

2019 Spring ICSII

## 计算机体系结构实验

### 实验一

#### 32位MIPS单周期处理器

#### 实验报告

姓名：田睿

学号17300750084

## 1 实验目的

熟悉单周期MIPS处理器的工作原理，掌握使用verilog设计硬件处理器的方法；

- ①完成MIPS单周期处理器设计。包含十四条基本指令；
- ②编写MIPS汇编测试代码测试MIPS处理器。
- ③在NEXYS4 DDR板上进行验证。

## 2 32位MIPS单周期处理器230的设计思路

### 2.1 处理器的总体结构

结合书本以及提供的参考资料，我在本次实验中主要使用模块例化的方法，将处理器的实现分散到许多模块：将显示模块与CPU核心模块分开，显示模块主要包括了七段数码管和LED的展示实现；CPU的核心模块则分成mips核心控制模块、指令存储器以及数据存储器；

最终实现的单周期处理器的层级结构及模块关系如下图所示：

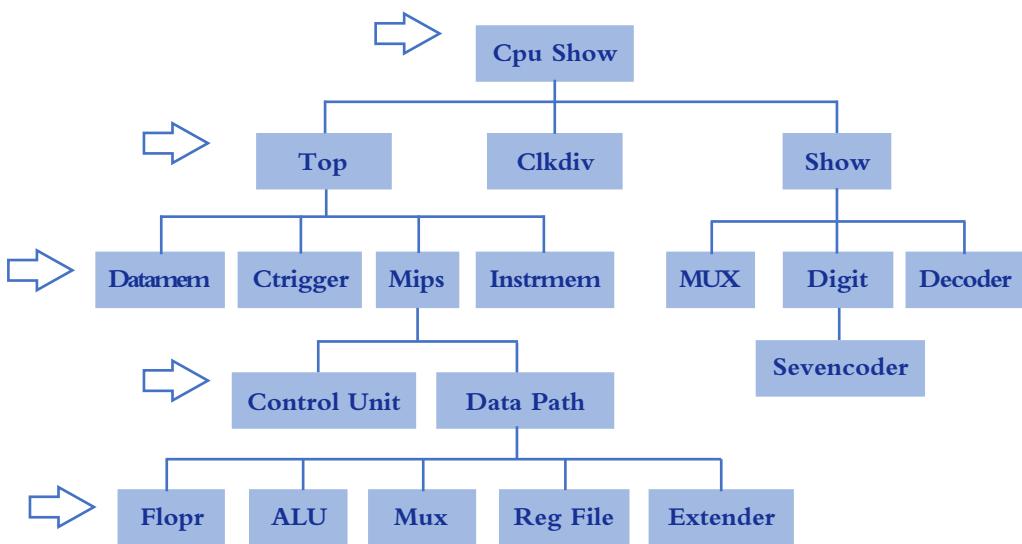


图2.1.1 处理器模块的结构层次

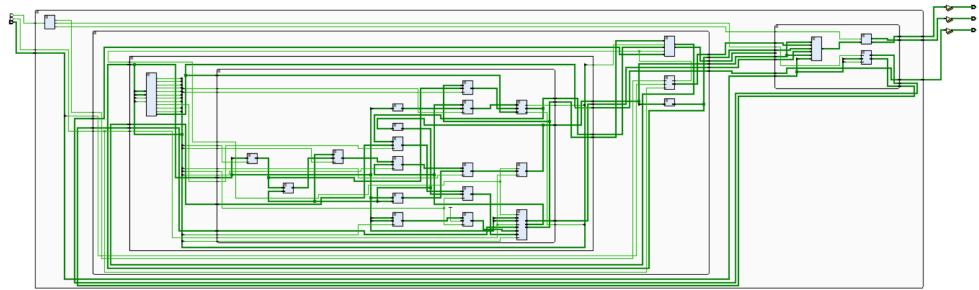


图2.1.2 处理器的硬件整体结构

## 2.2 顶层模块-Cpu\_show模块<sup>1</sup>

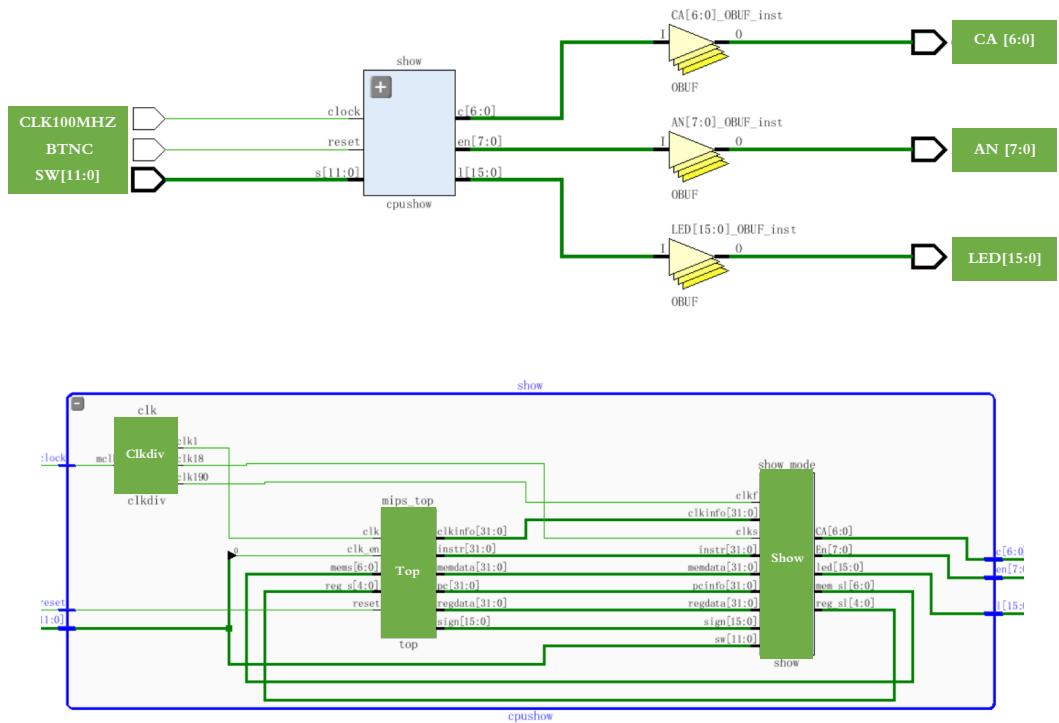


图2.2.1 cpu\_show 硬件结构

Cpu\_show作为顶层模块，链接内部核心与外部显示端口。输入输出信号全部来自或输出至NEXYS DDR板。输入CLK100MHZ的外部时钟信号，以及NEXYS DDR板的开关决定

的控制信号，以及由BTNC的reset信号，输出七段数码管的使能信号AN、显示信号CA以及LED信号；

在cpu\_show下有由clkdiv、top、show三个模块实例化的组件；

### 2.3 分频器 clkdiv

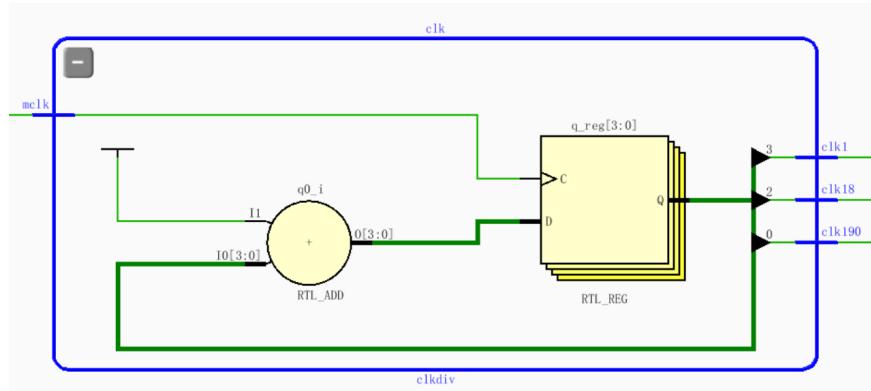


图2.3.1 clkdiv 硬件结构

clkdiv模块为分频器模块，clk信号经过clkdiv降低频率，输出三个时钟信号：clk\_cpu、clk\_c、clk\_refresh，分别作为mips的时钟信号，七段数码管数据循环模式下数据刷新的时钟信号以及为实现显示8“不同的”个数字，七段数码管的使能以及显示信号的刷新时钟信号；

