



《计算机组成原理实验》

实验报告

(实验二)

学院名称 : 数据科学与计算机学院

专业(班级) : 17级计科教务一班

学生姓名 : 陈鸿峥

学号 : 17341015

时间 : 2018 年 11 月 12 日

成绩：

实验二：单周期CPU设计与实现

一、 实验目的

1. 掌握单周期CPU数据通路图的构成、原理及其设计方法
2. 掌握单周期CPU的实现方法，代码实现方法
3. 认识和掌握指令与CPU的关系
4. 掌握测试单周期CPU的方法

二、 实验内容

设计一个单周期CPU，使其至少能实现以下指令功能操作。指令与格式如下：

表 1: 基本MIPS指令及其格式

	指令	功能	op	rs	rt	rd	sham/func
算术运算	add rd, rs, rt	$rd \leftarrow rs + rt$	000000	rs(5位)	rt(5位)	rd(5位)	reserved
	sub rd, rs, rt	$rd \leftarrow rs - rt$	000001	rs(5位)	rt(5位)	rd(5位)	reserved
	addiu rt, rs, imm	$rt \leftarrow rs + \text{SignExt}(imm)$	000010	rs(5位)	rt(5位)		imm16
逻辑运算	andi rt, rs, imm	$rt \leftarrow rs \& \text{ZeroExt}(imm)$	010000	rs(5位)	rt(5位)		imm16
	and rd, rs, rt	$rd \leftarrow rs \& rt$	010001	rs(5位)	rt(5位)	rd(5位)	reserved
	ori rt, rs, imm	$rt \leftarrow rs \text{ZeroExt}(imm)$	010010	rs(5位)	rt(5位)		imm16
	or rd, rs, rt	$rd \leftarrow rs rt$	010011	rs(5位)	rt(5位)	rd(5位)	reserved
移位	sll rd, rt, sa	$rd \leftarrow rt \ll \text{ZeroExt}(sa)$	011000	reserved	rt(5位)	rd(5位)	sa(5位)
比较	slti rt, rs, imm	$rs < \text{SignExt}(imm) ? rt=1 : rt=0$	011100	rs(5位)	rt(5位)		imm16
访存	sw rt, imm(rs)	$\text{mem}[rs + \text{SignExt}(imm)] \leftarrow rt$	100110	rs(5位)	rt(5位)		imm16
	lw rt, imm(rs)	$rt \leftarrow \text{mem}[rs + \text{SignExt}(imm)]$	100111	rs(5位)	rt(5位)		imm16
分支	beq rs, rt, imm	$rs == rt$? $pc + 4 + \text{SignExt}(imm) \ll 2$: $pc + 4$	110000	rs(5位)	rt(5位)		imm16
	bne rs, rt, imm	$rs != rt$? $pc + 4 + \text{SignExt}(imm) \ll 2$: $pc + 4$	110001	rs(5位)	rt(5位)		imm16
	bltz rs, imm	$rs < \$0$? $pc + 4 + \text{SignExt}(imm) \ll 2$: $pc + 4$	110010	rs(5位)	00000		imm16
跳转	j addr	$pc \leftarrow \{(pc+4)[31:28], addr[27:2], 2'b00\}$	111000				addr[27:2]
停机	halt	halt	111111				00000000000000000000000000000000(26位)

注意：reserved为预留部分，一般用0填充

三、 实验器材

电脑一台、Xilinx Vivado软件一套、Basys3板一块

四、 实验原理

I. 基本概念

i. 单周期CPU

单周期CPU是指CPU的每条指令都在一个时钟周期内完成，通常在每个时钟周期的上升沿触发。

CPU在处理指令时一般要经过以下五个阶段：

1. 取指(IF): 根据程序计数器PC中的指令地址，从存储器中取出一条指令，同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址；但遇到“地址转移”指令时，则需要对“转移地址”进行处理后送入PC。
2. 译码(ID): 对取指操作中得到的指令进行译码，确定该指令需要完成的操作，从而产生相应的操作控制信号，用于下一步的执行。
3. 执行(EXE): 根据指令译码得到的操作控制信号，执行指令动作。
4. 访存(MEM): 所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元，或者从存储器中得到数据地址单元中的数据。
5. 写回(WB): 将指令执行的结果或者访问存储器得到的数据写回相应的目标寄存器。

ii. 数据通路及控制信号

基本数据通路见图1，控制信号见表2。

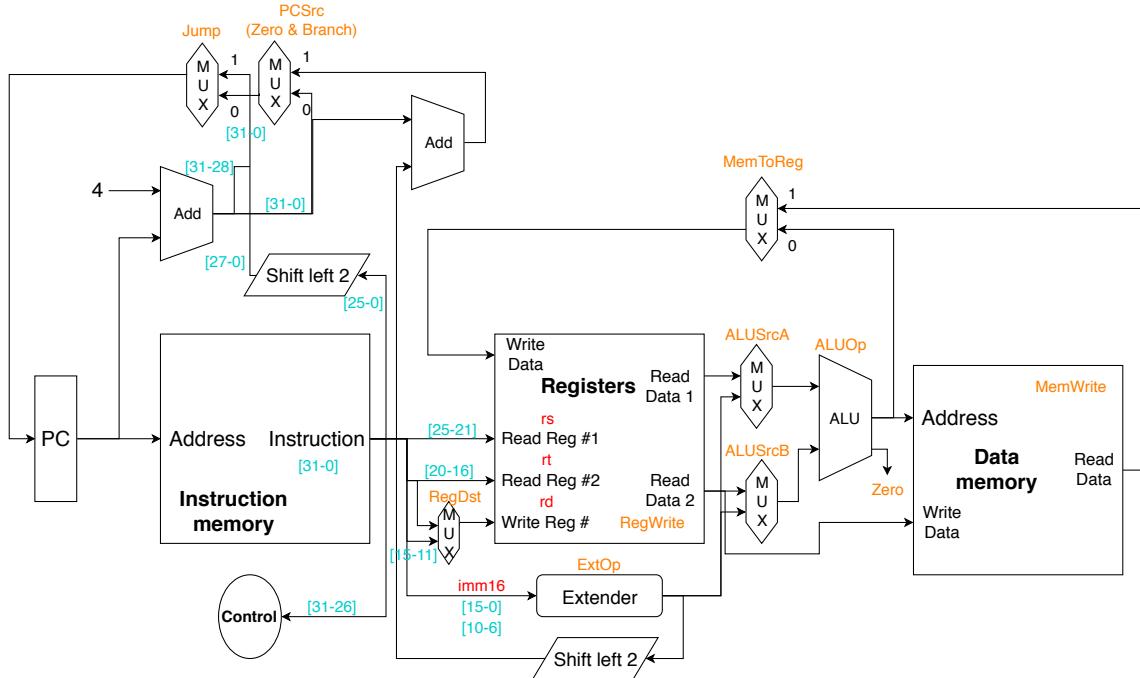


图 1: 基本数据通路

表 2: 控制信号

控制信号名	状态0	状态1
Reset	初始化PC为0	PC接收新地址
PCWre	PC不更改, halt	PC更改
RegDst	写入寄存器地址来自rt字段	写入寄存器地址来自rd字段
RegWrite	寄存器不可写	寄存器可写
ALUSrcA	寄存器rs内容	立即数sa
ALUSrcB	寄存器rt内容	立即数imm
ExtOp	零扩展	符号扩展
ALUOp	见表4	
MemToReg	ALU输出	内存读取
MemWrite	内存不可写	内存可写
Branch	非分支	分支
Jump	非跳转	跳转

iii. MIPS指令格式

MIPS指令可分为以下三种格式，见图2。其中各字段缩写含义如表3所示。

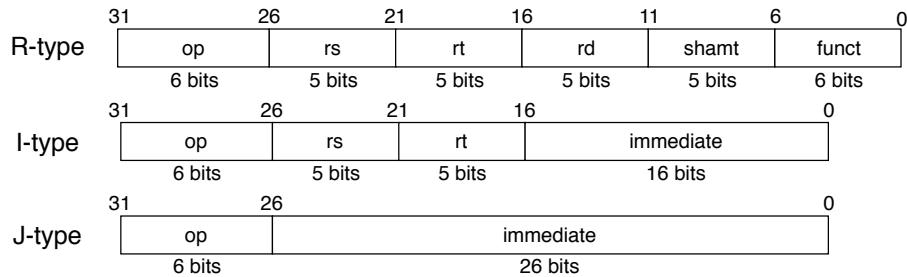


图 2: MIPS 指令类型

表 3: MIPS 指令各字段缩写含义

op	操作码
rs	只读，第一个源操作数寄存器地址/编号，范围为0x00~0x1F
rt	可读可写，第二个源操作数地址或目标操作数寄存器地址
rd	只写，目的操作数寄存器地址
sham(shift amt)	位移量，在移位指令中用于指定移多少位
funct	功能码，在R类型指令中配合op一起使用
immediate	16位立即数
address	目标转移地址

