Single-Cycle MIPS Processor 实验报告

16307130194 陈中钰

16 级 计算机科学技术学院

Contents

1 总体状况

1.1 单周期 MIPS 处理器

在一个周期内执行一条完整的指令。结构易于解释且控制单元简单，不需要其他非体系结构状态。时钟周期是由最慢的指令决定的。

1.2 指令集

1. 书中结构已包含的指令：add，sub，and，or，slt，addi，sw，lw，beq
2. 要求以内添加指令：bne，j，nop，andi，ori，slti
3. **要求以外添加指令：xor，xori，nor，sll，srl，sra，jal，lui**

（以上指令的实现完全按照 MIPS 指令集文档中的格式，故不再附上指令格式要求。）

1.3 规格

regfile: 32bit*32
data RAM: 32bit*64
instr RAM: 32bit*64

1.4 实现功能（均已向陈老师展示）（会在仿真中对部分功能进行验证）

1. 可以屏蔽时钟，实现**暂停**功能（cloak）
2. 可在暂停的情况下，实现**单条指令运行**功能（next）
3. 可以同步使 **PC 归零**（reset）
4. 可以同步使 **dmem、regfile 归零**（clear）
5. 由于 reset、clear 功能，**可以在程序运行结束后，直接重新开始运行程序**
6. 可以通过 16 个 LED 灯、8 个 7 段数码，**直观地查看**电路中**所有的值**

1.5 总体结构

1. 代码结构

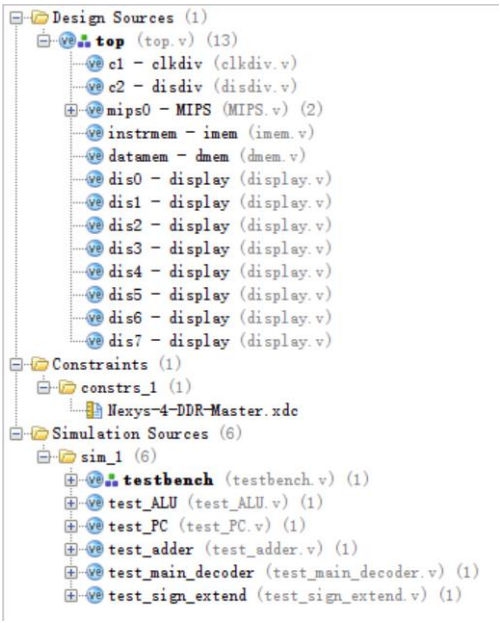

Figure 1 总体代码结构

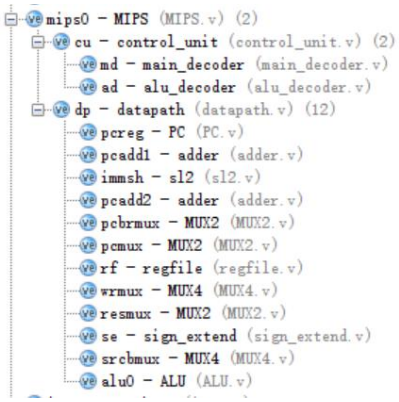

Figure 2 MIPS 模块结构

其中，testbench 为处理器的仿真文件，另外还有部分部件有对应的仿真文件。

2. 硬件结构

由于添加了 **xor，xori，nor，sll，srl，sra，jal，lui 共 8 条要求以外指令，** 在要求的基础上，增加/拓宽了一些控制信号，在后面会详细介绍。

1.6 测试代码

位于

\\MIPS_single_cycle_32bit\MIPS_single_cycle_32bit.srcs\test_files\

文件夹中，含有 **memfile0~6（其中 memfile6 含有所有指令）、sumfile0（1~100 的求和）、sortfile0（插入排序）共 9 份测试代码**。每份测试代码有.dat、.txt 为后缀的两份同名文件。其中，**.dat 为十六进制代码**，用于输入单周期处理器中运行；而**.txt 文件为完整的代码说明**，含有汇编代码、代码描述、地址、十六进制码等内容，方便检验代码、处理器的正确性。

```
1  #       Assembly            Description             Address     Machine     Binary
2  main    addi $t0, $0, 80    # t0=80=0x50            0           20060050    001000 00000 00110
3          addi $t1, $0, 8     # t1=8                  4           20010008
4
5          addi $8, $0, 8      # $8=8                  8           20080008
6          sw   $8, 0($t0)     # [80] = 8              c           acc80000    101011 00110 01000
7          addi $8, $0, 4      # $8=4                  10          20080004
8          sw   $8, 4($t0)     # [80] = 4              14          acc80004
9          addi $8, $0, 2      # $8=2                  18          20080002
10         sw   $8, 8($t0)     # [80] = 2              1c          acc80008
11         addi $8, $0, 7      # $8=7                  20          20080007
12         sw   $8, 12($t0)    # [80] = 7              24          acc8000c
```