### 2.4 显示模块 show

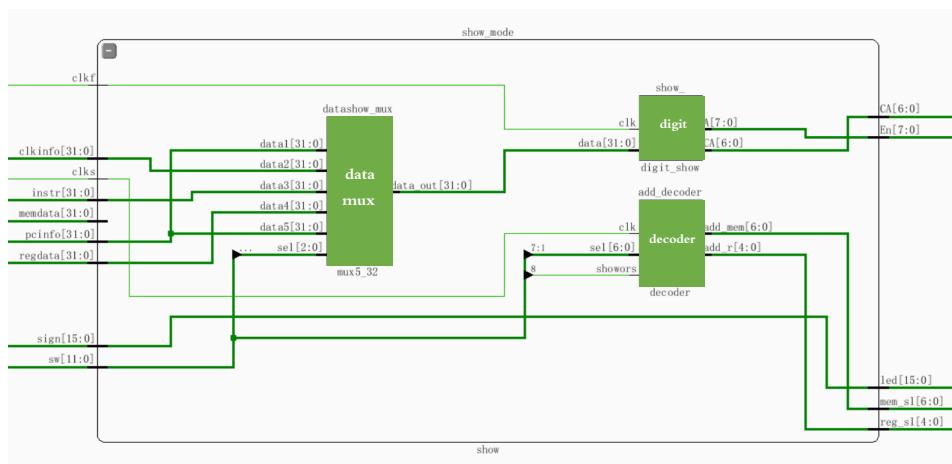


图2.4.1 show 硬件结构

`show` 模块是输出主要显示信号的枢纽，其输入信号有：`clkinfo`、`instr`、`memdata`、`pcinfo`、`regdata`，这五个信号分别为时钟信号，即mips处理器运行到的周期数、当前周期的指令、根据制定要求读取的内存地址的数据、当前的pc值、根据制定要求读取的寄存器数据，它们将作为`data_mux`的输入信号；输入信号`sign`为led显示提供驱动信号；输入控制信号`sw`则有多个用途：`sw[11:9]`则作为`data_mux`的选择信号，`sw[8:1]`是decoder的输入编码；经由decoder译码得出的内存以及寄存器地址选择信号`mem_s`、`reg_s`输出，同时与NEXYS DDR板相关联的CA、En、LED信号也是输出信号；

`show`模块下有`mux5_32`、`digit`、`decoder`三个模块实例化的组件；

对于模块`mux5_32`实例化而成的`data_mux`，`clkinfo`、`instr`、`memdata`、`pcinfo`、`regdata`是多选器的五个输入信号，而有NEXYS DDR开关控制的选择信号则确定要输出哪一个信号将输入，并且传输到`digit`模块中，作为7段数码管显示的数据；

该`data_mux`的选择规则如下：

SW[11:9]	3'b??1	3'b010	3'b100	3'b110	3'b000
输出信号	Pcinfo	Instr	Regdata	Memdata	Clockinfo

## 2.5 处理器核心模块 top

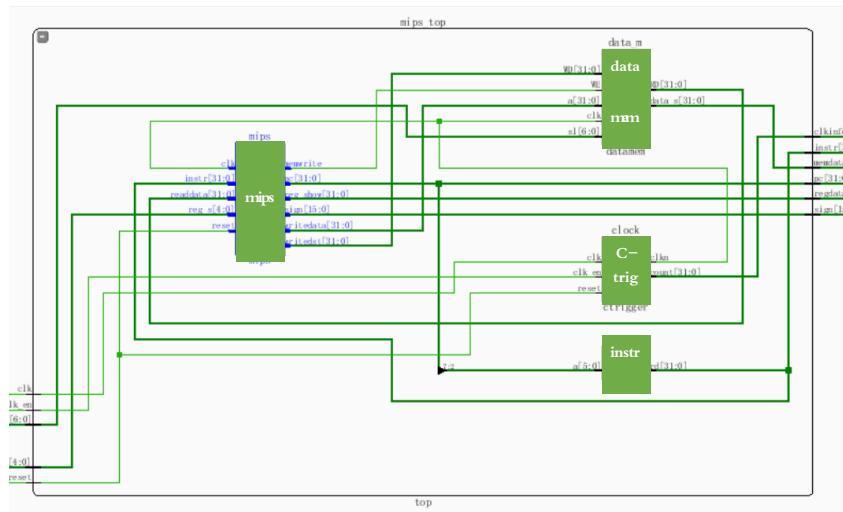


图2.5.1 top硬件结构

`top`模块中，输入时钟信号、reset清零信号以及时钟使能。这三个输入信号分别来自clkdiv、由上层模块引入的nexys ddr的开关信号；而输入的mems以及regs的地址信号则来自show模块的译码器；经由mips输出的信号是根据输入的内存地址以及寄存器地址读取的内存数据以及寄存器数据，它们将被引入show模块作为多路选择器的选择信号。

`top`模块下有由`ctrigger`、`mips`、`instrmem`、`datamem`实例化的组建。

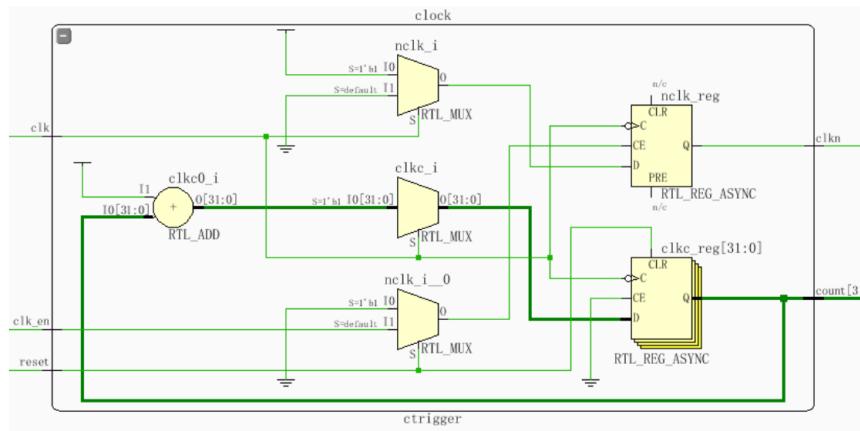


图2.5.2 `ctrigger` 硬件结构

### 1 ) `ctrigger`模块

`ctrigger`为使能时钟（运用使能信号过滤时钟信号），以便于使用开关（SW[0]）控制处理器运行进程。仅当使能信号为1时，处理器运行，时钟信息被更新。为了实现这一功能，我简单地利用了类似于分频器的原理：即当使能为1时计数存储器+1，因此，输入的时钟信号的频率为输出信号的两倍。

