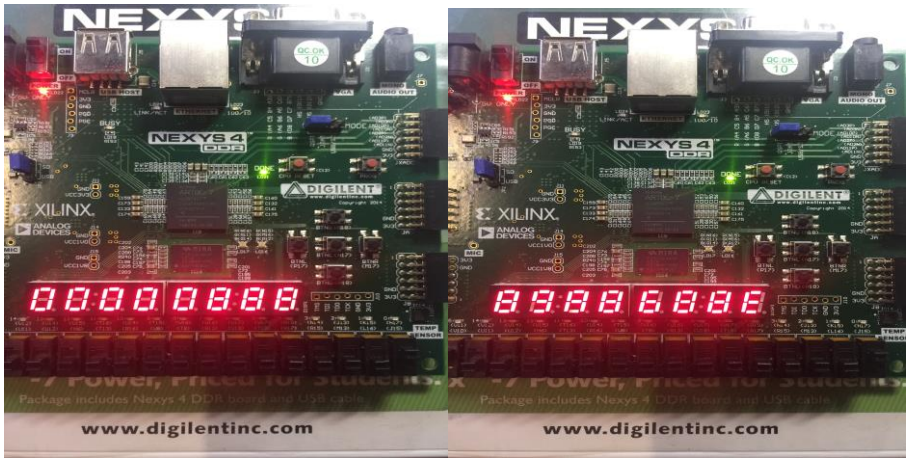
54 条下板测试：

下板后发现数码管上的数字卡在 0-7，断定这是 31 条出错，于是更换 31 条指令的 COE 文件，果然无法正常工作。于是对 CPU 进行前仿真，发现 LW/SW 指令通路在 54 条的时候被修改过，于是进行更正。接着更换 COE 为 54 条指令下板，数码管卡在 0-A，断定这是 lb/sb 出现问题。于是开始查询 lb/sb 的 verilog 代码，并没有发现什么大的问题。因此，去查询关于 lb/sb 的资料，发现对于指令的理解上出现问题。lb 指令中的地址最后两位会被忽略，要单独把最后两位取出进行独立的判断才行。于是继续修改 lb/sb 代码，继续下板验证，成功得到正确结果。

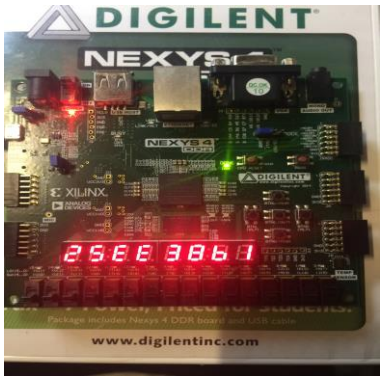
六、实验结果分析

（该部分可截图说明）

输入为 1，打开开关，过程中拍摄两张图

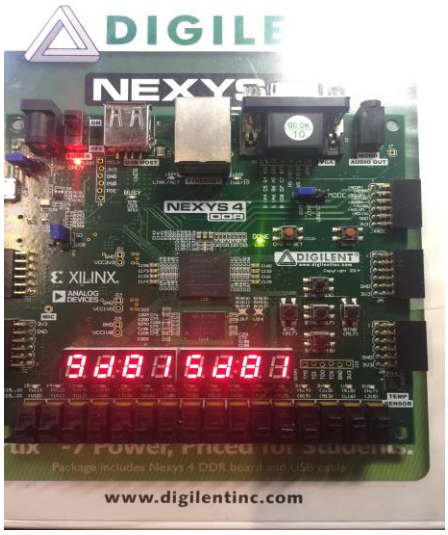


关闭开关，拍照结果



输入为 11，拍摄结果

输入为 111，拍摄结果


DIGILE
NEXYS
NEXYS 4
DDR
XILINX
DIGILENT
www.digilentinc.com

小程序演示图片：


PHONE
AV IN
MODE
V1/V2
–
M
+
POWER


PHONE
AV IN
MODE
V1/V2
–
M
+
POWER

七、结论

本次实验是使用 verilog 语言实现一个简单 MIPS 架构的 54 条指令单周期 CPU。在实验指导书的指导下，完成 CPU 的数据通路的构建，协调 CPU 各个部件之间的顺序和连接关系。

八、心得体会及建议

CPU 实验前前后后大概写了有一个月多的时间，期间查看了不少相关资料来理解 MIPS54 条指令中的每一条指令，通过对于指令的介绍来构建指令的数据通路以及控制信号。先从 8 条指令写起，然后写到 31 条，加入 CP0 协处理器，最后完成 54 条指令 CPU。整个过程层层递进，环环相扣。先进行前仿真验证，修改程序，接着再进一步通过后仿真看时序上的延迟问题去优化 CPU 的各个数据通路，最后再下板验证 CPU 的准确性以及性能。每一次下板验证都要等待很久的时间，因此尽量在前仿真的时候就把问题解决好，不然会很消耗时间。通过本次实验，充分理解了 CPU 的工作原理，锻炼了文献查询和阅读的能力。