计算机组成原理课程设计

题　目　　五级流水线处理器设计

学生姓名 _____

学　　号 _____

学　　院　　计算机科学与技术学院

专　　业　　计算机科学与技术

班　　级 _____

指导教师 _____

二〇一七年六月

目录

1．课程设计说明

1.1　指令实现

实现指令如下：

addu, addi, subu, slt, and, nor, or, xor, sll, srl, addiu, beq, bne, lw, sw, lui, j, sltu, jalr, jr, rs, sllv, sra, srav, srlv, slti, sltiu, bgez, bgtz, blez, bltz, lb, lbu, sb, andi, ori, xori, jal, mflo, mfhi, mtlo, mthi, mult

共 42 条指令

1.2　完成情况

1　处理了绝大多数可能的数据冒险，解决了分支指令和跳转指令所引起的控制冒险，这些采用了阻塞，转发技术实现，完美运行老师所给的测试指令。

2　采用五级流水线设计

3　可以在龙芯的开发板上进行测试

1.3　流水线设计说明

处理了一条指令的执行过程被分成五个阶段，每个阶段由相应的功能部件完成。如果将各阶段看成相应的流水段，则指令的执行过程就构成了一条指令流水线。

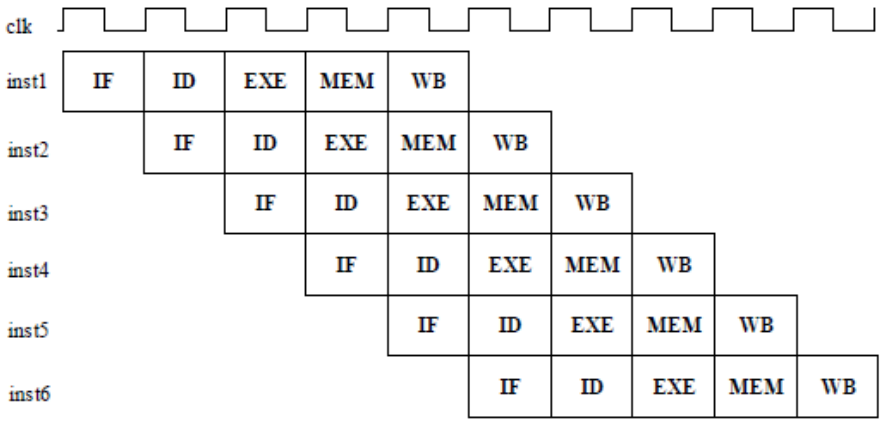
（1）Ifetch (取指)：取指令并计算 PC+4 。

（2）Reg/Dec (取数和译码)：取数同时译码 。

（3）Exec (执行)：计算内存单元地址 。

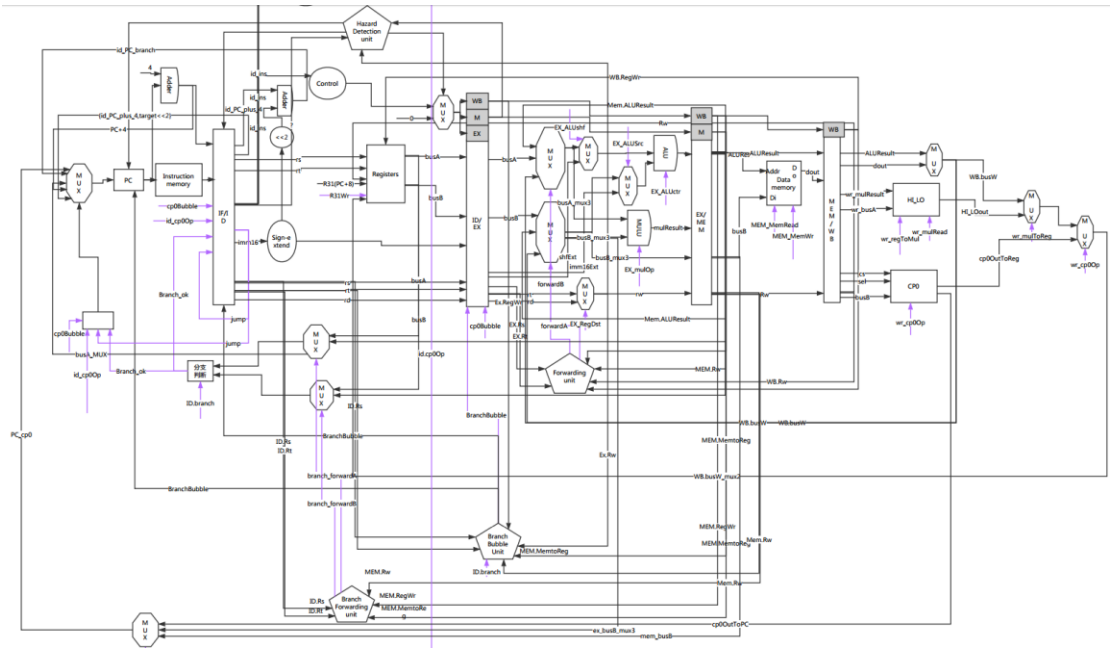
（4）Mem (读存储器)：从数据存储器中读 。

（5）Wr(写寄存器)：将数据写到寄存器中。

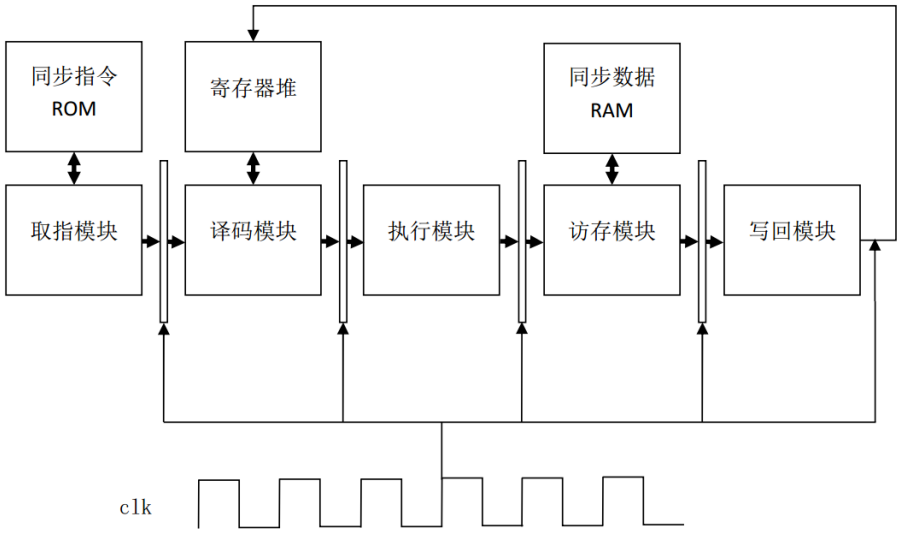


2．模块化和层次化设计说明

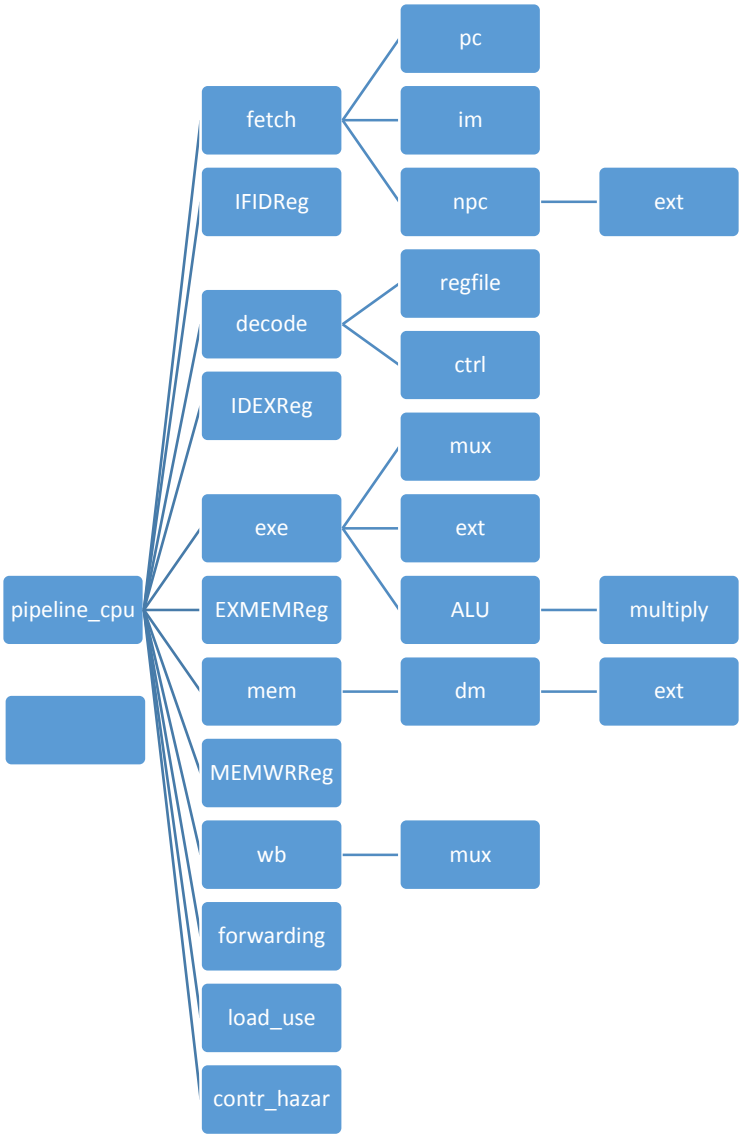
3.1　总电路设计图



3.2　模块化设计图



3.3　层次化设计



3.4 设计说明

电路设计使用模块化和层次化设计，根据五级流水线处理器的特点，为每一个阶段都写了一个模块，分别是 fetch, decode, exe, mem, wb。为每个流水段寄存器也写了一个模块，分别是 iFIDReg, IDEXReg, EXMEMReg, MEMWRReg，同时为了处理转发，数据冒险，控制冒险，将各个部件整合到了三个模块，分别是 forwarding, load_use, contr_hazar.以上就是 cpu 主要的大模块，把他们连接起来就是 pipeline_cpu。同时还有一些小部件，他们都整合到主要模块中。

3．具体模块化定义

3.1 fetch

1 模块介绍

该模块运行流水线的 ifetch 阶段，主要功能是输出当前指令的地址和获取下一条指令的地址

2 功能和输入输出

接受信号给子模块使用，同时把结果输出给段寄存器。

a) Input：clk, bubble, resetn, Zero, Jump, Branch_beq, Branch_bne, Bgez, Bgtz, Blez, Bltz, zBgez, zBgtz, Jalr, Jal, pre_PC, imm16, target, bus_rs
b) Output: inst, PC