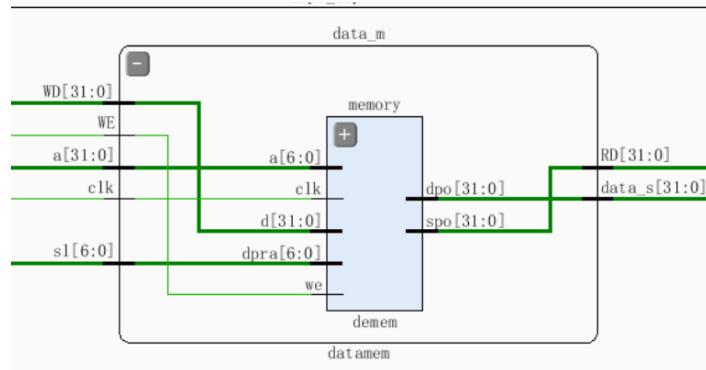


图2.5.3 `datamem`的硬件结构

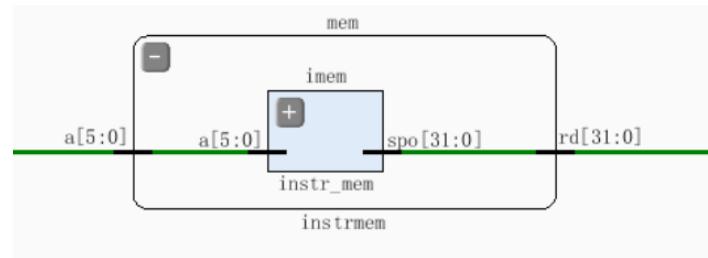


图2.5.4 instrmem的硬件结构

## 2 ) 内存存储器

**instrmem**与**datamem**的构造机制分别为单口ROM以及标准双口RAM。我利用了vivado的IP core来实现，并用.coe文件初始化内存。

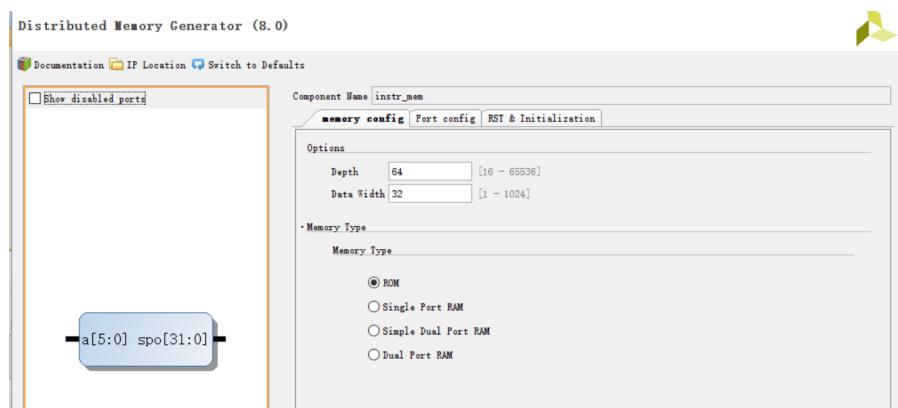


图2.5.5 instrmem 在 vivado ip中的设置

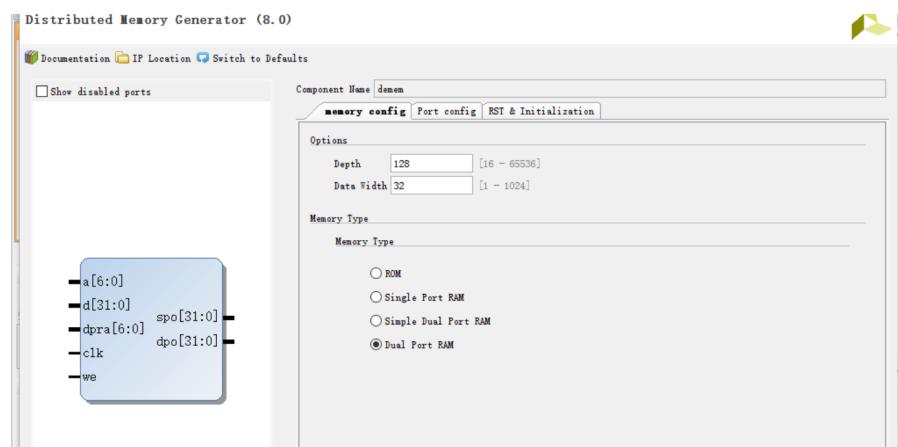


图2.5.6 datamem 在vivado中 ip的设置

由于在执行指令的过程中不需要对指令进行修改或写入，`instrmem`的设计我采用只读的ROM。而数据内存则同时需要读写的功能。值得一提的是，我选用了dual port RAM作为内存存储器。原因是我要通过外部控制实时读取内存指定地址的数据，而这一过程又不应该与处理器进程中的读写冲突。因此我加入了新的输入地址信号以及输出数据端口。这个端口只提供读功能，与写入无关。可以很好地满足显示数据的需求。

## 2.6 控制信号处理模块 control unit

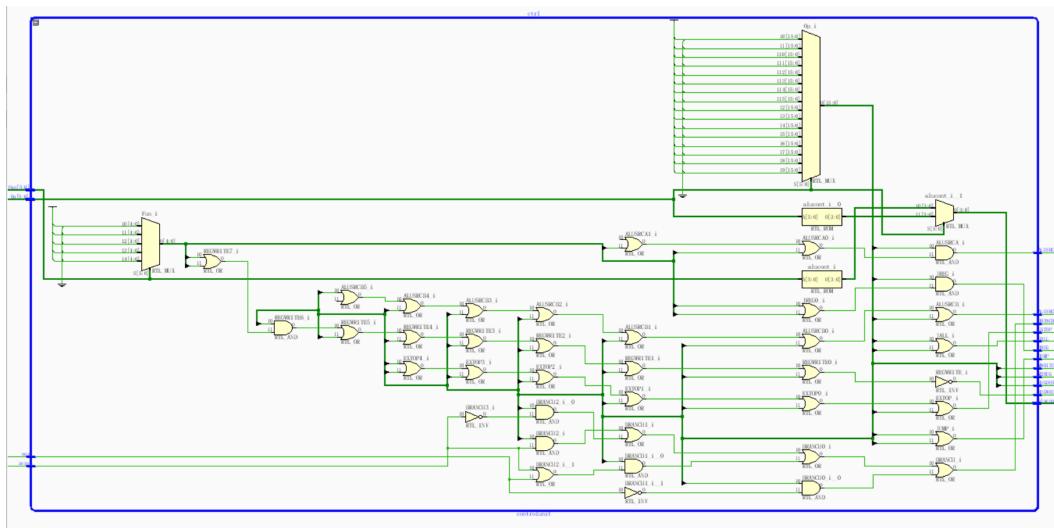


图2.6.1 control unit 硬件结构

`control unit`模块是mips处理器重要的信号处理器件。`control unit`输出的各控制信号使得数据通路上各多选器等处理相关数据。`control unit`的输出信号是完全由指令类型决定的（branch信号则需要考虑alu的运算信号）。因此仅需要考虑指令中的Op code以及Function code（以及zero以及neg标志）就可以做出相应的编码操作。下表为本处理器实现的指令在`control unit`中对应的生成信号。

	Jal	Jreg	Jump	Branch	Reg Write	Reg Dest	ALU SrcA
ADD	0	0	0	0	1	1	0
SUB	0	0	0	0	1	1	0
ADDU	0	0	0	0	1	1	0

	Jal	Jreg	Jump	Branch	Reg Write	Reg Dest	ALU SrcA
SUBU	0	0	0	0	1	1	0
OR	0	0	0	0	1	1	0
XOR	0	0	0	0	1	1	0
NOR	0	0	0	0	1	1	0
SLT	0	0	0	0	1	1	0
SLTU	0	0	0	0	1	1	0
SRL	0	0	0	0	1	1	1
SRA	0	0	0	0	1	1	1
SLL	0	0	0	0	1	1	1
SLLV	0	0	0	0	1	1	0
SRLV	0	0	0	0	1	1	0
SRAV	0	0	0	0	1	1	0
JR	0	1	0	0	0	x	x
JALR	1	1	0	0	1	x	x
ADDI	0	0	0	0	1	0	0
ANDI	0	0	0	0	1	0	0
ORI	0	0	0	0	1	0	0
XORI	0	0	0	0	1	0	0
LW	0	0	0	0	1	0	0
LUI	0	0	0	0	1	0	0
SW	0	0	0	0	0	x	0
BEQ	0	0	0	zero	0	x	0
BNE	0	0	0	!zero	0	x	0
BLEZ	0	0	0	neg zero	0	x	0
BGTZ	0	0	0	!neg	0	x	0
SLTI	0	0	0	0	1	0	0

	Jal	Jreg	Jump	Branch	Reg Write	Reg Dest	ALU SrcA
J	0	0	1	0	0	X	X
JAL	1	0	1	0	1	x	x

表2.6.2

	ALU SRCB	ALU CONT	EXTOP	MEM WRITE	MEM 2REG
ADD	0	Add	X	0	0
SUB	0	Sub	X	0	0
ADDU	0	Addu	X	0	0
SUBU	0	Subu	X	0	0
OR	0	Or	X	0	0
XOR	0	Xor	X	0	0
NOR	0	Nor	X	0	0
SLT	0	Subu	X	0	0
SLTU	0	Subu	X	0	0
SRL	0	Srl	X	0	0
SRA	0	Sra	X	0	0
SLL	0	Sll	X	0	0
SLLV	0	Sll	X	0	0
SRLV	0	Srl	X	0	0
SRAV	0	Sra	X	0	0
JR	x	X	X	0	0
JALR	x	X	X	0	0
-----	-----	-----	-----	-----	-----
ADDI	1	Add	1	0	0
ANDI	1	And	0	0	0
ORI	1	Or	0	0	0
XORI	1	Xor	0	0	0
LW	1	Addu	1	0	1
LUI	1	Lui	x	0	0