II. 各指令数据通路分析

公共的数据通路为取指和PC+4，见图3最左红色通路，下文将不再提及。

i. 算术逻辑运算

- add/sub/and/or指令均为R类型，数据通路相同，唯一不同为ALU操作码的选择。见图3，执行阶段从指令中读入rs、rt、rd三个寄存器的地址，然后从寄存器堆中读出rs和rt寄存器的内容，送至ALU进行运算，最后写回rd寄存器。
- addiu/andi/ori/slti指令均为I类型，数据通路相同，同样是ALU操作码不同。见图4，执行阶段从指令中读入rs、rt寄存器的地址，但rt是作为写入寄存器(RegDst=0)；指令低16位进行扩充，将rs的内容和立即数送入ALU运算，最后写回rt寄存器。注意addiu/slti进行符号扩展，andi/ori进行零扩展。
- sll指令为R类型，但是指令格式比较特殊。见图5，执行阶段从指令中读入rt寄存器的地址并取出，取指令的[10:6]位读出立即数sa并进行零扩展(ALUSrcA=1)，利用ALU对rt的内容及sa进行移位操作，结果写回rd寄存器。

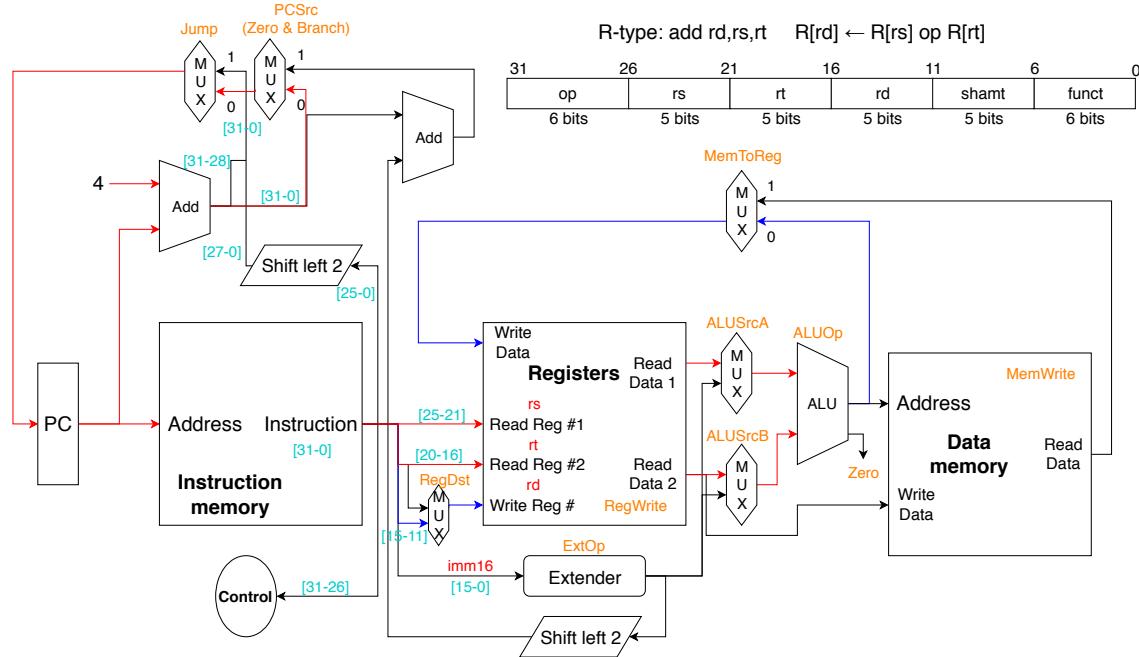


图 3: Add/sub/and/or通路

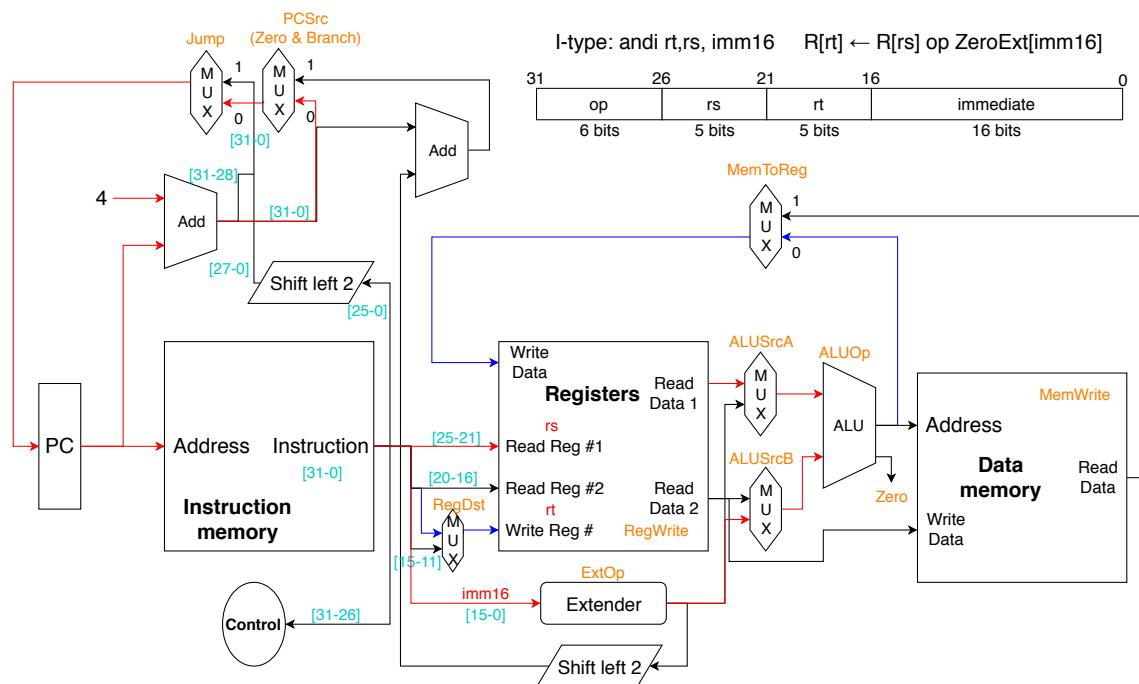


图 4: Addiu/andi/ori/slti通路

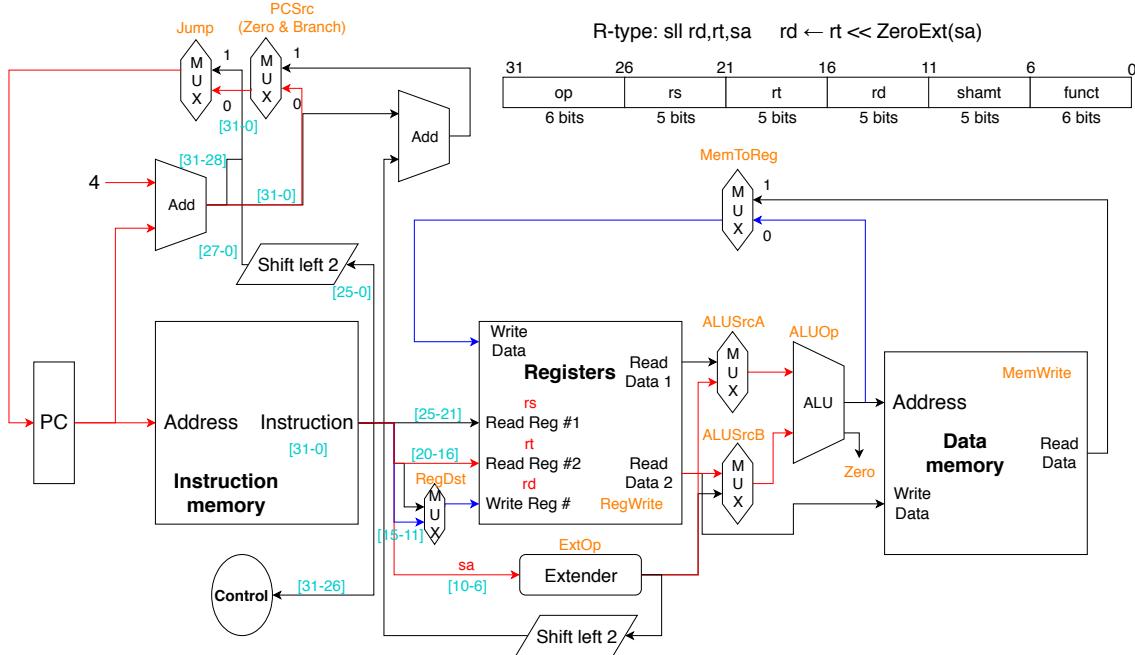


图 5: sll 通路

ii. 访存

1. sw为I类型。见图6，执行阶段从指令中读入rs、rt寄存器的地址，对偏移量(imm)进行符号扩展，与rs寄存器的内容相加得到内存地址，将rt寄存器的内容写入内存。
 2. lw为I类型。见图7，执行阶段从指令中读入rs、rt寄存器的地址，rt作为写入寄存器(RegDst=0)，对偏移量(imm)进行符号扩展，与rs寄存器的内容相加得到内存地址，将内存的内容写入rt寄存器。

iii. 分支跳转

由于MIPS指令为32位(4字节)，一般情况下PC每次自增都加4，即PC末两位一定为0，放在指令中（PC偏移/转移地址）即可省略末两位。指令读出时则要左移2位，将末尾的0补齐。为最大程度利用指令的每一位，跳转指令也是将高4位和低2位都省略掉了，读出时要补回。

1. beq/bne/bltz为I类型。见图8，执行阶段从指令中读入rs、rt寄存器的地址，利用ALU对rs、rt寄存器的内容进行相减(beq/bne)或有符号比较(bltz)，若结果为0，则Zero标志置1，否则置0；同时，立即数imm进行符号扩展，并左移两位，与原有的PC+4再相加，得到是否执行分支跳转指令(beq: PCSrc=Zero, bne/bltz: PCSrc=~Zero)。
 2. jump为j类型。见图9，执行阶段直接对指令的低26位左移2位，补上PC+4的高4位，得到跳转地址，更新PC。