3 嵌套模块介绍

（1） PC

a) 基本描述描述：
用来输出当前指令的地址
b) 接口定义：

Input	简介
NPC	下条指令的地址
clk	时钟信号
bubble	阻塞信号
rst	复位信号
Output	简介
PC	当前指令的地址

c) 功能定义：

条件	功能
clk 上升沿且 rst=0	复位，PC=0
clk 上升沿且 Bubble!=0	阻塞，PC 值不变
一般情况	PC=NPC

（2） NPC

a) 基本描述：
计算下一条指令并输出
b) 接口定义：

Input	简介
PC	当前指令的地址
pre_PC	计算分支，跳转指令时用到的地址
Target	跳转指令用来计算下条地址
Imm16	分支指令用来计算下条地址
bus_rs	jr,jalr 指令时用来跳转的地址
Branch_beq	beq 指令

Branch_bne	bne 指令
Bgez	bgez 指令
Bgtz	bgtz 指令
Blez	blez 指令
Bltz	bltz 指令
zBgez	判断 bgez 是否符合分支跳转条件
zBgtz	判断 bgtz 是否符合分支跳转条件
Jalr	jalr 和 jr 指令
Jal	jal 指令
Zero	判断 beq，bne 是否符合分支跳转条件
Jump	j 指令
Output	简介
NPC	下一跳指令地址

c) 功能定义

条件	功能
一般情况	NPC = NormalNPC
Jump = 1	输出 j 指令跳转地址 NPC = JumpNPC
Jal = 1	输出 jal 指令跳转地址 NPC = JumpNPC
Jalr = 1	输出 jalr 或 jr 指令跳转地址 NPC = bus_rs
Branch_beq=1 & Zero=1	输出 beq 指令分支跳转地址 NPC = BranchNPC
Branch_bne=1&Zero=0	输出 bne 指令分支跳转地址 NPC = BranchNPC
Bgez=1&zBgez=1	输出 bgez 指令分支跳转地址 NPC = BranchNPC
Bltz=1&zBgez=0	输出 bltz 指令分支跳转地址 NPC = BranchNPC
Bgtz=1&zBgtz=1	输出 bgtz 指令分支跳转地址 NPC = BranchNPC
Blez=1&zBgtz=0	输出 blez 指令分支跳转地址 NPC = BranchNPC

（3） im

a) 基本描述：
储存指令，根据输入地址输出指令

b) 接口定义：

Input	简介
addr	指令地址
Output	简介
dout	指令

c) 功能定义：

条件	功能
初始化	从文件读取指令
一般情况	根据所给地址输出指令

3.2 IFIDReg

1 模块介绍

该模块是一个流水段寄存器，时钟信号上升沿时把信号传给下一流水段或寄存器

2 功能和输入输出

接受信号，时钟信号上升沿时把信号传给下一流水段或寄存器。若 bubble 信号不为 0，说明有阻塞，不会把信号传给下一流水段或寄存器

a) Input：clk, bubble, PC_in, inst_in
b) Output: inst, PC

3.3 decode

1 模块介绍

该模块运行流水线的 Reg/Dec 阶段，主要功能是取数和译码

2 功能和输入输出

接受信号给子模块使用，同时把结果输出给段寄存器。本身具有分解指令的功能，把指令分为 op, rs, rt, rd, shamt, func, offset, imm16, target 给子模块使用或输出给后面阶段使用。

a) Input：clk, inst, busW, rw, rf_addr, RegWr_in, mtlo_in, mthi_in, mult_in, mult_result_in,
b) Output: imm16, busA, busB , rs , rt , rd , shf , target ,Branch_beq1, Branch_bne1, Jump , RegDst , ALUSrc , MemtoReg , RegWr , MemWr , ExtOp , ALUCtr ,MemRead , Bgez , Bgtz , Blez , Bltz , Jalr , B , LB , Jal , mflo , mfhi , mtlo , mthi , mult , hi, lo, rf_data

3 嵌套模块介绍

（1）regfile

a) 基本描述描述：
寄存器组，可以进行按字节读写，也可以按字读写。
b) 接口定义：

Input	简介
Rs	寄存器地址
Rt	寄存器地址
Rw	寄存器地址

test_addr	要观察的寄存器的地址
bus_W	要写的 32 位数据
clk	时钟信号
RegWr	写使能
mflo	读 lo 寄存器的值
mfhi	读 hi 寄存器的值
mtlo	写 lo 寄存器
mthi	写 hi 寄存器
mult	写乘法运算结果
mult_result	乘法运算结果
Output	简介
busA	输出地址为 rs 的寄存器的值
busB	输出地址为 rt 的寄存器的值
hi_out	输出 hi 寄存器的值
lo_out	输出 lo 寄存器的值
test_data	输出地址为 test_addr 的寄存器的值

c) 功能定义：

条件	功能
一般情况	输出地址为 test_addr，rt 的寄存器的值, 输出 hi,lo 寄存器的值
时钟信号下降沿到来且 RegWr=1，	写值到地址为 rw 的寄存器
时钟信号下降沿到来且 mthi=1	写 lo 寄存器
时钟信号下降沿到来且 mtlo=1	写 hi 寄存器
时钟信号下降沿到来且 mult=1	写乘法结果到 hi,lo 寄存器

（2）ctrl

a) 基本描述：

根据 op 和 func 输出译码信号

b) 接口定义：

Input	简介
op	操作码
func	func
rt	区分 bltz 和 bgez 指令
Output	简介
Jump	j 指令
Branch_beq	beq 指令
Branch_bne	bne 指令
Bgez	bgez 指令
Bgtz	bgtz 指令
Blez	blez 指令
Bltz	bltz 指令

Jalr	jalr 和 jr 指令
Jal	jal 指令
RegDst	选择目的寄存器信号
ALUSrc	选择操作数信号
MemtoReg	选择写入寄存器的信号
RegWr	寄存器写使能
MemWr	主存写使能
ExtOp	是否符号扩展信号
MemRead	lw 指令
B	按字节读写主存信号
LB	按字节读主存是否符号拓展信号
mfhi	mfhi 指令
mflo	mflo 指令
mtlo	mtlo 指令
mthi	mthi 指令
mult	mult 指令
ALUCtr	给 ALU 的操作信号

c) 功能定义

条件	功能
一般情况	根据 op 和 func 输出译码信号

3.4 IDEXReg

1 模块介绍

该模块是一个流水段寄存器，储存控制信号和数据，时钟信号上升沿时把信号，数据传给下一流水段或寄存器

2 功能和输入输出

接受信号，时钟信号上升沿时把信号传给下一流水段或寄存器。若 bubble 信号不为 0，说明有阻塞，不会把信号传给下一流水段或寄存器。同时，因为 Jalr，jr 需要用到 rs 寄存器的值，在这里处理了 jalr，jr 指令的数据冒险。同时处理了 jal 指令的冒险。

a) Input：clk, bubble, PC_in, target_in, imm16_in, busA_in, busB_in, rs_in, rt_in,

省略

mult_in,

b) Output: Branch_beq, Branch_bne, Jump , RegDst , ALUSrc , MemtoReg ,

省略

3.5 exe

1 模块介绍

该模块运行流水线的 Exec 阶段，主要功能是计算地址，结果

2 功能和输入输出

接受信号给子模块使用，同时把结果输出给段寄存器。把后几个阶段的数据和结果传进来，由 forwarding 模块提供转发信号，选择要进入 ALU 操作的数据，处理控制冒险。此阶段也选择 l 型指令和 r 型指令的目的寄存器

a) Input：clk, imm16_in, busA_in, busB_in, rt_in, rd_in, target_in,

省略

MemWr_in, ExtOp_in, mult_in, ALUCtr_in

b) Output:: MemWr , Jump , MemtoReg , RegWr , Zero , Overflow , rw , Result , target , imm16 , busB , zBgez , zBgtz , mult_result

3 嵌套模块介绍

（1）mux

a) 基本描述描述：

根据输入信号选择输出，有二路选择器，三路选择器，四路选择器三种，由于原理一样，此处介绍其中一种，三路选择器

b) 接口定义：

Input	简介
s	选择信号
a	待选数据 a
b	待选信号 b
c	待选信号 c
Output	简介
y	根据 s 输出

c) 功能定义：

条件	功能
s = 0	y=a, 输出 a
s = 1	y=b, 输出 b
s = 2	y=c, 输出 c

（2）ALU

a) 基本描述：

运算逻辑部件，在这里进行各种数值运算，

b) 接口定义：

Input	简介
ALUctr	ALU 控制信号
A	操作数 A
B	操作数 B
B_PC	用来计算分支/跳转指令地址
shif	移位

mult_in	乘法控制信号
Output	简介
Result	计算结果
Zero	判断是否符合 beq, bne 跳转条件
Bgez	判断是否符合 bgez 跳转条件
Bgtz	判断是否符合 bgtz 跳转条件
mult_result	乘法计算结果

c) 功能定义

条件	功能
ALUctr = 0	result = A + B
ALUctr = 1	result = A - B
ALUctr = 2	result = A < B 有符号比较
ALUctr = 3	result = A&
ALUctr = 4	result = ~(A | B)
ALUctr = 5	result = A | B
ALUctr = 6	result = A ^ B
ALUctr = 7	result = B << shif 逻辑左移
ALUctr = 8	result = B >> shif 逻辑右移
ALUctr = 9	result = {B[15:0], 16'b0 } 高位加载
ALUctr = 10	result = A < B 无符号比较
ALUctr = 11	result = B<<A 算术左移
ALUctr = 12	result = B>>shif 算术右移
ALUctr = 13	result = B>>A 算数右移
ALUctr = 14	result = B>>A 逻辑右移
ALUctr = 15	result = B_PC+4 跳转指令地址+4
ALUctr = 16	result = A 为 jalr,jr 指令计算
mult_in = 1	计算 A*B，mult_result 输出
一般情况	输出控制信号 Zero，Bgez, Bgtz 待需要时使用