	ALU SRCB	ALU CONT	EXTOP	MEM WRITE	MEM 2REG
SW	1	Addu	1	1	0
BEQ	0	subu	1	0	0
BNE	0	Subu	1	0	0
BLEZ	0	Subu	1	0	0
BGTZ	0	Subu	1	0	0
SLTI	1	Slt	1	0	0
—	—	—	—	—	—
J	x	x	x	x	x
JAL	x	x	x	x	x

表2.6.2

## 2.7 数据通路模块 datapath

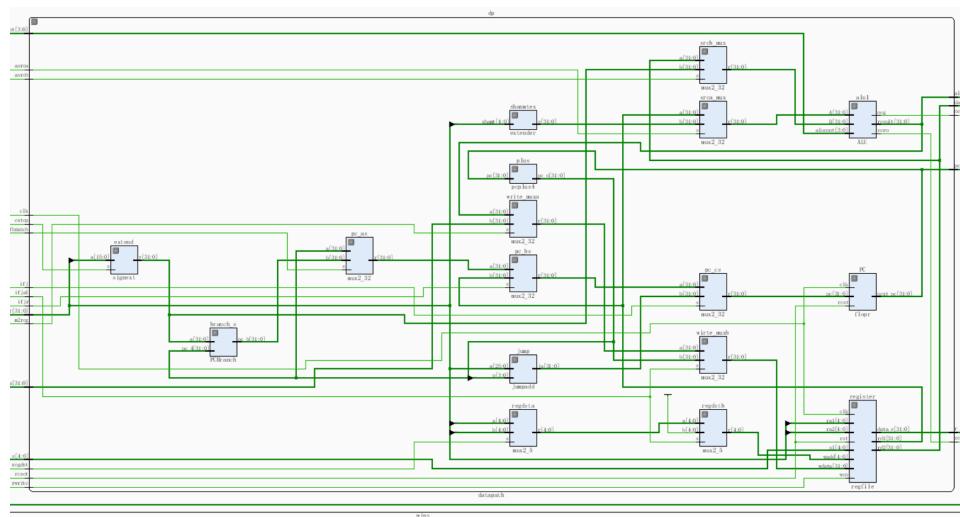


图2.7.1 datapath 硬件结构

**Datapath**模块为mips处理器的数据通路；控制信号由control unit输出后进入 datapath，由nexys ddr板开关控制的信号也传入数据通路。每条指令的运算与寄存器存取操作都在数据通路中完成。其下有模块：mux2\_5、mux2\_32、jumpadd、regfile、

alu、 flop、 extender、 pcbranch等实例化的部件。

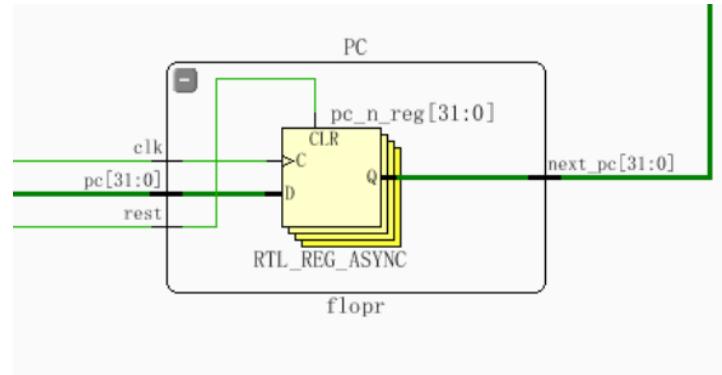


图2.7.2 flop 硬件结构

### 1 ) flop

Flop模块根据时钟信号以及清零信号来决定是否载入新的pc还是重新开始。当reset信号为高电平时，pc被置为0。输出的pc信号传入指令存储器就会重新读取最开始的数据

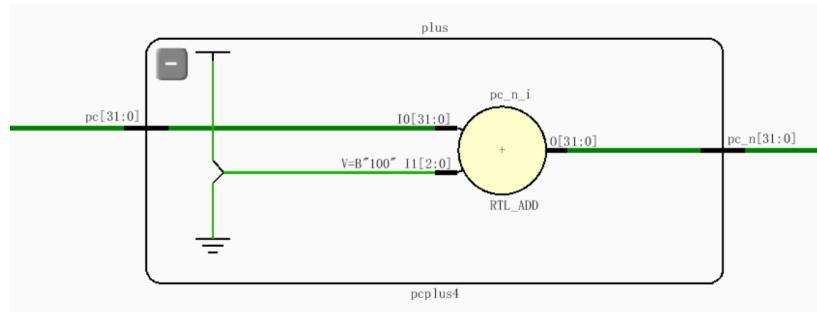


图2.7.3 pcplus4 硬件结构

### 2 ) pcplus4

pc累加器将输入的当前pc值加4输出。在不跳转的情况下 $pc + 4$ 即为下一指令的pc值。

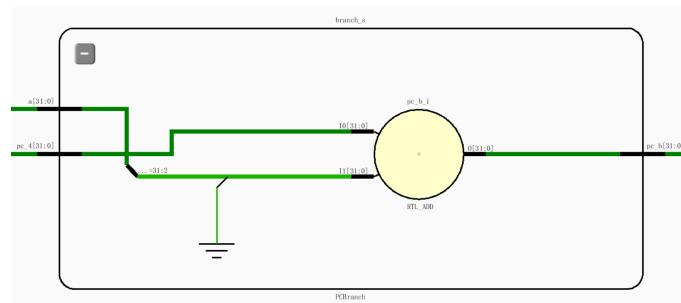


图2.7.4 pcbranch 硬件结构

### 3 ) pcbranch

在数据通路中pcbranch的功能是结合pc+4与立即数生成分支跳转的pc。

运算规则为  $pc\_b = pc\_f + im$  (偏移量)  $<< 2$

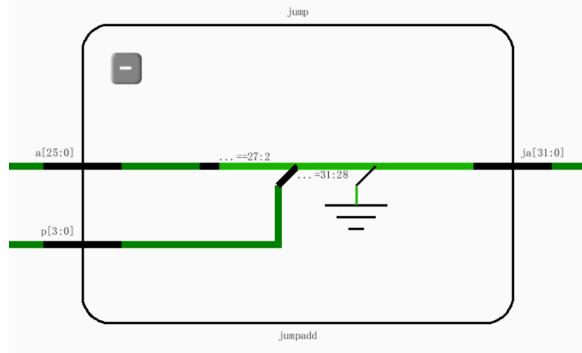


图2.7.5 jumpadd 硬件结构

### 4 ) jumpadd

直接跳转pc生成部件 **jump add** 根据指令[25:0] (jump类指令) 存储的26位地址以及pc + 4的高四位生成直接跳转的pc数据。具体的运算规则为  $pc = \{ pc + 4 [31:28], instr[25:21] << 2 \}$ 。

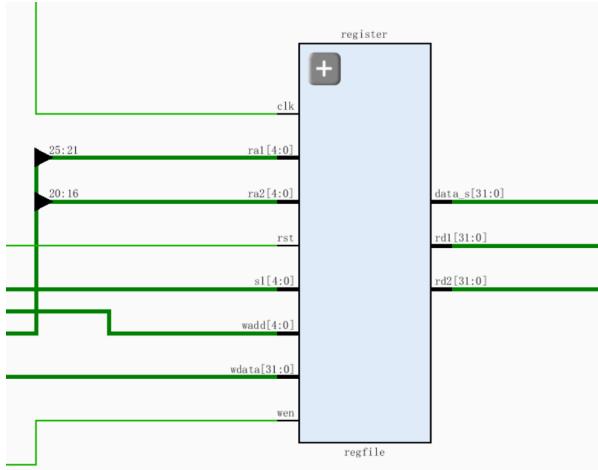


图2.7.6 regfile 硬件结构

### 5 ) 寄存器文件 regfile

在实验设计中为了简便以及准确，我选择将所有寄存器的值初设为0（实际上rsp应当指向内存最高地址，这个在初始化时设置也十分方便）。regfile我没有选择使用ip core来实现，以便用reset信号时复原。寄存器文件为宽度为32、深度也为32的寄存器

组。为了便于由外部开关控制展示寄存器数据，regfile的实现形式也为双读取端口。总共有4个输入地址信号，分别输出指令中寄存器指定的两个寄存器地址（reg类型指令）、写入寄存器地址以及nexys ddr板选择地址信号；写入寄存器的操作由使能控制；该使能信号rwrite由control unit生成