iv. 停机指令

PCWrite置为0，不改变PC的值，PC不再变化

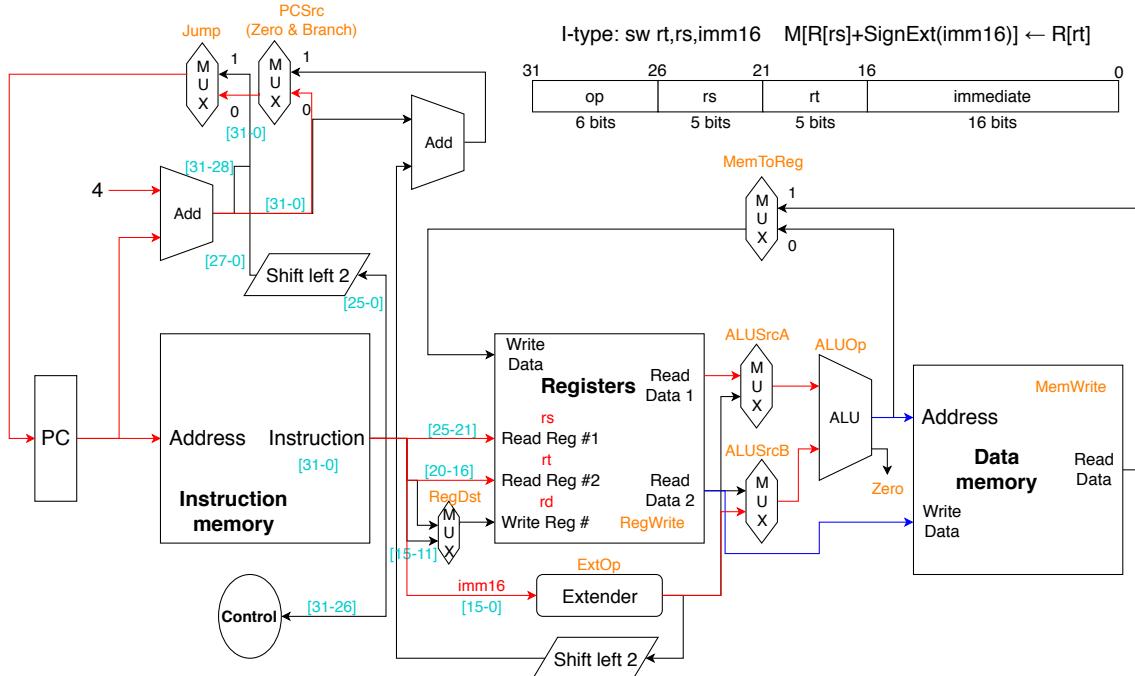


图 6: sw通路

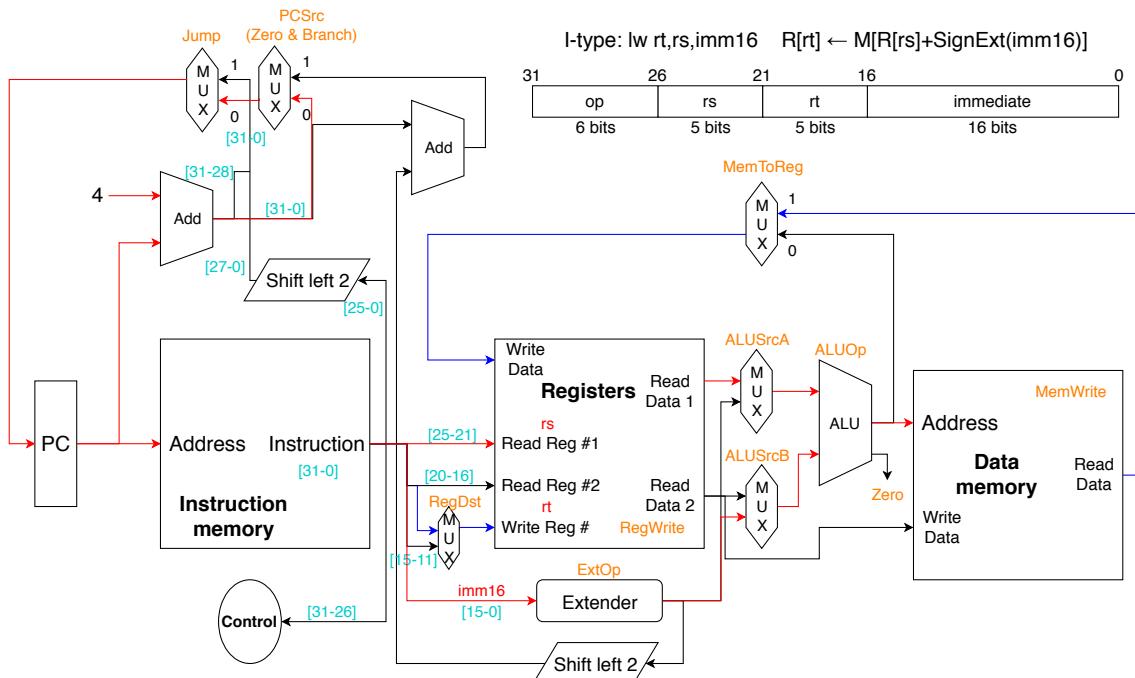


图 7: lw通路

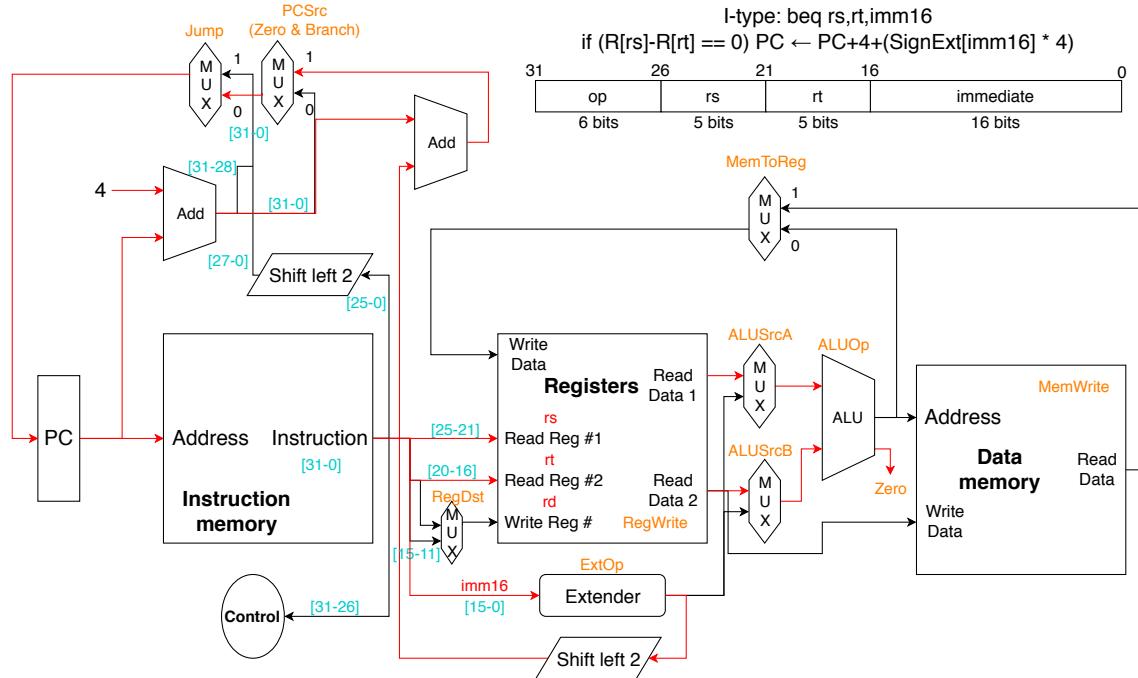


图 8: beq/bne/bltz通路

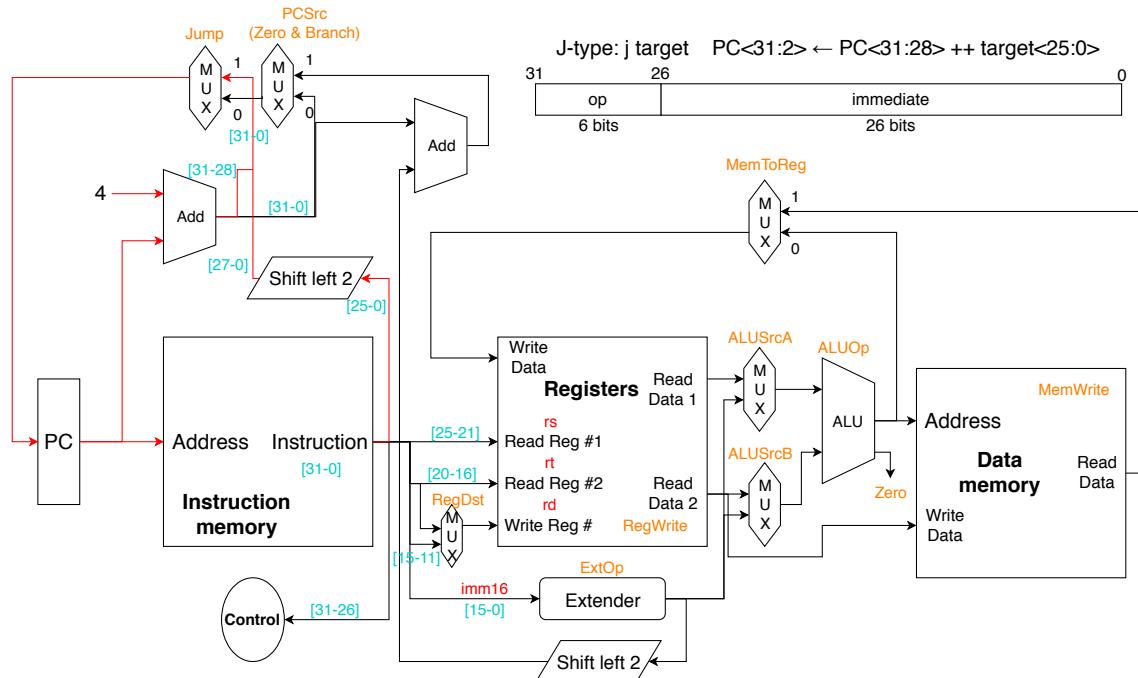


图 9: jump通路

III. 各组件实现

各组件实现详情见第8章节，均有详细注释。在这里仅仅提一些需要注意的点。

1. 指令存储器和数据存储器均采用小端(little endian)存储，并且位宽为8位，故生成的指令文件需要进行预处理，最先读入的应该是指令的低8位。这里用Python进行预处理并生成指令二进制代码文件，读入时直接将其初始化到指令存储器中。
2. 程序计数器(PC)需要初始化置零(0x000000)，遇到reset信号时同样置零，而且只有在PCWrite=1时才更新PC。
3. 多路选择器(MUX)需要实现一个32位和一个5位的，32位可以复用（创建多个实例）。
4. ALU的功能见表4。
5. 控制单元的设计见第4.4节。

表 4: ALU功能码

ALUOp[2:0]	功能	描述
000	$Y = A + B$	加
001	$Y = A - B$	减
010	$Y = B \ll A$	左移
011	$Y = A \vee B$	逻辑或
100	$Y = A \wedge B$	逻辑与
101	$Y = (A < B) ? 1 : 0$	比较无符号数
110	$Y = (((A < B) \&\& (A[31] == B[31]))$ $\quad \quad \quad \mid\mid ((A[31] == 1 \&\& B[31] == 0))$ $\quad \quad \quad ? 1 : 0$	比较有符号数
111	$Y = A \oplus B$	逻辑异或

IV. 控制单元

由前面的分析，可以得到控制单元的编码表，见表5。

五、 实验步骤

I. 仿真模拟