（3）ext

a) 基本描述：

根据输入信号，选择进行符号扩展或零扩展。还有一个符号扩展器，始终做符号扩展，原理跟扩展器的符号扩展是一样的，因此这里只说明扩展器。

b) 接口定义：

Input	简介
Extp	扩展器控制信号

in	需要扩展的值
Output	简介
out	输出结果

c) 功能定义：

Input	简介
ExtOp = 0	对 in 进行 0 扩展
ExtOp = 1	对 in 进行符号扩展

（4）multuply

a) 基本描述：

根据输入信号，数值，对两个数进行乘法运算，该模块嵌套在 ALU 模块中。

b) 接口定义：

Input	简介
mult	乘法器控制信号
mult_op1	操作数 1
mult_op2	操作数 2
Output	简介
mult_result	输出乘法结果

c) 功能定义：

Input	简介
mult = 1	进行乘法运算并输出

3.6 EXMEMReg

1 模块介绍

该模块是一个流水段寄存器，储存控制信号和数据，时钟信号上升沿时把信号，数据传给下一流水段或寄存器

2 功能和输入输出

接受信号，时钟信号上升沿时把信号传给下一流水段或寄存器。若 bubbles 信号不为 0，说明有阻塞，不会把信号传给下一流水段或寄存器。

a) Input：clk, MemWr_in, Branch_beqin, Branch_bnein, Jump_in, MemtoReg_in,
省略
mult_in,

b) Output: MemWr , Branch_beq, Branch_bne, Jump , MemtoReg , RegWr , Zero , Overflow , rw , pre_PC, Result , bus_rs, busB , target , imm16 , mult_result , Bgez , Bgtz , Blez , Bltz , zBgez , zBgtz , Jalr , B , LB , Jal , mtlo , mthi , mult,

3.7 mem

1 模块介绍

该模块运行流水线的 MEM 阶段，主要功能是从主存中读写数值

2 功能和输入输出

接受信号给子模块使用，同时把结果输出给段寄存器。

a) Input：clk, imm16_in, busA_in, busB_in, rt_in, rd_in, target_in,
省略

MemWr_in, ExtOp_in, mult_in, ALUCtr_in

b) Output:: MemWr , Jump , MemtoReg , RegWr , Zero , Overflow , rw , Result , target , imm16 , busB , zBgez , zBgtz , mult_result

3 嵌套模块介绍

（1）dm

a) 基本描述描述：

数据存储，指令读写内存中的数据都在这里进行。

b) 接口定义：

Input	简介
addr	要读写的地址
din	要写的数据 a
mem_addr	开发板要观察的储存地址
we	写使能
clk	时钟信号
ExtOp	字节读时判断进行符号扩展还是 0 扩展
B	判断是否进行字节的读写
Output	简介
dout	输出的数据
mem_data	输出值以便在开发板上观察

c) 功能定义：

条件	功能
B != 0	输出 addr 所对应的字
B != 0 & we=1	在 addr 所对应的存储单元写数据 din
B=1 & LB=1	将 addr 所对应的字节进行符号扩展，并输出
B=1 & LB=0	将 addr 所对应的字节进行 0 扩展，并输出
B=1 & we=1	在 addr 所对应的字节写数据 din

3.8 MEMWRReg

1 模块介绍

该模块是一个位于 MEM 和 WB 阶段间的流水段寄存器，储存控制信号和数据，时钟信号上升沿时把信号，数据传给下一流水段或寄存器

2 功能和输入输出

接受信号，时钟信号上升沿时把信号传给下一流水段或寄存器。刚开始会初始化

a) Input：clk, MemtoReg _in, RegWr _in, Dout _in, Result _in, PC_in, rw _in, 省略

b) Output: Dout , Result , PC , rw , Overflow , MemtoReg , RegWr , mtlo , mthi , mult , mult_result

3.9 wb

1 模块介绍

该模块运行流水段的写回阶段，来自 ALU 的结果和主存的结果经过选择，传到 decode 阶段的寄存器里面，顺便把写使能信号，乘法运算结果也传回去

2 功能和输入输出

接受信号给子模块使用，同时把结果输出给段寄存器。

a) Input：Dout, Result, MemtoReg。

b) Output: Din

3 嵌套模块设计

这里使用了一个二路选择器，当 MemtoReg = 0 时，输出 ALU 的结果 Result。当 MemtoReg=1 时，输出从主存取出的结果。

3.10 forwarding

1 模块介绍

该模块主要处理解决数据冒险的转发, 考虑了各种转发。

2 接口定义

Input	简介
rs	EXE 阶段的 rs
rt	EXE 阶段的 rt
rw_mem	MEM 阶段的 rw
rw_wr	WB 阶段的 rw
RegWr_mem	MEM 阶段的写使能
RegWr_wr	WB 阶段的写使能
ALUSrc	I 型指令选择立即数的信号
mflo	EXE 阶段的 mflo 信号
mffhi	EXE 阶段的 mfhi 信号

mtlo_mem	MEM 阶段的写使能
mthi_mem	MEM 阶段的写使能
mtlo_wr	MEM 阶段的写使能
mthi_wr	MEM 阶段的写使能
Output	简介
ALUSrc_A	选择 ALU 的操作数 A
ALUSrc_B	选择 ALU 的操作数 B

3 功能定义

条件	功能
ALUSrc_A = 0	ALU 操作数 A 选择来自 busA 的数据
ALUSrc_A = 1	ALU 操作数 A 选择来自 MEM 的 Result（上一条指令计算结果）
ALUSrc_A = 2	ALU 操作数 A 选择来自 WB 的 Result（上一两条指令计算结果）
ALUSrc_B = 0	ALU 操作数 B 选择来自 busB 的数据
ALUSrc_B = 1	ALU 操作数 B 选择来自 MEM 的 Result（上一条指令计算结果）
ALUSrc_B = 2	ALU 操作数 B 选择来自 WB 的 Result（上一条指令计算结果）
ALUSrc_B = 3	ALU 操作数 B 选择立即数
ALUSrc = 1	ALUSrc_B = 3，ALU 操作数 B 选择立即数

3.10 load_use

1 模块介绍

该模块主要处理 lw, lb, lbu 所引起的数据冒险，因为无法转发，所以采用阻塞的方法来处理。

2 接口定义

Input	简介
clk	时钟信号
MemRead	lw,lb,lbu 信号
rs_id	ID 阶段的 rs
rt_id	ID 阶段的 rt
rt_ex	EXE 阶段的 rt
Bgez	Bgez 指令
Bgtz	Bgtz 指令

Blez	Blez 指令
Bltz	Bltz 指令
zBgez	判断 Bgez 指令是否跳转
zBgtz	判断 Bgtz 指令是否跳转
Jalr	Jalr 指令
Jal	Jal 指令
Zero	判断 beq，bne 指令是否跳转
Jump	j 指令
Branch_beq	beq 指令
Branch_bne	bne 指令
Output	简介
bubble	气泡，若不为零，则收到这个气泡的部件都会阻塞

3 功能定义

条件	功能
发生 lw 冒险	bubble = 1，阻塞一次
发送 lb 冒险	bubble = 1，阻塞一次
发生 lbu 冒险	bubble = 1，阻塞一次
bubble = 1	bubble = 0，只阻塞一次
Jump = 1	上一条指令是 j 指令，需要跳转地址，则冒险不需要处理，bubble = 0。
Jal = 1	上一条指令是 jal 指令，需要跳转地址，则冒险不需要处理，bubble = 0。
Jalr = 1	上一条指令是 jalr 指令，需要跳转地址，则冒险不需要处理，bubble = 0。
Branch_beq=1 & Zero=1	上一条指令是 beq 指令，分支跳转条件成立，需要跳转地址，则冒险不需要处理，bubble = 1。
Branch_bne=1&Zero=0	上一条指令是 bne 指令，分支跳转条件成立，需要跳转地址，则冒险不需要处理，bubble = 1。
Bgez=1&zBgez=1	上一条指令是 Bgez 指令，分支跳转条件成立，需要跳转地址，则冒险不需要处理，bubble = 1。
Bltz=1&zBgez=0	上一条指令是 Bltz 指令，分支跳转条件成立，需要跳转地址，则冒险不需要处理，bubble = 1。
Bgtz=1&zBgtz=1	上一条指令是 Bgtz 指令，分支跳转条件成立，需要跳转地址，

	则冒险不需要处理，bubble = 1。
Blez=1&zBgtz=0	上一条指令 Blez 指令，分支跳转条件成立，需要跳转地址，则冒险不需要处理，bubble = 1。

3.11 contr_hazar

1 模块介绍

该模块主要处理跳转指令，分支指令产生的控制冒险。

2 接口定义

Input	简介
clk	时钟信号
Bgez	Bgez 指令
Bgtz	Bgtz 指令
Blez	Blez 指令
Bltz	Bltz 指令
zBgez	判断 Bgez 指令是否跳转
zBgtz	判断 Bgtz 指令是否跳转
Jalr	Jalr 指令
Jal	Jal 指令
Zero	判断 beq，bne 指令是否跳转
Jump	j 指令
Branch_beq	beq 指令
Branch_bne	bne 指令
Output	简介
bubble	气泡，若不为零，则收到这个气泡的部件都会阻塞

3 功能定义

条件	功能
bubble ！= 0	阻塞一次之后，气泡减一
Jump = 1	MEM 阶段指令是 j 指令，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Jal = 1	MEM 阶段指令是 jal 指令，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Jalr = 1	MEM 阶段指令是 jalr 指令，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Branch_beq=1 & Zero=1	MEM 阶段指令是 beq 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险

	bubble = 3。
Branch_bne=1&Zero=0	MEM 阶段指令是 bne 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Bgez=1&zBgez=1	MEM 阶段指令是 bgez 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Bltz=1&zBgez=0	MEM 阶段指令是 bltz 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Bgtz=1&zBgtz=1	MEM 阶段指令是 bgtz 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。
Blez=1&zBgtz=0	MEM 阶段指令是 blez 指令，分支跳转条件成立，需要跳转地址，采用阻塞解决控制冒险 bubble = 3。

3.12 pipeline_cpu

1 模块介绍

该模块把以上所介绍的主要部件连接起来，就是一个 mips 流水线处理器，从外围组件传进来时钟信号 clk 和复位信号 resetn，使流水线工作。同时把各阶段指令的 PC 值，寄存器里的值，要观察内存的内容，IF 阶段的指令展示在开发板上，我们可以观看

2 接口定义

Input	简介
clk	时钟信号
resetn	复位信号
rf_addr	要观察寄存器的地址
mem_addr	要观察主存的地址
Output	简介
rf_data	要观察寄存器的值
mem_data	要观察主存的值
IF_pc	IF 阶段的 PC
IF_inst	IF 阶段的指令
ID_pc	ID 阶段的 PC
EXE_pc	EXE 阶段的 PC
MEM_pc	MEM 阶段的 PC
WB_pc	WB 阶段的 PC