#### 6 ) mux2\_32 三个pc选择器

在datapath的模块下共有三个2-1 32-bit mux，作为pc的选择信号。

第一个多选器 pc\_as 在预测 pc 值（即顺序执行，由pcplus4输出信号提供）、branch类型指令跳转pc中选择其一，选择信号ifbranch由control unit生成。当指令信号branch为1时选择跳转pc，为0时选择顺序预测pc。输出信号 pc\_ar 作为下一个多选器的输入信号。

第二个多选器 pc\_bs 在 pc\_as 的结果与指令片段寄存器a读取的数据中选择其一，选择信号ifjr由control unit生成。当有jr类型指令时，ifjr为1，多选器输出regfile读取的寄存器a数据，直接跳转到寄存器中储存的数据。ifjr为0时，则继续pc\_as的结果。输出信号 pc\_br作为下一个多选器的输入信号。

第三个 多选器 pc\_cs 在 pc\_bs的结果与指令[25:0]字段拟合的跳转pc值中选择其一。选择信号ifj由control unit生成。当有直接跳转jump时，ifj为1，多选器选择直接跳转pc；ifj为0时，多选器继续pc\_bs的结果。pc\_cr为输出信号。将直接作为flop的输入信号。

#### 7 ) mux2\_5 寄存器文件地址多选器

在数据通路中有两个2-1 5-bit mux为regfile寄存器文件提供输入的地址信号。

第一个多选器 regdesta，输入信号分别为指令字段[20:16] 以及[25:21]，选择信号为 regdest，根据指令不同来选择需要写入的存储器地址、由control unit来提供。输出信号作为第二个多选器的输入信号。

第二个多选器 regdestb，输入信号分别为 regdesta输出信号以及常量信号31；选择信号 ifjal由control unit生成，当有jal类型指令时，ifjal为1，需要将pc + 4信号写入\$31寄存器，选择31作为输出信号（需要将pc + 4存入\$31寄存器）；如果jal为0，则继续 regdesta的输出结果。输出信号作为regfile的写入地址输入信号。

#### 8 ) mux2-32 数据多选器

在数据通路中有两个 2-1 32-bit mux，它们选择输入ALU的数据信号。

**srca\_mux** 是选择输入ALU A端口数据的多路器。它的输入信号为 shamt 扩展后的32位数据以及从寄存器文件中读取的指令[25:21]地址内存储的数据。其选择信号 srca 来自 control unit。当 srca 为1时，mux 选择 shamt 位移量，srca 为0时选取寄存器文件输出数据。

**srcb\_mux** 是选择输入 ALU B端口数据的多路器。它的输入信号为立即数经符号扩展后的32位数据以及寄存器文件中读取的指令[20:16]字段对应地址内存储的数据。其选择信号 srcb 来自 control unit。当 srcb 为1时，mux 选择立即数；当 srcb 为0时，mux 选择寄存器文件输出数据。

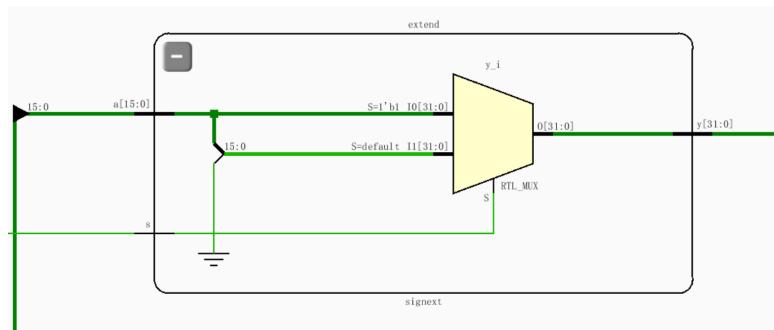


图2.7.7 signext 硬件结构

#### 9 ) signext 符号扩展

在数据通路中有一个符号扩展器，由control unit输出的extop信号作为控制信号。扩展器将16位的立即数扩展为32位。当extop为1时，按最高位符号扩展；当extop为0时，则按零高位扩充。输出的32位立即数数据将作为alu b端口输入信号之一，也将作为生成branch跳转信号的输入信号之一。

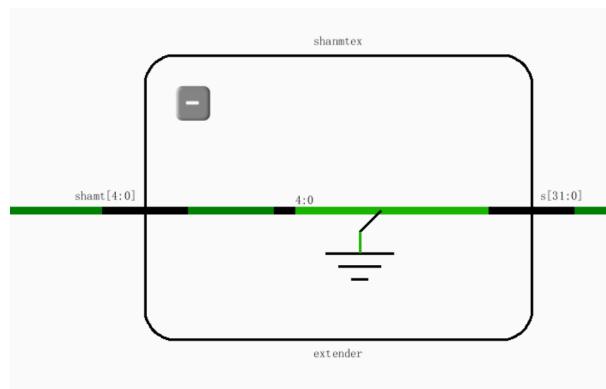


图2.7.8 extender 硬件结构

## 10 ) extender 偏移量扩展

偏移量的扩展为最高位0扩展。将5位的偏移量（所有位移指令将需要读取偏移量）扩展为32位信号。输出信号作为alu<sub>a</sub>端输入的选择信号之一。

## 11)ALU

ALU为数据通路中的运算器。其根据control unit输出的alucont信号决定对输入数据信号进行何种运算。

运算类型	ALU Select	运算
ADDU	4'b0000	$S = a + b$ (unsigned)
SUBU	4'b0001	$S = a - b$ (unsigned)
ADD	4'b0010	$S = a + b$
SUB	4'b0011	$S = a - b$
OR	4'b0100	$S = a   b$
AND	4'b0101	$S = a \& b$
XOR	4'b0110	$S = a ^ b$
NOR	4'b0111	$S = \sim(a   b)$
SLTU	4'b1000	$S = a < b ? 1: 0$ (unsigned)
SLT	4'b1001	$S = a < b ? 1 : 0$
SLL	4'b1010	$S = b << a$
SRL	4'b1011	$S = b >> a$
SRA	4'b1100	$S = b >>> a$
LUI	4'b1101	$S = \{b[15 : 0], 16'b0\}$

## 2.8 七段数码管显示模块 digit

digit 模块负责七段数码管的信号展示。它的输入信号为时钟信号（使能刷新的频率）、数据信号（显示在数码管上的32位数据信号）。输出即为nexys ddr板的使能以及CA管信号。七段数码管的原理实际上只支持同时显示一个数码管图样。为了让观察者看到32位十六进制数，我通过高频率循环刷新使能以及CA信号的方式来“同时显示”八个不同的数字。八个七段数码管的使能循环置1，不同位数的数据随着对应的数码管