采用表6给出的汇编程序代码段进行测试。注意指令的二进制编码应该对应到MIPS的每一个字段上，特别注意sll的书写，负数用补码表示等。生成二进制指令文件后，CPU将其读入指令存储器并执行，看波形结果。

II. 上板

使用Basys3板运行所设计的CPU时，需要通过4个七段数码管来查看当前CPU的执行情况。

只考虑指令和数据的低8位，即指令存储器中的指令地址范围和数据存储器中的数据地址范围均为 0x00~ 0xFF。

表 5：指令控制器编码

	指令	op	RegDst	ExtSel	RegWrite	ALUSrcA/B	ALUOp	MemToReg	MemWrite	Branch	Jump
算术运算	add rd, rs, rt	000000	1	x	1 0	0	000	0	0	0	0
	sub rd, rs, rt	000001	1	x	1 0	0	001	0	0	0	0
	addiu rt, rs, imm	000010	0	1	1 0	1	000	0	0	0	0
	andi rt, rs, imm	010000	0	0	1 0	1	100	0	0	0	0
逻辑运算	and rd, rs, rt	010001	1	x	1 0	0	100	0	0	0	0
	ori rt, rs, imm	010010	0	0	1 0	1	011	0	0	0	0
	or rd, rs, rt	010011	1	x	1 0	0	011	0	0	0	0
移位	sll rd, rt, sa	011000	1	x	1 1	0	010	0	0	0	0
	slti rt, rs, imm	011100	0	1	1 0	1	110	0	0	0	0
访存	sw rt, imm(rs)	100110	x	1	0 0	1	000	x	1	0	0
	lw rt, imm(rs)	100111	0	1	1 0	1	000	1	0	0	0
	beq rs, rt, imm	110000	x	1	0 0	0	001	x	0	1	0
分支	bne rs, rt, imm	110001	x	1	0 0	0	001	x	0	1	0
	bltz rs, imm	110010	x	1	0 0	0	110	x	0	1	0
跳转	j addr	111000	x	x	0 x	x	x	x	0	0	1
停机	halt	111111	x	x	x x	x	x	x	x	x	x