HI_data	HI 寄存器的数据
LO_data	LO 寄存器的数据

3 功能定义

条件	功能
一般情况	作为一个五级流水线处理器，发挥其处理指令的功能
一般情况	把各阶段的 PC 值传出去以供观察
一般情况	把个寄存器的值传出去以供观察
一般情况	把 IF 阶段的指令值传出去以供观察
一般情况	根据输入的主存地址，观察那一块地址的值

4．测试代码与结果

4.1 测试代码

（1） 测试代码使用老师给的测试代码，如下所示：

```
loop0:addiu $1, $0,1
      sll     $2, $1,4
      addu    $3, $2,$1
      srl      $4, $2,2
      slti     $25,$4,5
      bgez    $1,loop1
      subu    $5,$3,$4
      sw       $5, 20($0)
      nor      $6, $5,$2
      or       $7, $6,$3
      xor      $8, $7,$6
      sw       $8, 28($0)
      beq      $8, $3,loop2
      slt      $9, $6,$7
loop2: addiu $1, $0,8
      lw       $10,20($1)
      bne      $10,$5,loop3
      and      $11,$2,$1
```

```
        sw      $11,28($1)
        sw      $4, 16($1)
    loop3: jal    loop4
    loop1: lui      $12,12
        srav    $26,$12,$2
        sllv    $27,$26,$1
        jalr    $27
    loop4:  sb   $26,5($3)
        sltu    $13,$3,$3
        bgtz    $13,loop4
        sllv    $14,$6,$4
        sra     $15,$14,2
        srlv    $16,$15,$1
        blez    $16,loop1
        srav    $16,$15,$1
        addiu $11,$0,140
        bltz $16, loop6
        lw      $28,3($10)
        bne     $28,$29,loop7
    loop8:sb      $15,8($5)
        lb      $18,8($5)
        lbu     $19,8($5)
    loop6: sltiu $24,$15,-1
        or      $29,$12,$5
        jr      $11
    loop7: andi $20,$15,-1
        ori     $21,$15,-1
        xori $22,$15,-1
j     loop0
```

更详细请看附录，或者查看文档：测试代码.docx

（2）用 mars 将汇编指令生成机器码，但是后面四条指令会出现问题，如图

0x000000ac	0x3c01ffff	lui $1,0xffffffff	44: loop7: andi $20,$15,-1
0x000000b0	0x3421ffff	ori $1,$1,0x0000ffff	
0x000000b4	0x01e1a024	and $20,$15,$1	
0x000000b8	0x3c01ffff	lui $1,0xffffffff	45: ori $21,$15,-1
0x000000bc	0x3421ffff	ori $1,$1,0x0000ffff	
0x000000c0	0x01e1a825	or $21,$15,$1	
0x000000c4	0x3c01ffff	lui $1,0xffffffff	46: xori $22,$15,-1
0x000000c8	0x3421ffff	ori $1,$1,0x0000ffff	
0x000000cc	0x01e1b026	xor $22,$15,$1	
0x000000d0	0x08000000	j 0x00000000	47: j loop0

mars 不能将后面四条指令直接翻译成机器码，而是通过另外的方式实现，但这会破坏寄存器 1 的值，所以后四条指令采用手工翻译，所得到的机器码如下：

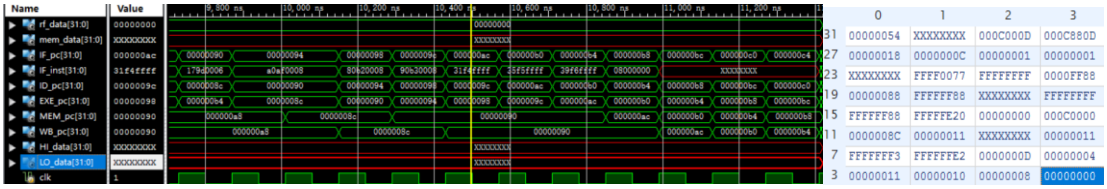
```
24010001
00011100
00411821
```

00022082
28990005
0421000f
00642823
ac050014
00a23027
00c33825
00e64026
ac08001c
11030001
00c7482a
24010008
8c2a0014
15450003
00415824
ac2b001c
ac240010
0c000019
3c0c000c
004cd007
003ad804
0360f809
a07a0005
0063682b
1da0fffd
00867004
000e7883
002f8006
1a00fff5
002f8007
240b008c
06000005
8d5c0003
179d0006
a0af0008
80b20008
90b30008
2df8ffff
0185e825
01600008
31f4ffff
35f5ffff
39f6ffff
08000000

4.2 测试仿真结果

用 ISE 的 ISim 进行仿真，仿真用的文件是 testbench，可以看到各个阶段的 PC 值和 IF 阶段的指令。因为要在开发板上测试，而由于开发板的原因，第一条指令没办法读出来，所以测试时将 PC 的初值赋值为-4，这样相当于在第一条指令前加了空指令，对后面的指令没有影响。也方便了板上的测试结果与仿真结果比较。部分仿真结果如下：

（仿真信号）　　（寄存器的值）

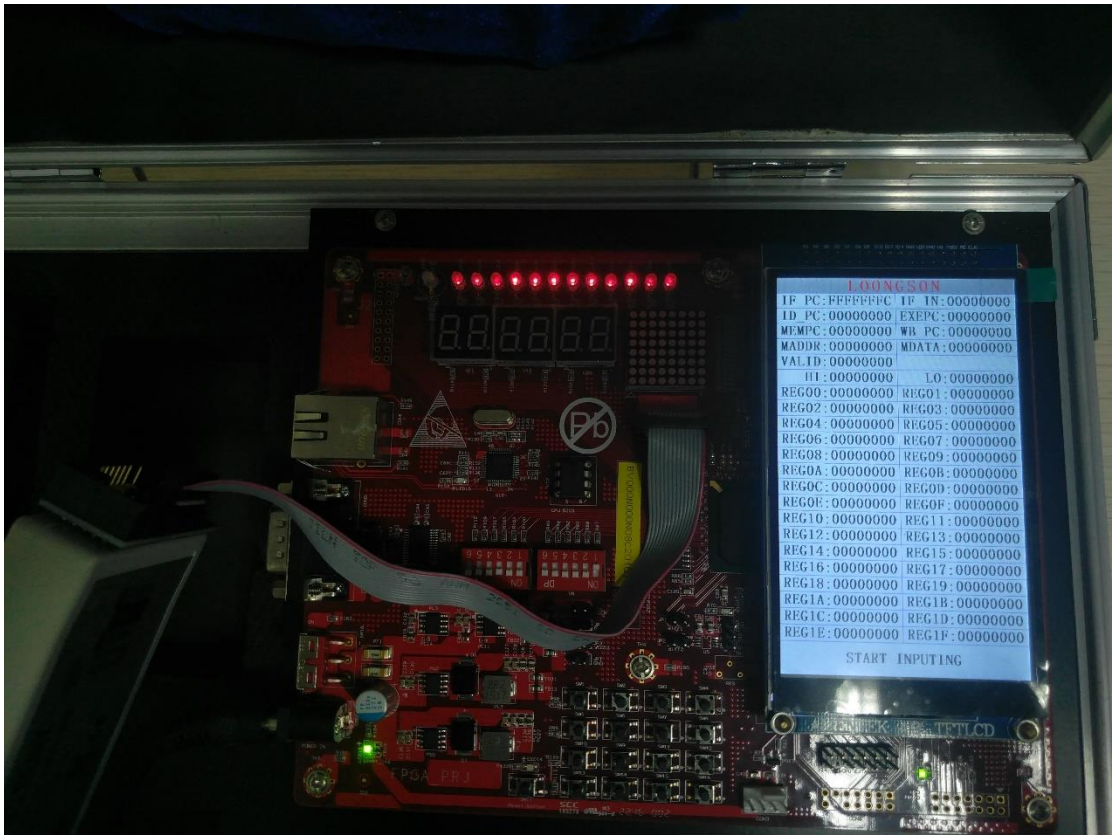


处理器正确指令测试代码，结果跟附录所示结果一样，测试成功

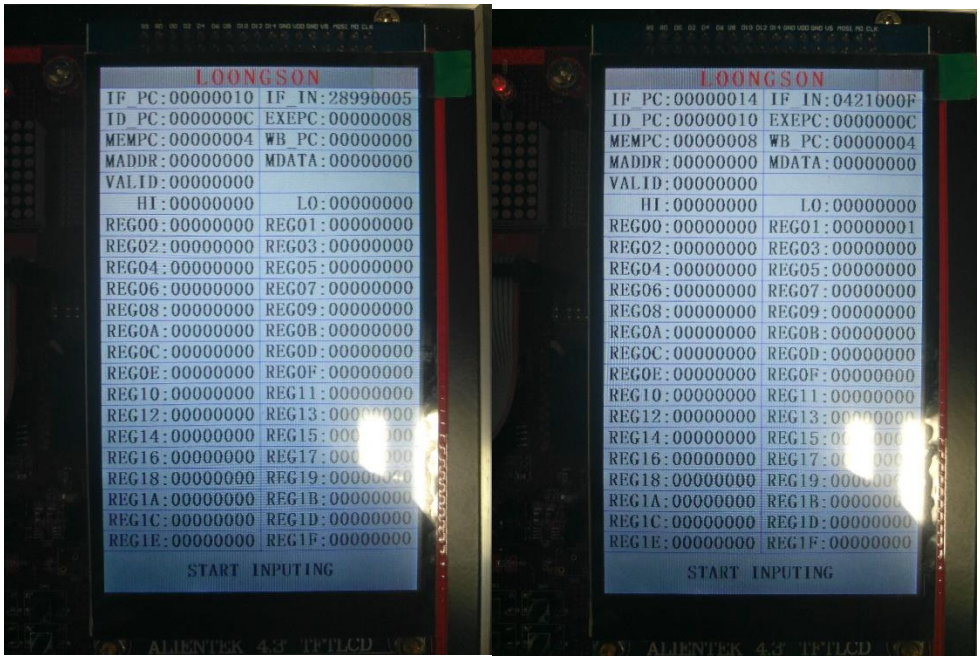
4.3 上板测试结果

用老师给的外围模块，使 pipeline_cpu 嵌套进去，可以在板子上看到仿真测试的结果。首先要先代码编译，生成比特流文件，再把比特流文件烧到板子上，就可以观察测试结果了。

有一点要注意，开发板不能读第一条指令，因此生成比特流文件之前需要将 PC 初始地址初始化为-4，这样相当于在第一条指令前插入了空指令，不影响指令的运行。部分结果如下：