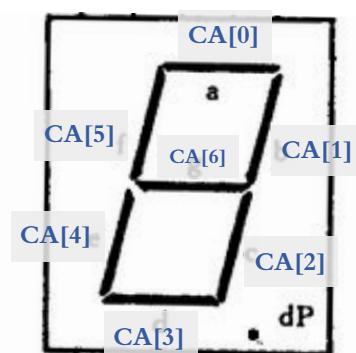
输出。由于视觉的延滞性，单独显示的不同位数字仿佛一起显示。

### sevencoder 模块

将数据信号转换为数据管的CA信号。实际就是一个译码器。对应的显示图案与数据如下。

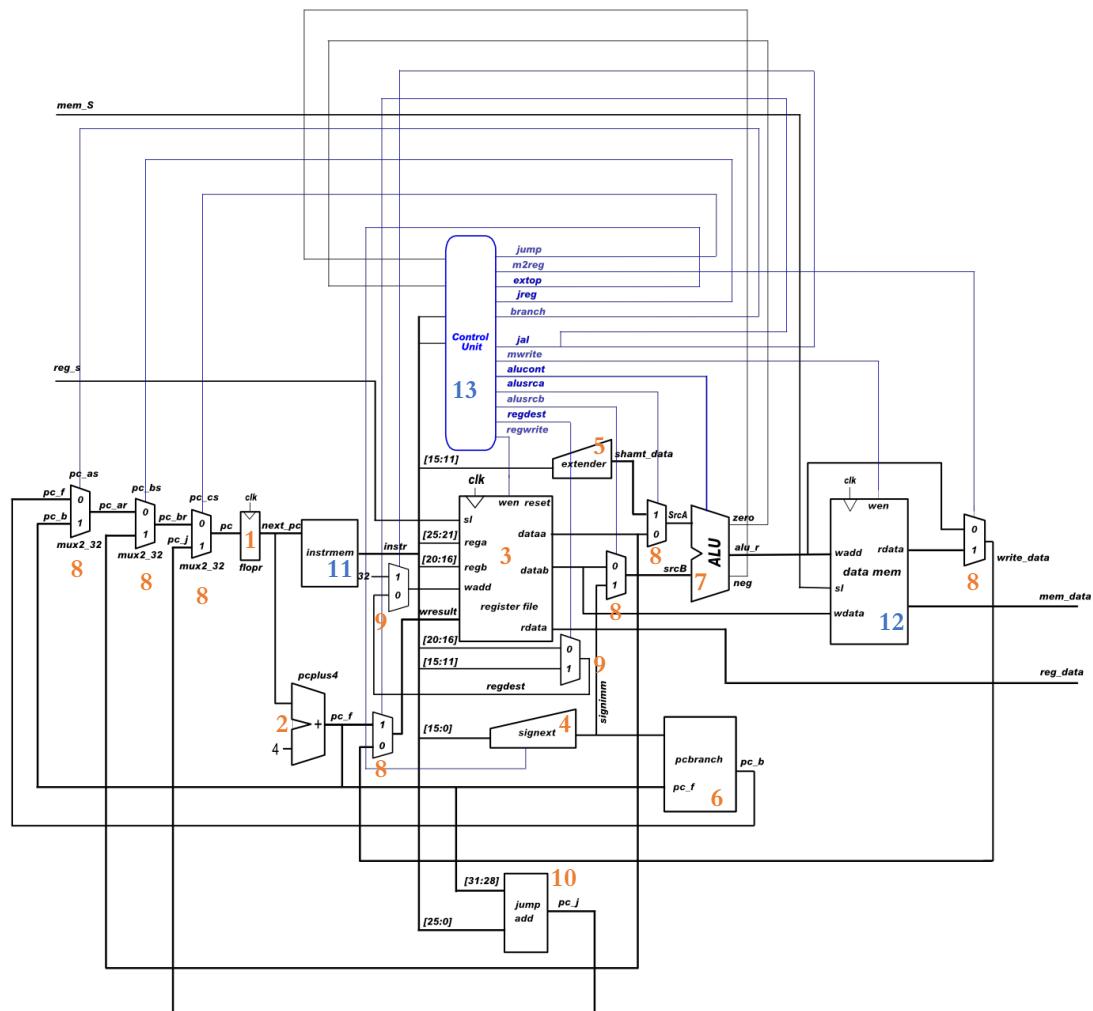
所有在七段数码管上显示的数据都为16进制。

显示数字	CA
1	7'b1111001
2	7'b0100100
3	7'b0110000
4	7'b0011001
5	7'b0010010
6	7'b0000010
7	7'b1111000
8	7'b0000000
9	7'b0011000
a	7'b0001000
b	7'b0000011
c	7'b1000110
d	7'b0100001
e	7'b0000110
f	7'b0001110
0	7'b1000000



## 2.9 数据通路整体构架的结构图

\*参考PPT绘制（图中数字对应第三部分的模块编号）



## 3 文件清单

- code

Nexys4DDR.xdc

引脚锁定文件

Sampleinstr.coe

初始化指令存储器的coe代码

### datamem.coe

初始化数据存储器的coe代码

### cup\_top.v

包括顶层模块cup\_top、分频模块clkdiv、链接模块cpushow

### digit\_show.v

包括单元模块：

mux5\_32 (8-5 32位多选器)、译码器decoder、

七段数码管显示模块sevencoder、显示模块digit\_show

### Top.v

包括处理器顶层模块top、使能时钟ctrigger、处理器核心mips

### controlUnit.v

包括datapath中所有单元化模块：

Flopr<sup>1</sup>、pcplus<sup>2</sup>、regfile<sup>3</sup>、signext<sup>4</sup>、extender<sup>5</sup>、

PCbranch<sup>6</sup>、ALU<sup>7</sup>、mux2\_32<sup>8</sup>、mux2\_5<sup>9</sup>、jumpadd<sup>10</sup>

### datapath.v

包括datapath模块及其下所有单元化模块：

instrmem<sup>11</sup>、datamem<sup>12</sup>、controlunit<sup>13</sup>

## - simulation

所有仿真代码以及其他文件存放于以下链接：

[https://github.com/Stephyuka/mips\\_32](https://github.com/Stephyuka/mips_32)

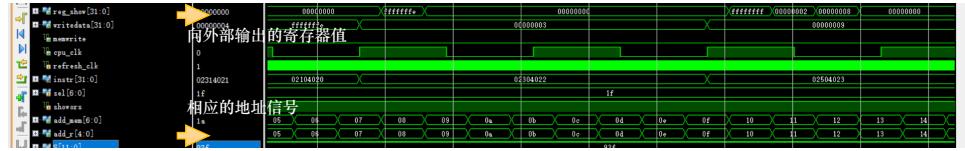
## 4 仿真实验与结果

### 4.1 顶层模块功能仿真

#### 4.1.1 循环展示开关

- 1) 当循环展示开关置0时，读取寄存器/内存数值时按照在nexys ddr板上选择的地址信号。

2) 当开关置1，则从地址 0 ~ 31 循环读取寄存器；从地址 0 ~ 127 循环读取内存；  
便于展示连续的寄存器/内存信息。



循环展示的寄存器信息



循环展示的内存信息

#### 4.1.2 选择输出信号开关

在本次实验的设计中，我选择用nexys ddr板上的SW[11] - SW[9]来选择输出到七段数码管上的数据。

##### 1) 输出寄存器的值



##### 2) 输出时钟周期数



##### 3) 输出内存数值



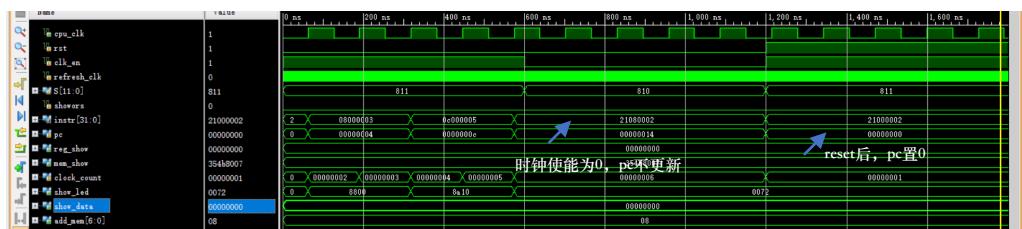
4) 输出当前指令



5) 输出当前PC值



6) 用时钟使能以及reset控制处理器进程



## 4.2 reg 类型指令仿真

#一些初始的值

\$s0 = -1; \$s1 = 2; \$s2 = 8

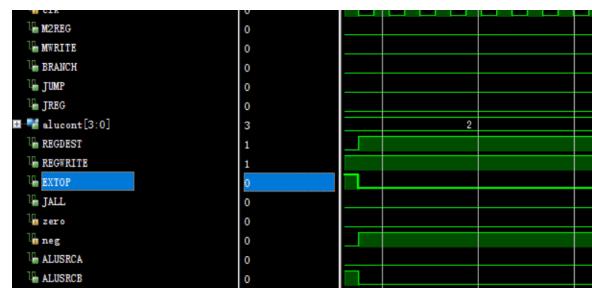
#### 4.2.2 指令 Add

语句: add \$t0, \$s0, \$s0 (\$s0 = -1)

可得: \$t0 = 0xffffffffe

instr[31:0]	02304022	02104020	02304022	X
pc	00000010	00000010	00000010	X
reg_show	ffffffffe	0000000e	0000000e	X
mem_show	354b8007	00000000	354b8007	X
clock_count	00000008	00000006	00000008	X
show_led	0113	00000007	00000009	X
show_data	ffffffffe	4112	0113	X
		00000000	ffffffffffe	

输出的\$t0的值在下一周期立刻发生改变



相应的control unit信号

#### 4.2.3 指令 Sub

语句: sub \$t0, \$s1, \$s0

\$t0 = 0x3

instr[31:0]	02304022	02304022	X	02504023	X
pc	00000010	00000010	X	00000014	X
reg_show	ffffffffe	ffffffffffe	X	00000003	X
mem_show	354b8007	354b8007			
clock_count	00000008	00000008	X	00000009	X
show_led	0113	0113	X	0111	X
show_data	ffffffffe	ffffffffffe	X	00000003	X