表 6: 测试代码段指令

address	instruction	op	rs	rt	rd/imm	hex
0x00000000	addiu \$1,\$0,8	000010	00000	00001	0000 0000 0000 1000	0x08010008
0x00000004	ori \$2,\$0,2	010010	00000	00010	0000 0000 0000 0010	0x48020002
0x00000008	add \$3,\$2,\$1	000000	00010	00001	0001 1000 0000 0000	0x00411800
0x0000000C	sub \$5,\$3,\$2	000001	00011	00010	0010 1000 0000 0000	0x04622800
0x00000010	and \$4,\$5,\$2	010001	00101	00010	0010 0000 0000 0000	0x44A22000
0x00000014	or \$8,\$4,\$2	010011	00100	00010	0100 0000 0000 0000	0x4C824000
0x00000018	sll \$8,\$8,1	011000	00000	01000	0100 0000 0100 0000	0x60084040
0x0000001C	bne \$8,\$1,-2 (\neq ,转18)	110001	01000	00001	1111 1111 1111 1110	0xC501FFFE
0x00000020	slti \$6,\$2,4	011100	00010	00110	0000 0000 0000 0100	0x70460004
0x00000024	slti \$7,\$6,0	011100	00110	00111	0000 0000 0000 0000	0x70c70000
0x00000028	addiu \$7,\$7,8	000010	00111	00111	0000 0000 0000 1000	0x08e70008
0x0000002C	beq \$7,\$1,-2 (=,转28)	110000	00111	00001	1111 1111 1111 1110	0xC0E1FFFE
0x00000030	sw \$2,4(\$1)	100110	00001	00010	0000 0000 0000 0100	0x98220004
0x00000034	lw \$9,4(\$1)	100111	00001	01001	0000 0000 0000 0100	0x9C290004
0x00000038	addiu \$10,\$0,-2	000010	00000	01010	1111 1111 1111 1110	0x080AFFFE
0x0000003C	addiu \$10,\$10,1	000010	01010	01010	0000 0000 0000 0001	0x094A0001
0x00000040	bltz \$10,-2 (< 0,转3C)	110010	01010	00000	1111 1111 1111 1110	0xC940FFFE
0x00000044	andi \$11,\$2,2	010000	00010	01011	0000 0000 0000 0010	0x404B0002
0x00000048	j 0x00000050	111000	00000	00000	0000 0000 0001 0100	0xE0000014
0x0000004C	or \$8,\$4,\$2	010011	00100	00010	0100 0000 0000 0000	0x4C824000
0x00000050	halt	111111	00000	00000	0000 0000 0000 0000	0xFC000000

通过Basys3板上的开关SW15、SW14选择七段数码管显示的内容，具体显示的功能码如表7所示。

表 7: 数码管显示功能码

SW15	SW14	左边	右边
0	0	当前PC	下条PC
0	1	rs寄存器地址	rs寄存器数据
1	0	rt寄存器地址	rt寄存器数据
1	1	ALU结果输出	DB总线数据

III. 七段数码管显示电路

基本实现步骤如下：

1. 对Basys3板系统时钟信号(100MHz)进行分频，使得数码管内容能够正常显示。频率不宜过快，过快会导致全部数码管显示的都是8；也不宜过慢，否则数码管会出现暂留现象，无法连续显示。在本次实现中采用10kHz的频率进行显示。
2. 用上面得到的频率生成4进制计数器，用于产生4个数位选信号AN3-AN0。这4个数可控制哪个数码管亮，一共四组编码(1110、1101、1011、0111)，每组编码中只有一位为0（亮），其余都为1（灭），在每一个位选信号到来之时更新数码管显示。其实前两步可以一并完成，见第8章分频计数器一节。
3. 将从CPU接收到的相应数据转换为数码管显示信号，与位选信号一起送往数码管显示输出。

IV. 其他注意事项

1. 共阳极数码管

Basys3板的数码管均为共阳极，故设置七段数码管和位选信号时都要考虑0为亮，1为灭。

2. 两个时钟信号

CPU工作时钟和Basys3板系统时钟是两个不同的时钟，但是FPGA只支持单一全局时钟，否则在routing阶段会报错

`Poor placement for routing between an IO pin and BUFG`

故需要采取其他方法。只设置全局时钟为`clk`，并将CPU工作时钟`clk_cpu`与其同步，即在每一个`clk`上升沿时，赋值`in<=clk_cpu`，通过下面消抖处理后，将`in`作为真正的CPU工作时钟。

3. 消抖处理

如果一次触发持续一段时间不改变状态（如原状态是0，变更为1且保持100ns），则该触发是有效的人为触发，否则则视为抖动。故可以设置一个按键状态的历史记录(16位)，

如果连续14位都为1，则将clk_cpu设为高电平，否则为无效触发。具体实施细节见第8章写板电路一节。

V. 引脚分配

见表8。

表 8: 引脚分配表

引脚	名称	作用
W5	clk	全局时钟
T17	clk_cpu	CPU时钟（单脉冲信号）
V17	reset	复位信号
R2/T1	SW_in	数码管显示内容选择
W4-U2	AN3-AN0	数码管位选信号
W7-U7	seg6-seg0	七段数码管内容

VI. 代码层次结构

见图10。

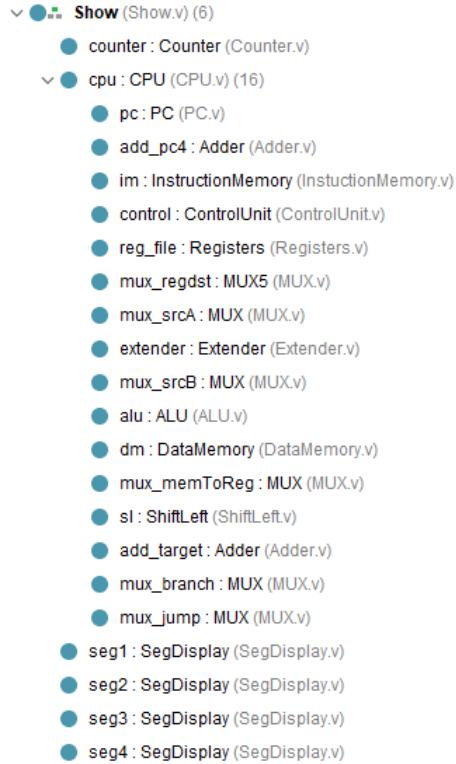


图 10: 代码层次结构

六、结果与分析

I. 仿真模拟

设置时钟周期为100ns，初始30ns为准备时间，然后将Reset设为1使其开始工作。仿真结果见图11到图15，每条指令的说明均已附在波形图之下。



图 11: 波形图1



图 12: 波形图2



图 13: 波形图3



图 14: 波形图4



图 15: 波形图5

II. 上板

几条代表性指令的Basys3板结果如下，其他指令结果见具体实现。从左到右从上到下四幅图依次是：当前/下条PC、rs寄存器地址/值、rt寄存器地址/值、ALU结果输出/DB总线数据。可以看见上板结果与前面仿真分析的结果相同，成功证明了程序的正确性。

1. 0x00: addiu \$1, \$0, 8



图 16: 0x00结果

2. 0x0C: sub \$5, \$3, \$2



图 17: 0x0C结果

3. 0x14: or \$8, \$4, \$2



图 18: 0x14结果

4. 0x18: sll \$8, \$8, 1



图 19: 0x18结果

5. 0x1C: bne \$8, \$1, -2



图 20: 0x1C结果

6. 0x30: sw \$2, 4(\$1)



图 21: 0x30结果

7. 0x34: lw \$9, 4(\$1)