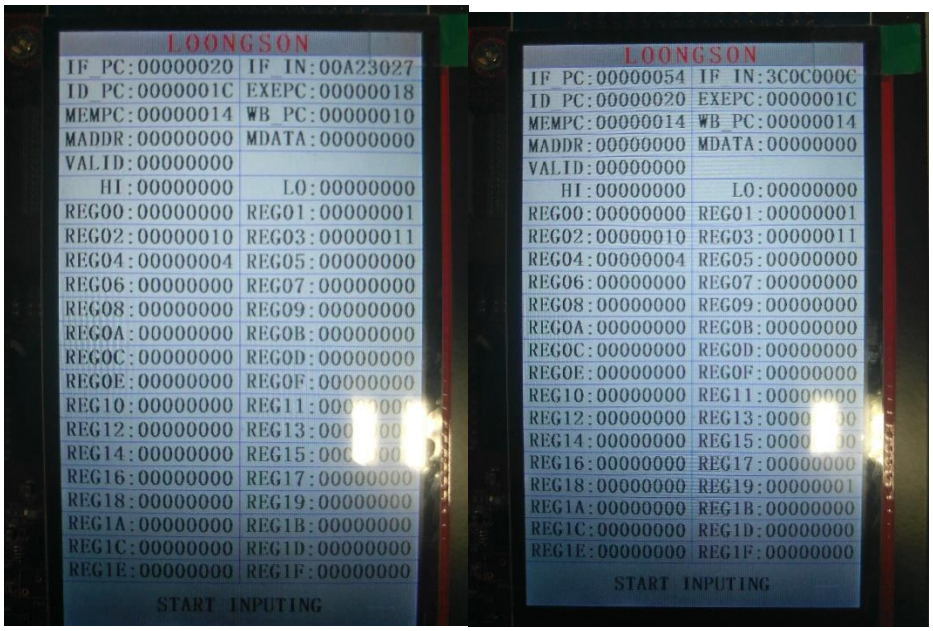


因为初始化地址是-4，所以一开始 IF_PC 显示-4 的补码 FFFFFFFCH，这很正常。



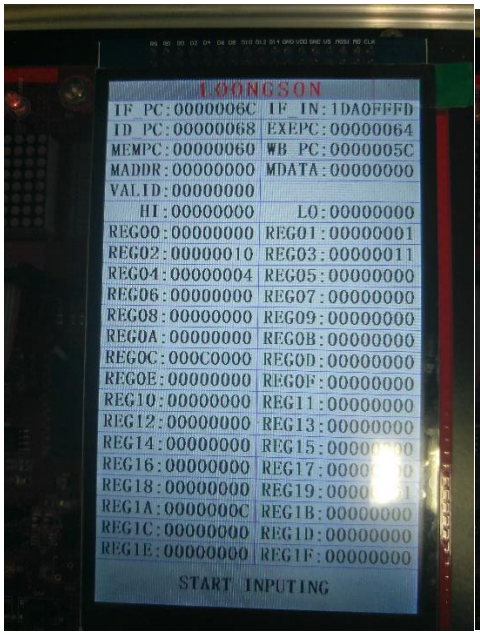
第一张图 MEM_PC = 4H,WB_PC = 0H,说明第一条指令在写回阶段

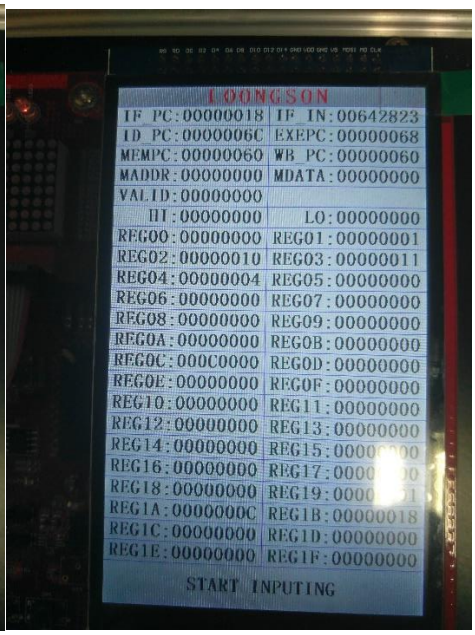
第二张图 WB_PC = 4H，第一条指令已经写完，寄存器 1 的值为 1，写值成功



图一 MEMPC 为 14H，是 bgez 指令，跳转条件成立，要跳转到指令地址 54H

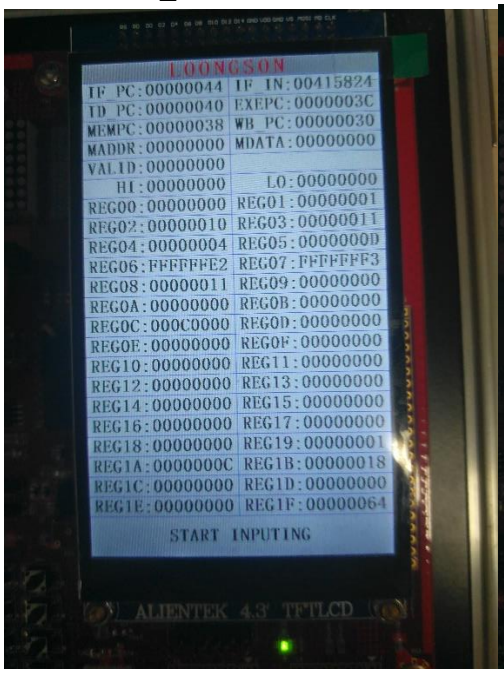
图二 IF_PC 为 54H，跳转成功

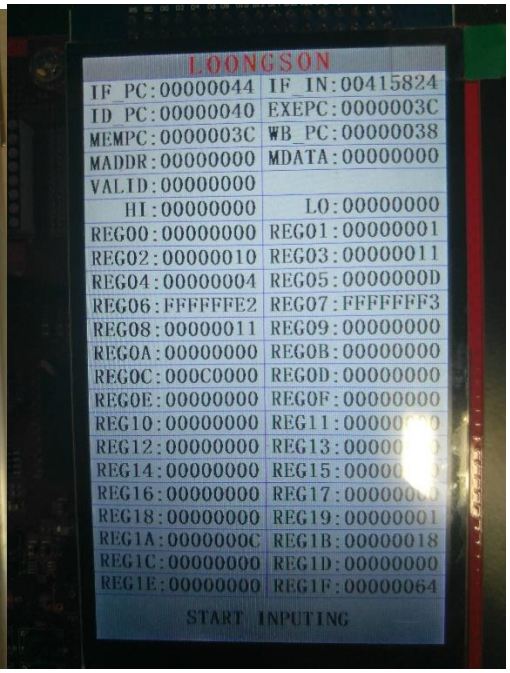


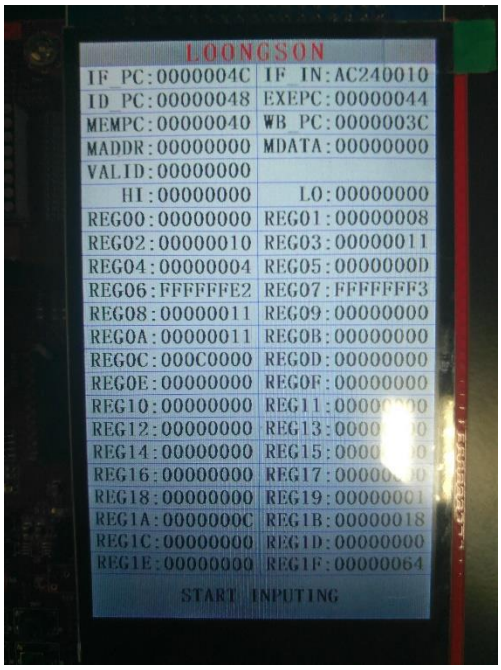


图一 MEMPC 为 60H，是 jalr 指令，跳转地址为 REG1BH（寄存器 27）的值，即要跳转到地址 18H

图二 IF_PC 为 18H，跳转成功



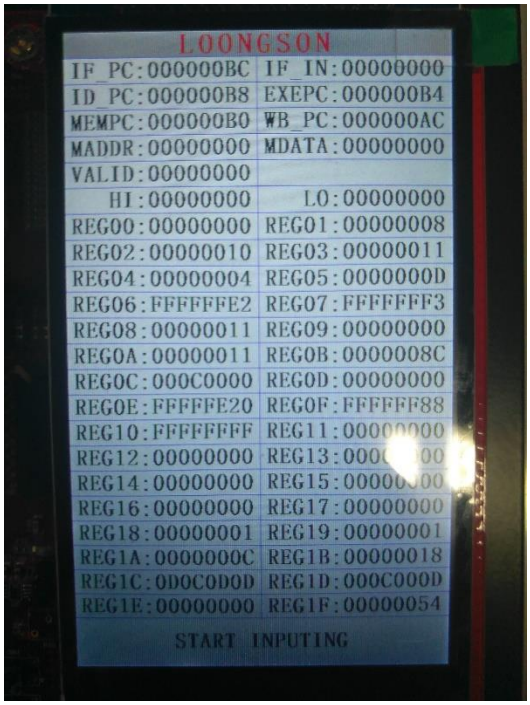


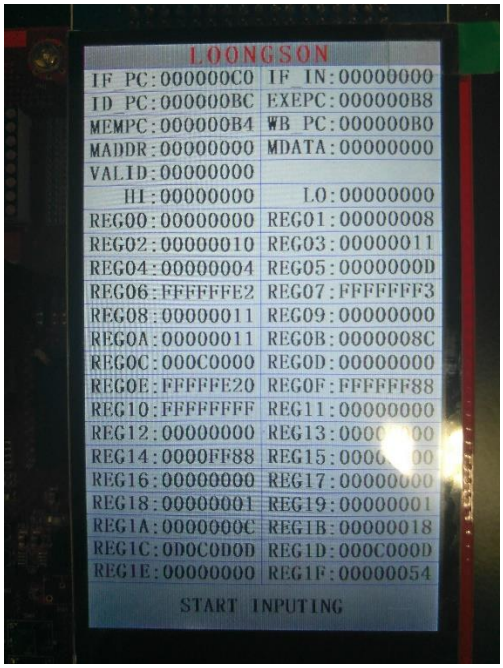


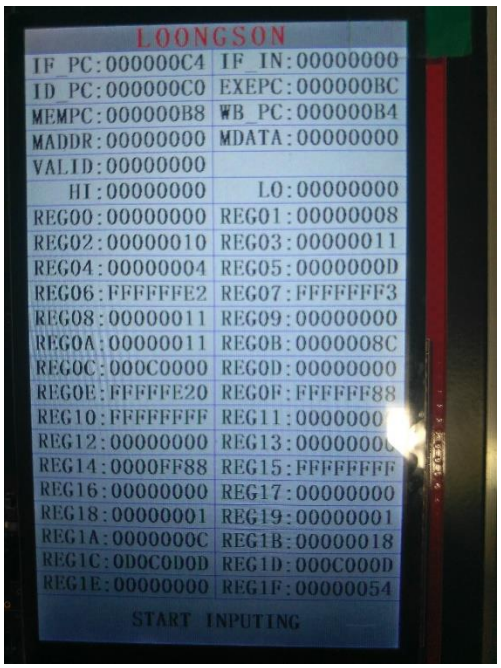
图一 EXEPC= 3CH，是 lw 指令，因为有数据冒险，要用阻塞解决

图二 IF_PC，IF_IN， ID_PC，EXEPC 的值跟图一相比没有改变，阻塞成功，lw 指令在访存阶段

图三 EXEPC = 44H，说明跟图二相比进行了两个周期，lw 指令成功把值写到 REG0AH 里。







图一 WB_PC = ACH，说明是 andi 指令（andi $20,$15,#0xFFFF）

图二是图一执行一个阶段后的图，REG14H 写值成功，正确，指令正确指令。此时 WB_PC = B0H，是 ori 指令（ori $21,$15,#0xFFFF）