#### 4.2.4 指令 Subu

语句: subu \$t0, \$s2, \$s0

\$t0 = 0x9

instr[31:0]	02304022	02504023	X	02314021	
pc	00000010	00000014	X	00000018	
reg_show	ffffffffe	00000003	X	00000009	
mem_show	354b8007	354b8007			
clock_count	00000008	00000008	X	00000009	X
show_led	0113	0111	X	0110	X
show_data	ffffffffe	00000003	X	00000009	

#### 4.2.5 Addu

语句: addu \$t0, \$s1, \$s1

$$\$t0 = 0x4$$

instr[31:0]	02304022	02314021	X	02514027
pc	00000010	00000018	X	0000001c
reg_show	fffffff8	00000009	X	00000004
mem_show	354b8007			354b8007
clock_count	00000008	0000000c	X	0000000d
show_led	0113	0110	X	4117
show_data	fffffff8	00000009	X	00000004

#### 4.2.6 Nor

语句: nor \$t0 \$s2 \$s1

$$\$t0 = 0xffffffff$$

instr[31:0]	02304022	02514027	X	02514028
pc	00000010	0000001c	X	00000024
reg_show	fffffff8	00000004	X	ffffffff
mem_show	354b8007			354b8007
clock_count	00000008	0000000*	X	00000010
show_led	0113	4117	X	0114
show_data	fffffff8	00000004	X	ffffffff

#### 4.2.7 Or

语句: or \$t0 \$s2 \$s1

$$\$t0 = 0xa$$

instr[31:0]	02304022	02514025	X	02284026
pc	00000010	00000020	X	00000024
reg_show	fffffff8	fffffff5	X	0000000a
mem_show	354b8007			354b8007
clock_count	00000008	00000010	X	00000012
show_led	0113	0114	X	0116
show_data	fffffff8	fffffff5	X	0000000a

#### 4.2.8 Xor

语句: xor \$t0 \$s1 \$t0

$$\$t0 = 0x8$$

instr[31:0]	02304022	0	X	02284026
pc	00000010	0	X	00000028
reg_show	fffffff8	f	X	0000000a
mem_show	354b8007			354b8007
clock_count	00000008	00000012	X	00000013
show_led	0113	0	X	0116
show_data	fffffff8	f	X	0000000a

#### 4.2.9 Slt

语句: slt \$t0 \$s1 \$s2

$$\$t0 = 0x1$$

instr[31:0]	02304022	02	X	02324024	X	0251402B	
pc	00000010	00	X	00000028	X	0000002C	
reg_show	ffffffffff	00	X	00000008	X	00000001	
mem_show	354b8007					354b8007	
clock_count	00000008	00	X	00000014	X	00000015	
show_led	0113	0116	X	0119	X	8118	
show_data	ffffffffff	00	X	00000008	X	00000001	

#### 4.2.10 Sltu

语句: sltu \$t0 \$s2 \$s1

$$\$t0 = 0x0$$

instr[31:0]	00114040	02	X	0251402B		00114040	
pc	00000030	00	X	0000002C		00000030	
reg_show	00000004	00	X	00000001		00000000	
mem_show	354b8007					354b8007	
clock_count	00000018	00	X	00000016	X	00000017	
show_led	019a	0119	X	8118	X	019a	
show_data	00000000	00	X	00000001		00000000	

#### 4.2.11 Sll

语句: sll \$t0 \$s1 1

$$\$t0 = 0x4$$

instr[31:0]	02324004	025	X	00114040		02324004	
pc	00000034	000	X	00000030		00000034	
reg_show	00000004	000	X	00000000		00000004	
mem_show	354b8007					354b8007	
clock_count	0000001a	000	X	00000018	X	00000019	
show_led	011a	0119	X	8118	X	011a	
show_data	00000004	000	X	00000000		00000004	

#### 4.2.12 Sllv

语句: sllv \$t0 \$s1 \$s2

$$\$t0 = 0x20$$

instr[31:0]	02324004	023	X	00114040		00114082	
pc	00000034	000	X	00000034		00000038	
reg_show	00000004	000	X	00000004		00000020	
mem_show	354b8007					354b8007	
clock_count	0000001a	00000001a	X	0000001b	X	0000001c	X
show_led	011a	011a	X			019b	
show_data	00000004	00000004	X			00000020	

#### 4.2.13 Srl

语句: srl \$t0 \$s0 2

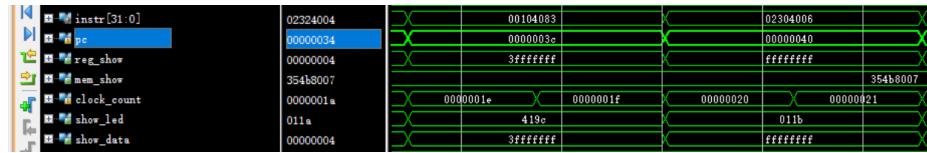
$$\$t0 = 0x3fffffff$$

instr[31:0]	02324004	X	00114082	X	00104083		
pc	00000034	X	00000038	X	0000003c		
reg_show	00000004	X	00000020	X	3fffffff		
mem_show	354b8007					354b8007	
clock_count	0000001a	X	0000001c	X	0000001e	X	0000001f
show_led	011a	X	019b	X	419c		
show_data	00000004	X	00000020	X	3fffffff		

#### 4.2.14 Sra

语句: sra \$t0 \$s0 2

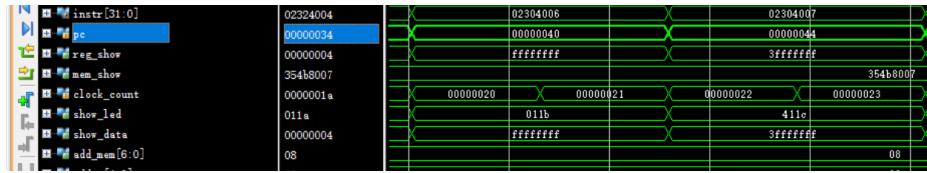
$$\$t0 = 0xffffffff$$



#### 4.2.16 Srlv

语句: srlv \$t0 \$s0 \$s1

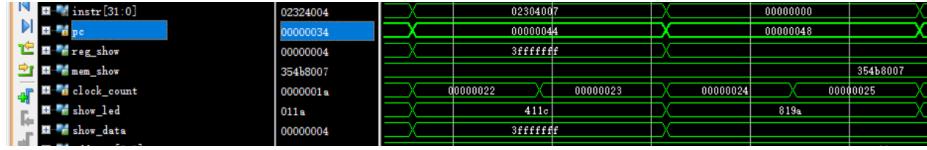
$$\$t0 = 0x3fffffff$$



#### 4.2.17 Srav

语句: srav \$t0 \$s0 \$s1

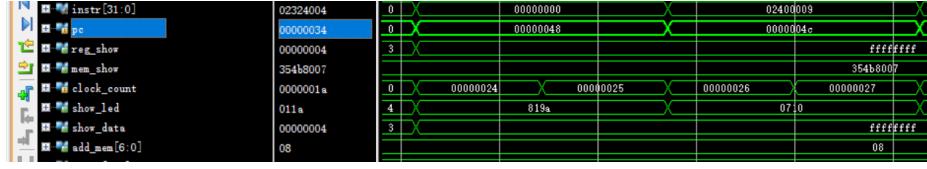
$$\$t0 = 0xffffffff$$



#### 4.2.18 nop

语句: nop

$$\$t0 = 0xffffffff$$



#### 4.2.18 Jr

语句: jr \$s2

Jump to 0x08

instr[31:0]	02324004	00	02400009	X	20120008	X
pc	00000034	00	0000004c	X	00000008	X
reg_show	00000004				fffffff	
mem_show	354b8007					
clock_count	0000001a	00	00000026	00000027	X	00000028
show_led	011a	019a	0710	X	0072	00000029
show_data	00000004				fffffff	