图 22: 0x34结果

七、 实验心得

本次实验可以算是从大一入学到现在做过的最大型的项目了，单单是代码文件就有十几个，虽然弄清数据通路后并不难实现，但是其中的小细节却得十分小心。

Verilog代码编写阶段：

1. 区分非阻塞赋值`<=`和阻塞赋值`=`。简单理解，非阻塞赋值可以并行执行，而阻塞赋值只能串行执行，这里就考虑到指令之间相关性的问题了。如果指令之间不相关，则可直接并行，否则只能串行。
2. 区分`wire`和`reg`类型。简单来说，`wire`主要起信号间连接作用，由于它不保存状态，故它的值可以随时改变，不受时钟信号的限制，有点像Python里对象的引用赋值。而`reg`顾名思义为寄存器，类似于高级程序语言的变量，可以存储数值、保存输出状态。

3. 由于所有module的输入输出均默认为wire类型，写代码的时候常常会忘记加reg导致出错。不过这个错误较为明显，在静态代码检查时即被检查出来。
4. 在模块输入参数最后常常多了一个，报错(ISE port connections mix)。
5. always @写时序逻辑确定好上升沿和下降沿通常没有问题，而组合逻辑常常忘记写部分输入参数，导致该输入改变了，但寄存器的值并没有改变。
6. 要在每个模块合适的位置对寄存器进行初始化initial，否则在仿真中会输出xxxx。
7. 位宽[x:y]要记清楚，对应好模块接口，否则多的位宽在跑实施时会报错。当然少了位宽没发现的话，调试起来也会很麻烦。
8. 应在时钟下降沿才进行写入操作
9. case要记得写default条件，注意检查是否所有情况都已涵盖，要不然FPGA会以奇怪的方式对未符合的情况进行匹配。
10. vivado编译器对代码检查还是很不严格，比如在条件判断中写add >= 0 && add <= 255，可能后者会被认为是赋值，导致结果错误。这是仿真不会出现但具体运行时会出现的问题。
11. 32位的二进制编码不要写成8'h00000000的形式，会被提示位数少于预定值，可能出现不可预料的错误，这个在0x3C这条指令处卡了很久。

Vivado模拟仿真上板阶段：

1. 由于本实验采用小端存储数据，在读入时也要小心。第一次测试时因为测试指令写入文件的顺序不对，导致读入的指令全部都是反的，进而没法正常执行。通过Python重新对指令文件处理后才正常执行。
2. 前面提及的双时钟问题，折腾了非常久的时间。
3. 初始PC值要从-4开始否则第一条指令如果有写操作则无法写入
4. 写限制文件时，漏了一个端口设置，导致到最后一步生成二进制位流才报错，又得重新跑实施综合，很费时间。
5. 可能是电脑驱动的问题，一直报No hardware target is open，没法将程序写入FPGA板，只能连着电脑运行程序，这个问题最终也没有解决。只好用FAT32格式化的U盘，通过Basys3的J2-USB端口写入bit文件。

总的来说，本次实验收获非常非常多。尽管仿真没有出现问题，但是一到上板就出现了很多不可预料的问题，这些地方卡了我几天时间，非常难受。要解决硬件的问题一定要冷静下来，认真分析每条语句出现的漏洞，并且查看vivado给出的warning提示。

好的方面是，我更加熟悉了Verilog的语法，熟悉Vivado的使用，同时也了解了硬件设计与软件设计的巨大区别。如FPGA要考虑指令之间是否存在依赖关系、是否可以并行等问题，而平时我们在程序设计课或数据结构课上写的算法或项目往往是不需要考虑并行的问题的，这一点在硬件设计时要考虑清楚。当然了，更好地了解硬件也能够让我们编写出更高质量的软件代码，这两者应该相辅相成相互促进。

八、 程序清单

I. CPU主模块

```

1 timescale ins / 1ps
2
3 module CPU (
4     input clk, reset,
5     output RegDst,
6     output ExtSel,
7     output RegWrite, MemWrite,
8     output ALUSrcA, ALUSrcB,
9     output [2:0] ALUOp,
10    output MemToReg,
11    output Branch, Jump, Zero,
12    output PCWrite,
13    output [31:0] currPC, nextPC, instruction, alu_res
14    output wire [31:0] d1, d2, rsData, rtData, dbData,
15    output wire [4:0] rs, rt, rd, sa
16 );
17
18     wire [5:0] opcode;
19     wire [15:0] imm;
20     wire [31:0] pc4;
21
22     assign opcode = instruction[31:26];
23     assign rs = instruction[25:21];
24     assign rt = instruction[20:16];
25     assign rd = instruction[15:11];
26     assign sa = instruction[10:6];
27     assign imm = instruction[15:0];
28
29     PC pc(
30         .clk(clk),
31         .reset(reset),
32         .PCWrite(PCWrite),
33         .currPC(currPC),
34         .nextPC(nextPC)
35     );
36
37     Adder add_pc4(
38         .A(currPC),
39         .B({29{1'b0}},3'b100),
40         .res(pc4)
41     );
42
43 // instruction fetch (IF)
44 InstructionMemory im(
45     .address(currPC),
46     .dataOut(instruction)
47 );
48
49 // instruction decode (ID)
50 ControlUnit control(
51     // input
52     .opcode(opcode),
53     .Zero(Zero),
54     // output
55     .RegDst(RegDst),
56     .ExtSel(ExtSel),
57     .RegWrite(RegWrite),
58     .ALUSrcA(ALUSrcA),
59     .ALUSrcB(ALUSrcB),
60     .ALUOp(ALUOp),
61     .MemToReg(MemToReg),
62     .MemWrite(MemWrite),
63     .Branch(Branch),
64     .Jump(Jump),
65     .PCWrite(PCWrite)
66 );
67
68 // execution (EXE)
69 Registers reg_file(
70     .clk(clk),
71     .reset(reset),
72     .r1(rs),
73     .r2(rt),
74     .wr(mux_srcA.res),
75     .RegWrite(RegWrite),
76     .wd(mux_memToReg.res)
77     // d1 -> mux_srcA.A
78     // d2 -> mux_srcB.A / dm.dataIn
79 );
80
81     assign d1 = mux_srcA.res;
82     assign d2 = mux_srcB.res;
83     assign rsData = reg_file.d1;
84     assign rtData = reg_file.d2;
85
86     MUX5 mux_regdst(
87         .Sel(RegDst),
88         .A(rt),
89         .B(rd)
90         // res -> reg_file.wr
91     );
92
93     MUX mux_srcA(
94         .Sel(ALUSrcA),
95         .A(reg_file.d1),
96         .B({27{1'b0}},sa)
97         // res -> alu.A
98     );
99
100    Extender extender(
101        .Sel(ExtSel),
102        .dataIn(imm)
103        // dataOut -> mux_srcB.B / sl_16
104    );
105
106    MUX mux_srcB(
107        .Sel(ALUSrcB),
108        .A(reg_file.d2),
109        .B(extender.dataOut)
110        // res -> alu.B
111    );
112
113    ALU alu(
114        .op(ALUOp),
115        .A(mux_srcA.res),
116        .B(mux_srcB.res),
117        // res -> dm.address / mux_memToReg.A
118        .res(alu_res),
119        .zero(Zero)
120    );
121
122 // access memory (MEM)
123 DataMemory dm(
124     .clk(clk),
125     .reset(reset),
126     .address(alu_res),
127     .MemWrite(MemWrite),
128     .dataIn(reg_file.d2)
129     // dataOut -> mux_memToReg
130 );
131
132 // write back (WB)
133 MUX mux_memToReg(
134     .Sel(MemToReg),
135     .A(alu_res),
136     .B(dm.dataOut)
137     // res -> reg_file.wd
138 );
139
140     assign dbData = mux_memToReg.res;
141
142 // jump & branch
143 ShiftLeft sl(
144     .dataIn(extender.dataOut)
145     // dataOut -> add_target.B
146 );
147
148     Adder add_target(
149         .A(pc4),
150         .B(sl.dataOut)
151         // res -> mux_branch.B
152 );
153
154     MUX mux_branch(
155         .Sel(Branch),
156         .A(pc4),
157         .B(add_target.res)
158         // res -> mux_jump.A
159     );
160
161     MUX mux_jump(

```

```

162     .Sel(Jump),
163     .A(mux_branch.res),
164     .B({pc4[31:28],instruction[25:0],2'b00}),
165     .res(nextPC)
166   );
167
168 endmodule