图三是图二执行一个阶段后的图，REG15H 写值成功，正确，指令正确指令

总之，开发板观测到指令全部正确执行，测试成功

5．心得体会

写到这里的时候，这场艰辛，历时两周的课程设计终于完成了。回想起这十几天日日夜夜废寝忘食地做课设，同时还顶着期末考试的压力，不断提高的要求和老师对我们恨铁不成钢(？没有的事)的期望，真是让我感慨万分。

在我们完成计算机组成原理的实验之后，我们马不停蹄就开始了做课程设计。一开始我没有马上开始敲代码，我花了一天的时间熟悉课本上有关流水线处理器的知识和思考该怎么设计电路图。之后，就决定按照流水线处理器的原理来设计，它分阶段，每个阶段中间还有个流水段寄存器，那就把他们分模块来写就好了。

我设计思路是先设计九条带冒险处理的流水线处理器，因为 9 条指令跟 36 条指令的总体结构是差不多的。之后，继续在这个基础上一步步添加指令，处理冒险。然后，我花了一天多的时间，终于把九条带冒险处理的流水线处理器做好了。这期间最难的不是写代码，是修改 bug，因为用 modelsim 写代码，编译的时候它只提示了一点点语法错误，有很多其他错误它没提示！结果我大部分时间都是在调代码。之后要花时间准备期末考试，就先把课设放一边去了。

上午考完试之后，为了赶时间下午就开始课设了。可没想到的是，老师对我们的勤奋优异的表现很满意，将标准提高了，要做 45 条指令。于是我只能没日没夜的写代码，调试代码。这期间遇到了很多问题。45 条指令跟 9 条指令整体架构上的差别还是有点大的，主要是因为多了个协处理器和 hi，lo 这两个特殊的寄存器。因为我不懂协处理器是用来干什么的，所以我问了一些同学，结果他们也不知道，而且大多人没写协处理器，我就先放一边了。结果到后面又要调试，又要准备上开发板，还要花时间写报告，暑期实训又要开始了。实在赶不过来了，就不做后面几条有关协处理器的指令，最终完成了 42 条指令，看起来是个吉利的数字

实验做完了，总算可以松一口气。这次课设我收获了很多，不实用的 modelsim 大大提高了我改 bug 的能力，而且温故知新，我对流水线处理器有了更深入的了解。因为有一些人问我问题，我也提高了我的表达能力，也加深了我与同学间的联系。总之，这次课设受益匪浅，做出来个 CPU 让我很有满足感，自豪。这次经历对我来说将会是宝贵的财富。

测试代码解析

指令地址	汇编指令	结果描述	机器指令的机器码	
			16 进制	二进制
00H	addiu $1, $0,#1	[$1] = 0000_0001H	24010001	0010_0100_0000_0001_0000_0000_0000_0001
04H	sll $2, $1,#4	[$2] = 0000_0010H	00011100	0000_0000_0000_0001_0001_0001_0000_0000
08H	addu $3, $2,$1	[$3] = 0000_0011H	00411821	0000_0000_0100_0001_0001_1000_0010_0001
0CH	srl $4, $2,#2	[$4] = 0000_0004H	00022082	0000_0000_0000_0010_0010_0000_1000_0010
10H	slti $25,$4,#5	[$25] = 0000_0001H	28990005	0010_1000_1001_1001_0000_0000_0000_0101
14H	bgez $25,#16	跳转到 54H	07210010	0000_0111_0010_0001_0000_0000_0001_0000
18H	subu $5, $3,$4	[$5] = 0000_000DH	00642823	0000_0000_0110_0100_0010_1000_0010_0011
1CH	sw $5, #20($0)	Mem[0000_0014H] = 0000_000DH	AC050014	1010_1100_0000_0101_0000_0000_0001_0100
20H	nor $6, $5,$2	[$6] = FFFF_FFE2H	00A23027	0000_0000_1010_0010_0011_0000_0010_0111
24H	or $7, $6,$3	[$7] = FFFF_FFF3H	00C33825	0000_0000_1100_0011_0011_1000_0010_0101
28H	xor $8, $7,$6	[$8] = 0000_0011H	00E64026	0000_0000_1110_0110_0100_0000_0010_0110
2CH	sw $8, #28($0)	Mem[0000_001CH] = 0000_0011H	AC08001C	1010_1100_0000_1000_0000_0000_0001_1100
30H	beq $8, $3,#2	跳转到 38H	11030002	0001_0001_0000_0011_0000_0000_0000_0010
34H	slt $9, $6,$7	不执行	00C7482A	0000_0000_1100_0111_0100_1000_0010_1010
38H	addiu $1, $0,#8	[$1] = 0000_0008H	24010008	0010_0100_0000_0001_0000_0000_0000_1000
3CH	lw $10,#20($1)	[$10] = 0000_0011H	8C2A0014	1000_1100_0010_1010_0000_0000_0001_0100
40H	bne $10,$5,#4	跳转到 50H	15450004	0001_0101_0100_0101_0000_0000_0000_0100
44H	and $11,$2,$1	不执行	00415824	0000_0000_0100_0001_0101_1000_0010_0100
48H	sw $11,#28($1)	不执行	AC2B001C	1010_1100_0010_1011_0000_0000_0001_1100
4CH	sw $4, #16($1)	不执行	AC240010	1010_1100_0010_0100_0000_0000_0001_0000
50H	jal #25	跳转到 64H, [$31] = 0000_0054H	0C000019	0000_1100_0000_0000_0000_0000_0001_1001
54H	lui $12,#12	[$12] = 000C_0000H	3C0C000C	0011_1100_0000_1100_0000_0000_0000_1100
58H	srav $26,$12,$2	[$26] = 0000_000CH	004CD007	0000_0000_0100_1100_1101_0000_0000_0111
5CH	sllv $27,$26,$1	[$27] = 0000_0018H	003AD804	0000_0000_0011_1010_1101_1000_0000_0100
60H	jalr $27	跳转到 18H ,	0360F809	0000_0011_0110_0000_1111_1000_0000_1001

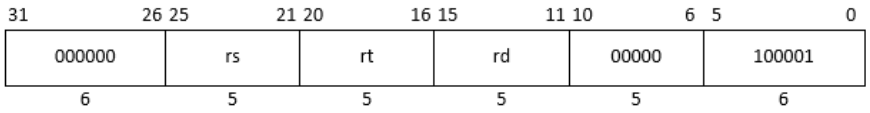
		[$31] = 0000_0064H		
64H	sb $26,#5($3)	MEM[0000_0016H] = 000C_000DH	A07A0005	1010_0000_0111_1010_0000_0000_0000_0101
68H	sltu $13,$3,$3	[$13] = 0000_0000H	0063682B	0000_0000_0110_0011_0110_1000_0010_1011
6CH	bgtz $13,#3	不跳转	1DA00003	0001_1101_1010_0000_0000_0000_0000_0011
70H	sllv $14,$6,$4	[$14] =FFFF_FE20H	00867004	0000_0000_1000_0110_0111_0000_0000_0100
74H	sra $15,$14,#2	[$15] =FFFF_FF88H	000E7883	0000_0000_0000_1110_0111_1000_1000_0011
78H	srlv $16,$15,$1	[$16] =00FF_FFFFH	002F8006	0000_0000_0010_1111_1000_0000_0000_0110
7CH	blez $16,#8	不跳转	1A000008	0001_1010_0000_0000_0000_0000_0000_1000
80H	srav $16,$15,$1	[$16] =FFFF_FFFFH	002F8007	0000_0000_0010_1111_1000_0000_0000_0111
84H	addiu $11,$0,#140	[$11] = 0000_008CH	240B008C	0010_0100_0000_1011_0000_0000_1000_1100
88H	bltz $16, #6	跳转到 A0H	06000006	0000_0110_0000_0000_0000_0000_0000_0110
8CH	lw $28,#3($10)	[$28] = 000C_000DH /000C_880DH	8D5C0003	1000_1101_0101_1100_0000_0000_0000_0011
90H	bne $28,$29,#7	不跳转/跳转 ACH	179D0007	0001_0111_1001_1101_0000_0000_0000_0111
94H	sb $15,#8($5)	Mem[0000_0015H] = 0000_0088H	A0AF0008	1010_0000_1010_1111_0000_0000_0000_1000
98H	lb $18,#8($5)	[$18] =FFFF_FF88H	80B20008	1000_0000_1011_0010_0000_0000_0000_1000
9CH	lbu $19,#8($5)	[$19] = 0000_0088H	90B30008	1001_0000_1011_0011_0000_0000_0000_1000
A0H	sltiu $24,$15,#0xFFFF	[$24] = 0000_0001H	2DF8FFFF	0010_1101_1111_1000_1111_1111_1111_1111
A4H	or $29,$12,$5	[$29] = 000C000DH	0185E825	0000_0001_1000_0101_1110_1000_0010_0101
A8H	jr $11	跳转指令 8CH	01600008	0000_0001_0110_0000_0000_0000_0000_1000
ACH	andi $20,$15,#0xFFFF	[$20] = 0000_FF88H	31F4FFFF	0011_0001_1111_0100_1111_1111_1111_1111
B0H	ori $21,$15,#0xFFFF	[$21] =FFFF_FFFFH	35F5FFFF	0011_0101_1111_0101_1111_1111_1111_1111
B4H	xori $22,$15,#0xFFFF	[$22] = FFFF_0077H	39F6FFFF	0011_1001_1111_0110_1111_1111_1111_1111
B8H	j #00H	跳转指令 00H	08000000	0000_1000_0000_0000_0000_0000_0000_0000