### 4.3 Imm 类型指令仿真

#### 4.3.1 Addi

addi \$t0, \$0, 16

addi \$s1, \$0, -4

addi \$s0, \$0, 2

instr[31:0]	3508ffff	X	3e08001f			
pc	00000010	X	0000000c			
reg_show	001f0000		00000010			
mem_show	354b8007					
clock_count	00000008	X	00000006	X	00000007	
show_led	0054	X	0054	X	0054	
show_data	001f0000		00000010			

#### 4.3.2 Lui

lui \$t0 31

\$t0 = 1f0000

instr[31:0]	3508ffff	X	3e08001f		3508ffff	
pc	00000010	X	0000000c		00000010	
reg_show	001f0000		00000010		001f0000	
mem_show	354b8007					
clock_count	00000008	X	00000006	X	00000007	
show_led	0054	X	0054	X	0054	
show_data	001f0000		00000010		001f0000	

#### 4.3.3 Ori

ori \$t0, \$t0, -1

\$t0 = 1fffff

instr[31:0]	3508ffff	X	3508ffff	X	39080003	X
pc	00000010	X	00000010	X	00000014	X
reg_show	001f0000		001f0000	X	00ffff	X
mem_show	354b8007					
clock_count	00000008	X	00000008	X	00000009	
show_led	0054	X	0054	X	0056	
show_data	001f0000		001f0000	X	00ffff	X

#### 4.3.4 Xori

xori \$t0, \$t0, 3

\$t0 = 1fffffc

<b>instr[31:0]</b>	3508ffff	X	39080003	X	a100002	X
<b>pc</b>	00000010	X	00000014	X	00000018	
<b>reg_show</b>	001f0000	X	00fffffe	X	001fffff	
<b>mem_show</b>	354b8007				354b8007	
<b>clock_count</b>	00000008	X	0000000a	X	0000000c	X
<b>show_led</b>	0054	X	0056	X	2060	
<b>show_data</b>	001f0000	X	00fffffe	X	001fffff	

#### 4.3.5 SW

sw \$s0 2(\$s0)

mem[4] = 2

#### 4.3.6 LW

lw \$t0 2(\$s0)

\$t0 = 2

<b>instr[31:0]</b>	3508ffff	X	3e080002	X	1110000	X
<b>pc</b>	00000010	X	0000001e	X	00000020	
<b>reg_show</b>	001f0000	X	00fffffe	X		
<b>mem_show</b>	354b8007				354b8007	
<b>clock_count</b>	00000008	X	0000000e	X	00000010	X
<b>show_led</b>	0054	X	0070	X	9021	
<b>show_data</b>	001f0000	X	00fffffe	X	00000002	

#### 4.3.7 beq

0x20: beq \$t0, \$s0, 0x28

Jump to 0x28

0x28: beq \$t0, \$s1, 0x24

Go down to 0x2c

<b>instr[31:0]</b>	3508ffff	X	1110000	X	1111ffff	X
<b>pc</b>	00000010	X	00000029	X	00000028	
<b>reg_show</b>	001f0000	X			00000002	
<b>mem_show</b>	354b8007				354b8007	
<b>clock_count</b>	00000008	X	00000010	X	00000012	X
<b>show_led</b>	0054	X	9021	X	0021	
<b>show_data</b>	001f0000	X			00000002	

<b>instr[31:0]</b>	3508ffff	1	X	1111ffff	X	1510ffff	X
<b>pc</b>	00000010	0	X	00000028	X	0000002c	
<b>reg_show</b>	001f0000	X				00000002	
<b>mem_show</b>	354b8007					354b8007	
<b>clock_count</b>	00000008	0	X	00000012	X	00000014	X
<b>show_led</b>	0054	9	X	00000013	X	8021	
<b>show_data</b>	001f0000					00000002	

#### 4.3.8 bne

0x2c: bne \$t0, \$s0, 0x28

Go down to 0x30

0x30: bne \$s1, \$t0, 0x38

Jump to 0x38

showers	0	11	1510ffff	X	16280001	
instr[31:0]	3508ffff	00	0000002c	X	00000030	
pc	00000010					00000002
reg_show	001f0000					354b8007
mem_show	354b8007					
clock_count	00000008	00	00000014	X	00000015	X 00000017
show_led	0054	0021	8023	X	5021	
show_data	001f0000					00000002

showers	0	151	16280001	X	1d10001	X
instr[31:0]	3508ffff	000	00000030	X	00000038	X
pc	00000010					00000002
reg_show	001f0000					354b8007
mem_show	354b8007					
clock_count	00000008	000	00000016	X	00000017	X 00000019
show_led	0054	8021	5021	X	1d21	
show_data	001f0000					00000002

#### 4.3.9 bgz

0x38: bgz \$t0, \$s1, 0x40

Jump to 0x40

0x40: bgz \$s1,t0, 0x48

Go down to 0x44

showers	0	162	1d10001	X	1e300001	X
instr[31:0]	3508ffff	000	00000038	X	00000040	X
pc	00000010					00000002
reg_show	001f0000					354b8007
mem_show	354b8007					
clock_count	00000008	000	00000018	X	00000019	X 0000001a
show_led	0054	5021	1d21	X	4021	
show_data	001f0000					00000002

showers	0	X	1e300001	X	00000000	X
instr[31:0]	3508ffff	X	00000040	X	00000044	X
pc	00000010					00000002
reg_show	001f0000					354b8007
mem_show	354b8007					
clock_count	00000008	X	0000001a	X	0000001b	X 0000001c
show_led	0054	X	4021	X	819a	
show_data	001f0000					00000002

#### 4.3.10 slti

slti \$t0 \$s0 -1

\$t0 = 0

showers	0	0	2a0fffff	X	19100001	
instr[31:0]	3508ffff	0	00000048	X	0000004c	
pc	00000010					00000002
reg_show	001f0000		00000002			354b8007
mem_show	354b8007					
clock_count	00000008	0	0000001e	X	0000001f	X 00000020
show_led	0054	8	8079	X	5021	
show_data	001f0000		00000002			

#### 4.3.11 blez

blez \$t0, \$s0, 0x54

Jump to 0x54

instr[31:0]	3508ffff	2k	19100001	X		
pc	00000010	00	0000004c	X	00000054	
reg_show	001f0000	00			00000040	
mem_show	354b8007				354b8007	
clock_count	00000008	00	00000020	X	00000021	X
show_led	0054	8079	5021	X	00000022	X
					00000023	

## 4.4 jump 类型指令仿真

### 4.4.1 j

j 0x0c

instr[31:0]	00000000	X	08000003	X	0e000005	
pc	0000001c	X	00000004	X	0000000c	
reg_show	00000010		00000000			
mem_show	00000000				00000000	
clock_count	0000000a	X	00000002	X	00000003	X
show_led	819a		8800		00000004	X
show_data	00000010				00000005	
add_mem[6:0]	1f				8+10	
						1f

### 4.4.2 jal

Jal 0x14

\$31 = 0x10

instr[31:0]	21080002	X	0e000005	X	20000002	X	
pc	00000014	X	0000000e	X	00000014	X	
reg_show	00000010	00000000	X		00000010		0000001c
mem_show	00000000						
clock_count	00000006	X	00000004	X	00000005	X	
show_led	0072		00000005	X	00000006	X	
show_data	00000010		00000006	X	00000007	X	
add_mem[6:0]	1f				00000008	X	00000009

## 5 申A理由

### 5.1 处理器操作友好性

- 设计时钟使能信号

通过时钟信号，可以暂停处理器，使得显示板上数据便于查看。

具体实现见2.5. (1)

- 设计循环查看开关

对于内存以及存储器的数据查看，不仅可以通过开关选择相应地址；

还可以通过开启循环，顺序自动播放内存、寄存器内的数据。便于验证一些指令运算的正确性（如排序）

其实现方式为：

show模块中有子模块 decoder，decoder 通过信号showors判断输出不同的数据；

当showors为1时，没到达时钟上升沿（这个时钟并非处理器时钟），计数器加一，

再将计数器数据转化输出变化的地址信号，并传入mips，由组合逻辑电路输出（几

乎) 同步变化的数据。

当showors为0时，则直接译码输出传入的地址信号；

- 支持切换查询不同数据

通过外部switch控制，数码管可以显示PC、指令、内存、寄存器、时钟周期数等不同的数据。十分便于用户使用。具体呈现见 4.1.2

## 5.2 指令集及数据通路的扩充

将指令扩充至31条，并于数据通路中添加相应的部件，具体内容见 2·32位MIPS单周期处理器231的设计思路。