```

II. 指令存储器

III. 程序计数器(PC)

IV. 寄存器堆

```

1 module Registers (
2     input clk,
3     input reset,
4     input [4:0] r1, // read reg #1 address
5     input [4:0] r2, // read reg #2 address
6     input [4:0] wr, // write reg address
7     input RegWrite,
8     input [31:0] wd, // write data
9     output [31:0] d1, // read data 1
10    output [31:0] d2 // read data 2
11 );
12
13 reg [31:0] register [0:31]; // 32 bits (bandwidth)
14     * #32 (address)
15
16 // initialization
17 integer i;
18 initial begin
19     for (i = 0; i < 32; i = i + 1)
20         register[i] = 0;
21 end
22
23 // read data
24 assign d1 = (r1 == 0) ? 0 : register[r1];
25 assign d2 = (r2 == 0) ? 0 : register[r2];
26
27 // write data
28
29
30
31
32
33
34
35
36 end
37
38 endmodule



## VI. 控制单元


1 module ControlUnit (
2     input [5:0] opcode,
3     input Zero,
4     output reg RegDst,
5     output reg ExtSel,
6     output reg RegWrite,
7     output reg ALUSrcA,
8     output reg ALUSrcB,
9     output reg [2:0] ALUOp,
10    output reg MemToReg,
11    output reg MemWrite,
12    output reg Branch,
13    output reg Jump,
14    output reg PCWrite
15 );
16
17 always @ (opcode or Zero) begin // Zero!
18     RegDst    <= 0;
19     ExtSel   <= 0;
20     RegWrite <= 1;

```

V. 数据存储器

```

module DataMemory (
    input clk,
    input reset,
    input [31:0] address,
    input MemWrite,
    input [31:0] dataIn, // write data
    output [31:0] dataOut // read data
);

reg [7:0] memory [0:255]; // 8 bits (bandwidth) *
#256 (address)

// initialization
integer i;
initial begin
    for (i = 0; i < 256; i = i + 1)
        memory[i] = 0;
end

// read data
assign dataOut = (address == 0) ? 0 : {memory[
    address + 3],
    memory[
        address +
        2],
    memory[
        address +
        1],
    memory[
        address
    ]};

// write data
always @(posedge clk) begin
    if (reset == 0)
        memory[0] <= 0;
    else if (MemWrite == 1 && address != 0) begin
        // do not use <=255!!!
        // little endian
        memory[address + 3] <= dataIn[31:24];
        memory[address + 2] <= dataIn[23:16];
        memory[address + 1] <= dataIn[15:8];
        memory[address] <= dataIn[7:0];
    end
end

```

VI. 控制单元

```

module ControlUnit (
    input [5:0] opcode,
    input Zero,
    output reg RegDst,
    output reg ExtSel,
    output reg RegWrite,
    output reg ALUSrcA,
    output reg ALUSrcB,
    output reg [2:0] ALUOp,
    output reg MemToReg,
    output reg MemWrite,
    output reg Branch,
    output reg Jump,
    output reg PCWrite
);

always @ (opcode or Zero) begin // Zero!
    RegDst      <= 0;
    ExtSel     <= 0;
    RegWrite   <= 1;

```

```

21 ALUSrcA <= 0;
22 ALUSrcB <= 0;
23 ALUOp <= 3'b000;
24 MemToReg <= 0;
25 MemWrite <= 0;
26 Branch <= 0;
27 Jump <= 0;
28 PCWrite <= 1;
29 case (opcode)
30   6'b000000: begin // add rd, rs, rt
31     RegDst <= 1;
32     end
33   6'b000001: begin // sub rd, rs, rt
34     RegDst <= 1;
35     ALUOp <= 3'b001;
36     end
37   6'b000010: begin // addiu rt, rs, imm
38     ExtSel <= 1; // ???
39     ALUSrcB <= 1;
40     end
41   6'b010000: begin // andi rt, rs, imm
42     ALUSrcB <= 1;
43     ALUOp <= 3'b100;
44     end
45   6'b010001: begin // and rd, rs, rt
46     RegDst <= 1;
47     ALUOp <= 3'b100;
48     end
49   6'b010010: begin // ori rt, rs, imm
50     ALUSrcB <= 1;
51     ALUOp <= 3'b011;
52     end
53   6'b010011: begin // or rd, rs, rt
54     RegDst <= 1;
55     ALUOp <= 3'b011;
56     end
57   6'b011000: begin // sll rd, rt, sa
58     RegDst <= 1;
59     ALUSrcA <= 1;
60     ALUOp <= 3'b010;
61     end
62   6'b011100: begin // slti rt, rs, imm
63     ExtSel <= 1;
64     ALUSrcB <= 1; // remember!
65     ALUOp <= 3'b110;
66     end
67   6'b100110: begin // sw rt, imm(rs)
68     ExtSel <= 1;
69     RegWrite <= 0;
70     ALUSrcB <= 1;
71     MemWrite <= 1;
72     end
73   6'b100111: begin // lw rt, imm(rs)
74     ExtSel <= 1;
75     ALUSrcB <= 1;
76     MemToReg <= 1;
77     end
78   6'b110000: begin // beq rs, rt, imm
79     ExtSel <= 1;
80     RegWrite <= 0;
81     ALUOp <= 3'b001;
82     Branch <= Zero;
83     end
84   6'b110001: begin // bne rs, rt, imm
85     ExtSel <= 1;
86     RegWrite <= 0;
87     ALUOp <= 3'b001;
88     Branch <= ~Zero; // (rs - rt == 0)
89     ? 1 : 0 Not equal!
90     end
91   6'b110010: begin // bltz rs, imm
92     ExtSel <= 1;
93     RegWrite <= 0;
94     ALUOp <= 3'b110; // compare sign
95     Branch <= ~Zero; // a < 0 ? 1 : 0
96     end
97   6'b111000: begin // j addr
98     RegWrite <= 0;
99     Jump <= 1;
100    end
101   6'b111111: begin // halt
102     PCWrite <= 0;
103     end
104   end
105 endmodule

```

VII. 算术逻辑单元(ALU)

```

1 module ALU (
2   input [2:0] op,
3   input [31:0] A,
4   input [31:0] B,
5   output reg [31:0] res,
6   output zero
7 );
8
9 initial begin
10   res = 0;
11 end
12
13 always @ (op or A or B) begin
14   case (op)
15     3'b000: res = A + B;
16     3'b001: res = A - B;
17     3'b010: res = B << A; // B first!
18     3'b011: res = A | B;
19     3'b100: res = A & B;
20     3'b101: res = (A < B) ? 1 : 0;
21     3'b110: res = ((A < B) && A[31] == B[31]) ||
22       both pos/neg num
23       || (A[31] == 1 && B[31] == 0)
24       // A neg B pos
25       ? 1 : 0; // not 8'h00000001 !!!
26     3'b111: res = A ^ B;
27   endcase
28 end
29
30 assign zero = (res == 0) ? 1 : 0;
31 endmodule

```

VIII. 多路选择器(MUX)

```

1 module MUX (
2   input Sel,
3   input [31:0] A,
4   input [31:0] B,
5   output reg [31:0] res
6 );
7
8 always @ (Sel or A or B) begin
9   res <= (Sel == 0) ? A : B;
10 end
11
12 endmodule
13
14 module MUX5 (
15   input Sel,
16   input [4:0] A,
17   input [4:0] B,
18   output reg [4:0] res
19 );
20
21 always @ (Sel or A or B) begin
22   res <= (Sel == 0) ? A : B;
23 end
24
25 endmodule

```

IX. 数据扩展器

```

1 module Extender (
2   input Sel,
3   input [15:0] dataIn,
4   output reg [31:0] dataOut
5 );
6
7 initial dataOut = 0;
8
9 always @ (Sel or dataIn) begin // dataIn!!!
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
994
995
996
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1193
1194
1195
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1204
1205
1206
1206
1207
1208
1208
1209
1209
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
1689
1689
1690
1690
1691
1691
1692
1692
1693
1693
1694
1694
1695
1695
1696
1696
1697
1697
1698
1698
1699
1699
1700
1700
1701
1701
1702
1702
1703
1703
1704
1704
1705
1705
1706
1706
1707
1707
1708
1708
1709
1709
1710
1710
1711
1711
1712
1712
1713
1713
1714
1714
1715
1715
1716
1716
1717
1717
1718
1718
1719
1719
1720
1720
1721
1721
1722
1722
1723
1723
1724
1724
1725
1725
1726
1726
1727
1727
1728
1728
1729
1729
1730
1730
1731
1731
1732
1732
1733
1733
1734
1734
```