附录 2

实现指令详解

（1） 无符号加法

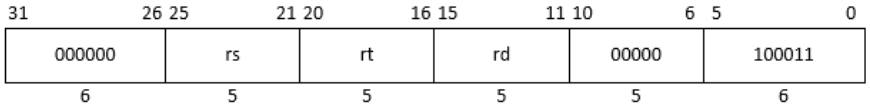
ADDU rd, rs, rt　　R 型



GPR[rd] ← GPR[rs] + GPR[rt]

（2） 无符号减法

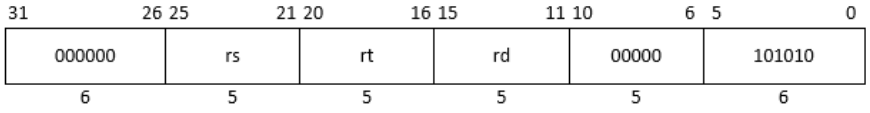
SUBU rd, rs, rt　　R 型



GPR[rd] ← GPR[rs] - GPR[rt]

（3） 有符号比较，小于置位

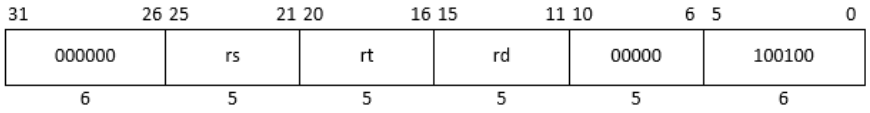
SLT rd, rs, rt　　R 型



GPR[rd] ← (sign(GPR[rs]) < sign(GPR[rt]))

（4） 按位与

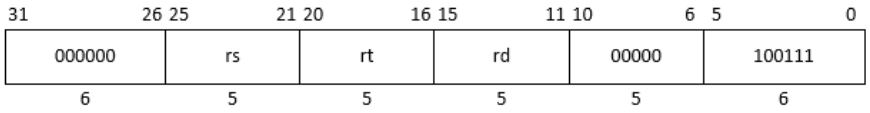
AND rd, rs, rt　　R 型



GPR[rd] ← GPR[rs] & GPR[rt]

（5） 按位或非

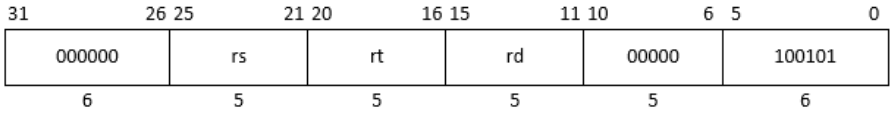
NOR rd, rs, rt　　R 型



GPR[rd] ← ~(GPR[rs] | GPR[rt])

（6） 按位或

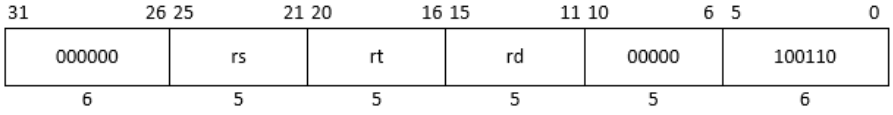
OR rd, rs, rt　　R 型



GPR[rd] ← GPR[rs] | GPR[rt]

（7） 按位异或

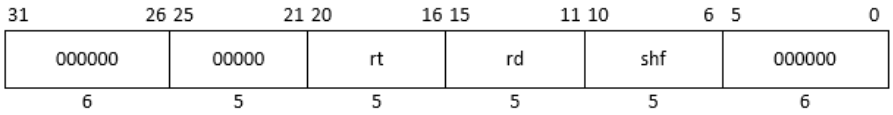
XOR rd, rs, rt　　R 型



GPR[rd] ← GPR[rs] ^ GPR[rt]

（8） 逻辑左移

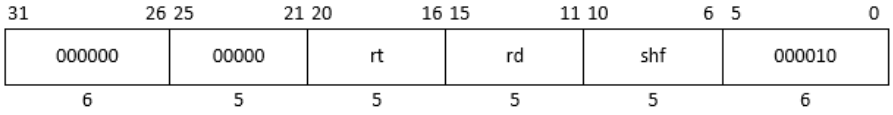
SLL rd, rt, shf　　R 型



GPR[rd] ← zero(GPR[rt]) << shf

（9） 逻辑右移

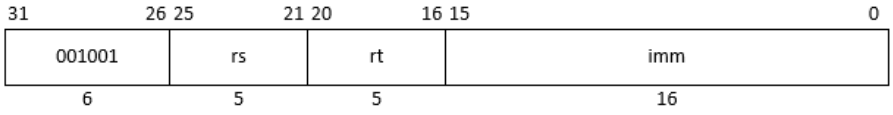
SRL rd, rt, shf　　R 型



GPR[rd] ← zero(GPR[rt]) >> shf

（10） 立即数、无符号加法

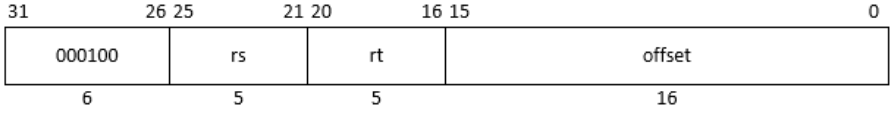
ADDIU rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] + sign_ext(imm)

（11） 相等跳转

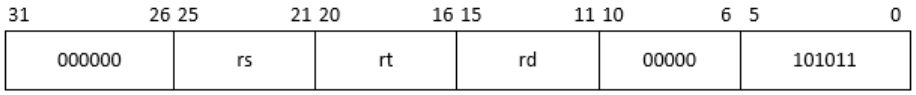
BEQ rs, rt, offset　　I 型



if GPR[rs] = GPR[rt] then PC ← B_PC + sign_ext(offset)<<2

（17） 无符号小于置位

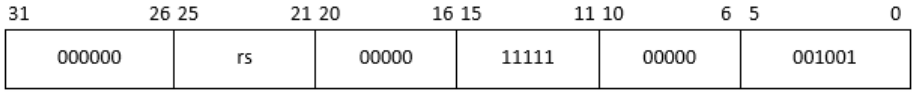
SLTU rd, rs, rt　　R 型



GPR[rd] ← (zero(GPR[rs]) < zero(GPR[rt]))

（18） 跳转寄存器并链接

JALR rs　　R 型

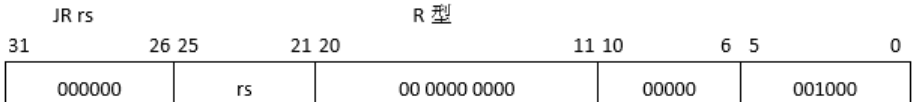


GPR[31] ← B_PC + 4, PC ← GPR[rs]

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（19） 跳转寄存器

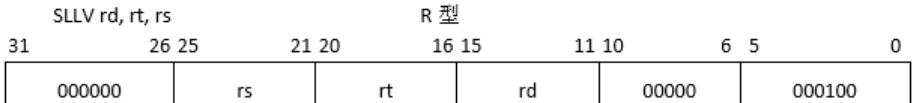
JR rs　　R 型



PC ← GPR[rs]

（20） 变量逻辑左移

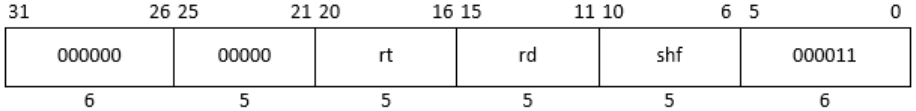
SLLV rd, rt, rs　　R 型



GPR[rd] ← zero(GPR[rt]) << GPR[rs]

（21） 算术右移

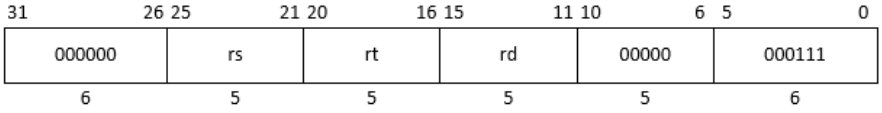
SRA rd, rt, shf　　R 型



GPR[rd] ← sign(GPR[rt]) >> shf

（22） 变量算术右移

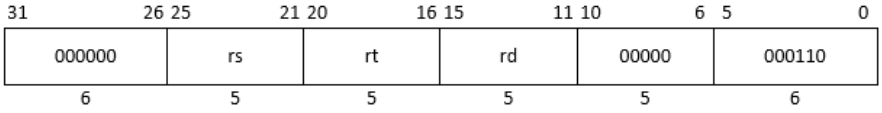
SRAV rd, rt, rs　　R 型



GPR[rd] ← sign(GPR[rt]) >> GPR[rs]

（23） 变量逻辑右移

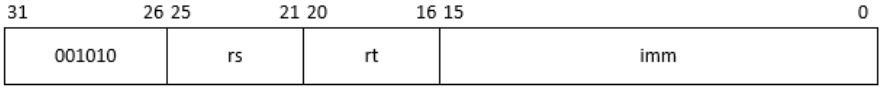
SRLV rd, rt, rs　　R 型



GPR[rd] ← zero(GPR[rt]) >> GPR[rs]

（24） 立即数有符号比较，小于置位

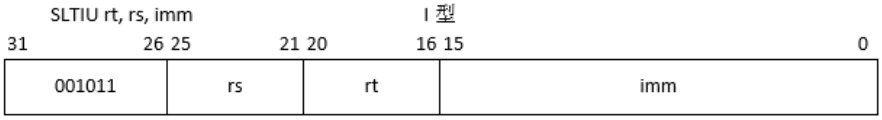
SLTI rt, rs, imm　　I 型



GPR[rt] ← (sign(GPR[rs]) < sign_ext(imm))

（25） 立即数无符号比较，小于置位

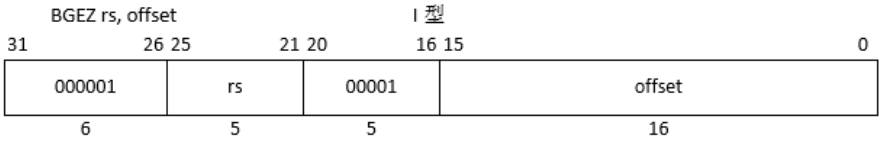
SLTIU rt, rs, imm　　I 型



GPR[rt] ← (zero(GPR[rs]) < sign_ext(imm))

（26） 大于或等于零跳转

BGEZ rs, offset　　I 型

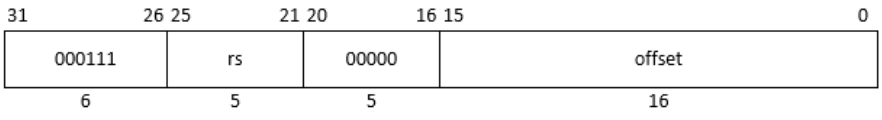


if GPR[rs]≥0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

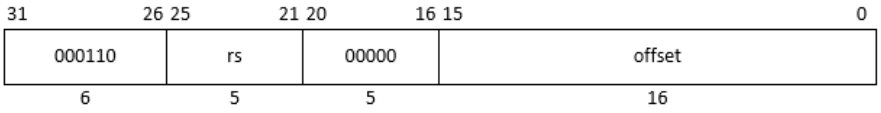
（27） 大于零跳转

BGTZ rs, offset　　I 型



（28） 小于或等于零跳转

BLEZ rs, offset　　I 型

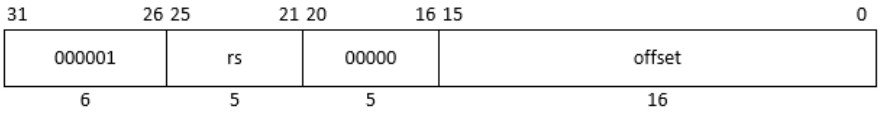


if GPR[rs] ≤ 0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（29） 小于零跳转

BLTZ rs, offset　　I 型

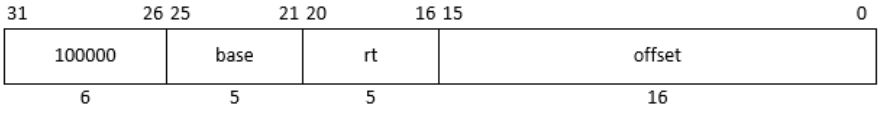


if GPR[rs] < 0 then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（30） 装载字节，并作符号扩展

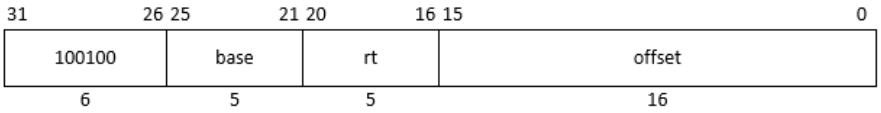
LB rt, offset(base)　　I 型



GPR[rt] ← sign(Mem[GPR[base] + sign_ext(offset)])

（31） 装载字节，并作无符号扩展

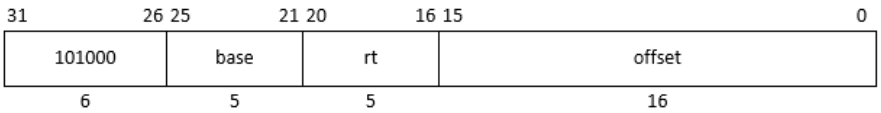
LBU rt, offset(base)　　I 型



GPR[rt] ← zero(Mem[GPR[base] + sign_ext(offset)])

（32） 存储字节

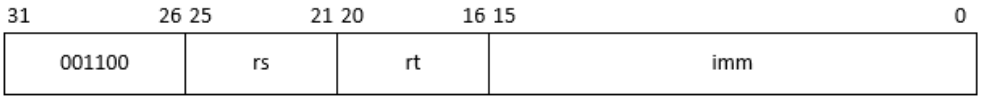
SB rt, offset(base)　　I 型



Mem[GPR[base] + sign_ext(offset)] ← GPR[rt]

（33） 立即数按位与

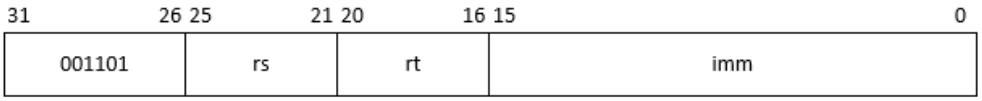
ANDI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] & zero_ext(imm)

（34） 立即数按位或

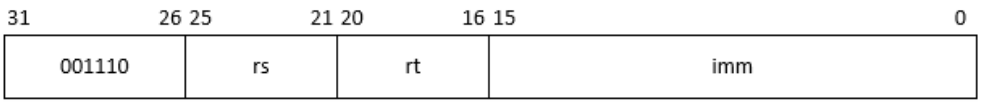
ORI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] | zero_ext(imm)

（35） 立即数按位异或

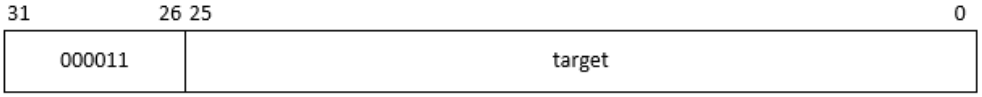
XORI rt, rs, imm　　I 型



GPR[rt] ← GPR[rs] ^ zero_ext(imm)

（36） 跳转和链接

JAL target　　J 型

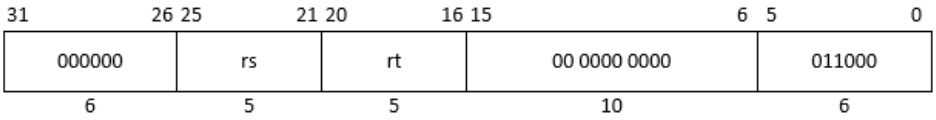


GPR[31] ← B_PC + 4, PC←{B_PC[31:28], target << 2}

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（37） 有符号字乘法

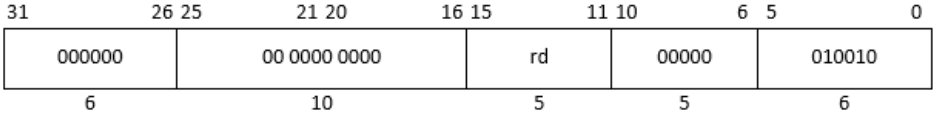
MULT rs, rt　　R 型



(HI, LO) ← sign(GPR[rs]) * sign(GPR[rt])

（38） 从 LO 寄存器取值

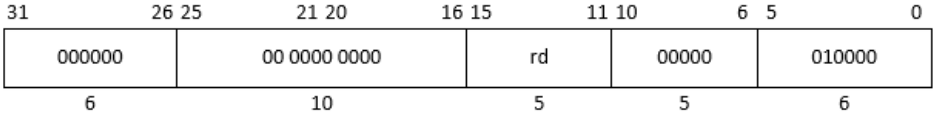
MFLO rd　　R 型



GPR[rd] ← [LO]

（39） 从 HI 寄存器取值

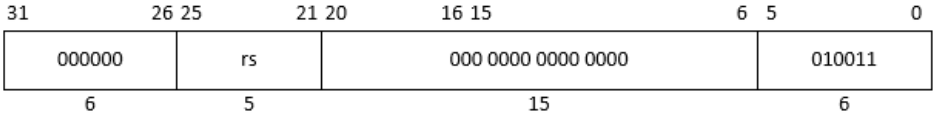
MFHI rd　　R 型



GPR[rd] ← [HI]

（40） 向 LO 寄存器存值

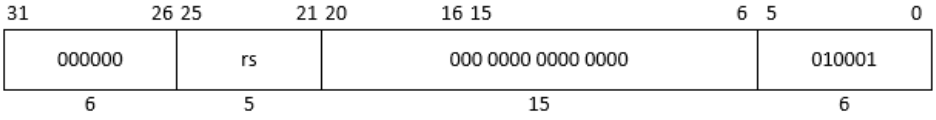
MTLO rs　　R 型



[LO] ← GPR[rs]

（41） 向 HI 寄存器存值

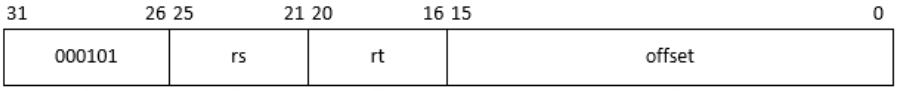
MTHI rs　　R 型



[HI] ← GPR[rs]

（12） 不等跳转

BNE rs, rt, offset　　I 型

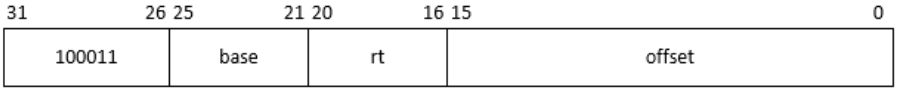


if GPR[rs] ≠ GPR[rt] then PC ← B_PC + sign_ext(offset)<<2

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。

（13） 装载字

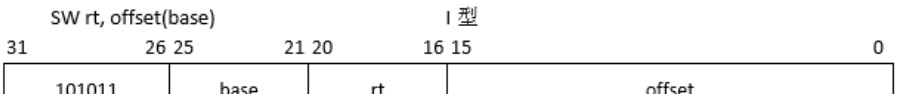
LW rt, offset(base)　　I 型



GPR[rt] ← Mem[GPR[base] + sign_ext(offset)]

（14） 存储字

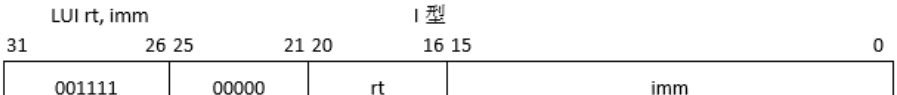
SW rt, offset(base)　　I 型



Mem[GPR[base] + sign_ext(offset)] ← GPR[rt]

（15） 立即数装载高位

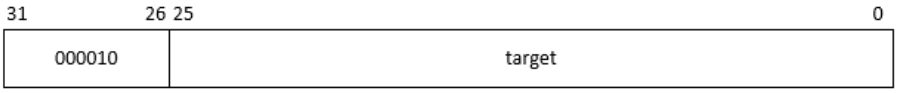
LUI rt, imm　　I 型



GPR[rt] ← {imm,16'd0}

（16） 直接跳转

J target　　J 型



PC ← {B_PC[31:28], target << 2}

B_PC：分支跳转参与运算的 PC，在不考虑延迟槽时为分支跳转指令的 PC，考虑延迟槽时为延迟槽指令的 PC，即分支跳转指令的 PC+4。