VII. 算术逻辑单元(ALU)

VIII. 多路选择器(MUX)

IX. 数据扩展器

```

10      if (Sel == 0) // ZeroExt           12      if (clr == 1) // reset
11          dataOut = {{16{1'b0}},dataIn[15:0]};    13      begin
12      else // SignExt                14          clk_seg <= 0;
13          dataOut = {{16{dataIn[15]}},dataIn[15:0]}; 15          count_dis <= 0;
14      end                           16      end
15                                         17      else if (count_dis == 50_000 - 1) // return 0
16  endmodule                         18      begin

```

X. 仿真代码

XI. 分频计数器

```

1 module Counter(
2     input clr, // clear, say reset
3     input clk, // original clock
4     output reg [1:0] count_4,
5     output reg clk_seg
6 );
7
8 // display 10kHz
9 reg [16:0] count_dis; // 26 bits to store count:
10    2^17 > 10^5
11 always @ (posedge clk or posedge clr)
12 begin

```

```

if (clr == 1) // reset
begin
    clk_seg <= 0;
    count_dis <= 0;
end
else if (count_dis == 50_000 - 1) // return 0
begin
    clk_seg <= ~clk_seg;
    count_dis <= 0;
end
else
begin
    clk_seg <= clk_seg;
    count_dis <= count_dis + 1;
end
end

always @ (posedge clk_seg or posedge clr)
begin
    if (clr == 1 || count_4 == 4)
        count_4 <= 0;
    else
        count_4 <= count_4 + 1;
end
endmodule

```

XII. 七段数码管

```

1 module SegDisplay (
2     input [3:0] data,
3     output reg [6:0] dispcode
4 );
5
6     always @ (data)
7         case(data)
8             // 0: on      1: off
9             0: dispcode = 7'b000_0001; // remember!
10            1: dispcode = 7'b100_1111;
11            2: dispcode = 7'b001_0010;
12            3: dispcode = 7'b000_0110;
13            4: dispcode = 7'b100_1100;
14            5: dispcode = 7'b010_0100;
15            6: dispcode = 7'b010_0000;
16            7: dispcode = 7'b000_1111;
17            8: dispcode = 7'b000_0000;
18            9: dispcode = 7'b000_0100;
19            10: dispcode = 7'b000_1000; // A
20            11: dispcode = 7'b110_0000; // b
21            12: dispcode = 7'b011_0001; // C
22            13: dispcode = 7'b100_0010; // d
23            14: dispcode = 7'b001_0000; // e
24            15: dispcode = 7'b011_1000; // F
25
26     endcase
27
28 endmodule

```

XIII. 写板电路

```

1 module Show(
2     input clk,
3     input clk_cpu, // button
4     input reset,
5     input [1:0] SW_in,
6     output reg [6:0] dispcode,
7     output reg [3:0] out
8 );
9
10 // synchronize and reduce jitter
11 reg in_detected = 1'b0;
12 reg [15:0] inhistory = 16'h0000;
13 always @(posedge clk) begin
14     inhistory = {inhistory[15:0], clk_cpu};
15     if (inhistory == 16'b0001111111111111)
16         in_detected <= 1'b1;
17     else
18         in_detected <= 1'b0;
19 end
20
21 wire [1:0] seg_num; // not reg!
22

```

```

23     Counter counter(
24         .clk(clk),
25         // output clock/counter
26         .count_4(seg_num)
27     );
28
29     reg [31:0] firstNum;
30     reg [31:0] secondNum;
31
32     initial firstNum = 0;
33     initial secondNum = 0;
34
35     wire [31:0] currPC, nextPC, rsData, rtData, dbData;
36     wire [4:0] alu_res;
37
38     CPU cpu(
39         // input
40         .clk(in_detected),
41         .reset(reset),
42         // output
43         .currPC(currPC),
44         .nextPC(nextPC),
45         .rs(rs),
46         .rt(rt),
47         .rsData(rsData),
48         .rtData(rtData),
49         .dbData(dbData),
50         .alu_res(alu_res)
51     );
52
53     always @ (SW_in) begin
54         case (SW_in)
55             2'b00: begin
56                 firstNum <= currPC;
57                 secondNum <= nextPC;
58             end
59             2'b01: begin
60                 firstNum <= {{27{1'b0}}, rs};
61                 secondNum <= rsData;
62             end
63             2'b10: begin
64                 firstNum <= {{27{1'b0}}, rt};
65                 secondNum <= rtData;
66             end
67             2'b11: begin
68                 firstNum <= alu_res;
69                 secondNum <= dbData;
70             end
71         endcase
72     end
73
74     SegDisplay seg1(
75         .data(firstNum[7:4]),
76         // .dispcode
77     );
78
79     SegDisplay seg2(
80         .data(firstNum[3:0]),
81         // .dispcode
82     );
83
84     SegDisplay seg3(
85         .data(secondNum[7:4]),
86         // .dispcode
87     );
88
89     SegDisplay seg4(
90         .data(secondNum[3:0])
91
92         // .dispcode
93     );
94
95     always @ (seg_num or firstNum or secondNum)
96         case (seg_num)
97             0: begin
98                 out = 4'b1110;
99                 dispcode = seg4.dispcode;
100            end
101            1: begin
102                 out = 4'b1101;
103                 dispcode = seg3.dispcode;
104            end
105            2: begin
106                 out = 4'b1011;
107                 dispcode = seg2.dispcode;
108            end
109            3: begin
110                 out = 4'b0111;
111                 dispcode = seg1.dispcode;
112            end
113        endcase
114    endmodule

```

XIV. 限制文件(constraints.xdc)

```

1 | set_property PACKAGE_PIN W5 [get_ports clk]
2 | set_property PACKAGE_PIN T17 [get_ports clk_cpu]
3 | set_property PACKAGE_PIN V17 [get_ports reset]
4 | set_property PACKAGE_PIN R2 [get_ports {SW_in[1]}]
5 | set_property PACKAGE_PIN T1 [get_ports {SW_in[0]}]
6 | set_property PACKAGE_PIN W4 [get_ports {out[3]}]
7 | set_property PACKAGE_PIN V4 [get_ports {out[2]}]
8 | set_property PACKAGE_PIN U4 [get_ports {out[1]}]
9 | set_property PACKAGE_PIN U2 [get_ports {out[0]}]
10 | set_property PACKAGE_PIN W7 [get_ports {dispcode[6]}]
11 | set_property PACKAGE_PIN W6 [get_ports {dispcode[5]}]
12 | set_property PACKAGE_PIN U8 [get_ports {dispcode[4]}]
13 | set_property PACKAGE_PIN V8 [get_ports {dispcode[3]}]
14 | set_property PACKAGE_PIN V5 [get_ports {dispcode[2]}]
15 | set_property PACKAGE_PIN V5 [get_ports {dispcode[1]}]
16 | set_property PACKAGE_PIN U7 [get_ports {dispcode[0]}]
17
18 | set_property IOSTANDARD LVCMOS33 [get_ports clk]
19 | set_property IOSTANDARD LVCMOS33 [get_ports clk_cpu]
20 | set_property IOSTANDARD LVCMOS33 [get_ports reset]
21 | set_property IOSTANDARD LVCMOS33 [get_ports {SW_in[1]}]
22 | set_property IOSTANDARD LVCMOS33 [get_ports {SW_in[0]}]
23 | set_property IOSTANDARD LVCMOS33 [get_ports {out[3]}]
24 | set_property IOSTANDARD LVCMOS33 [get_ports {out[2]}]
25 | set_property IOSTANDARD LVCMOS33 [get_ports {out[1]}]
26 | set_property IOSTANDARD LVCMOS33 [get_ports {out[0]}]
27 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[6]}]
28 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[5]}]
29 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[4]}]
30 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[3]}]
31 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[2]}]
32 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[1]}]
33 | set_property IOSTANDARD LVCMOS33 [get_ports {dispcode[0]}]

```