Figure 3 sortfile0.txt 部分示例

```
1  20060050
2  20010008
3  20080008
4  acc80000
5  20080004
6  acc80004
7  20080002
8  acc80008
9  20080007
10 acc8000c
11 20080003
```

Figure 4 sortfile.dat 部分示例

1.7 文件读入地址修改（以下文件均可在 test_file 文件夹中可以找到）

在 imem 模块，需把如下地址修改为执行代码文件的实际地址。

```
33  //initialize instr memory
34  initial
35  $readmemh("C:/Users/ECHOES/Desktop/single_cycle_pj/MIPS_single_cycle_32bit/MIPS_single_cycle_32bit.srcs/test_files/sortfile0
```

在 dmem 模块，需要把如下地址修改为 emptyRAM.dat 的实际地址。

```
43  //clear dataRAM
44  always@(posedge clk)
45  if(clear)
46      $readmemh("C:/Users/ECHOES/Desktop/single_cycle_pj/MIPS_single_cycle_32bit/MIPS_single_cycle_32bit.sr
```

在 regfile 模块，需要把如下地址修改为 emptyreg.dat 的实际地址。

```
61  always@(posedge clk)
62  if(clear)
63      $readmemh("C:/Users/ECHOES/Desktop/single_cycle_pj/MIPS_single_cycle_32bit/MIPS_single_cycle_32bit.src
```

2 显示实现

（代码实现请看工程文件，在此不展示）

2.1disdiv

这是用于七段数码管扫描显示用的分频模块。在每次数次 input clk 的上升沿时，对 reg 向量 q 进行+1 的操作，另外把 q[x]赋值给 output clk。其中，如果 x 越大则 output clk 的频率越小，而 x 越小则频率越大、越接近 input clk 的频率。

由于开发板在任意时刻，只能显示一个 7 段数码，故需要利用人的视觉残留，在适当的频率下使每个 7 段数码轮流显示其对应值，可以产生 8 个 7 段数码同时显示的效果。在这里取 x=17，能稳定显示数字。如果 x 偏大则频率过小，会看到 8 个 7 段数码轮流显示，如果 x 偏小则频率过高，会导致明暗不一、闪烁、部分不显示、显示错误数字等错觉。

2.2display

传入一个 4bit 的数 T。如**控制 s 为 1，则将 T 转化为十六进制（0~F）的 7 段数码显示**，并记录在 7bit 的 C 中；如 **s 为 0，则转化为 - 符号进行显示**。

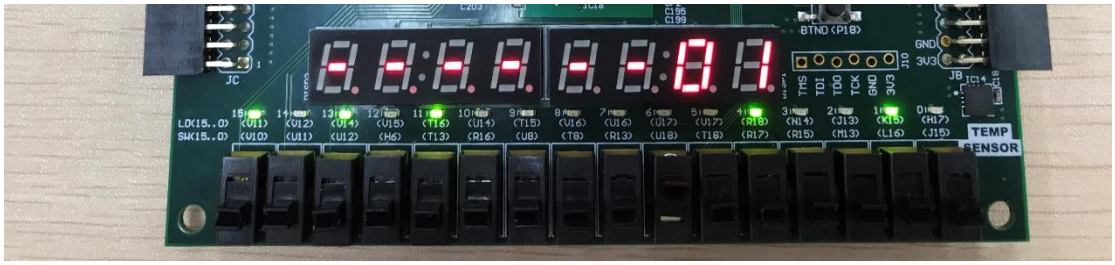

Figure 5 7 段数码显示示例

2.3top 及约束文件

top 为顶层模块，在这里介绍其中用于显示的部分。

2.3.1 输入及调控

1. next-SW[15]：在暂停的情况下，上升沿时，运行到下一条指令
2. cloak-SW[14]：为 1 时，屏蔽时钟，暂停
3. reset-SW[13]：为 1 时，同步 PC 归零
4. clear-SW[12]：为 1 时，同步 dmem、regfile 归零（如果 clear 与 reset 同时使用，可以把程序运行状态调整为一开始时的状态，并可以重新运行程序）
5. clk-CLK100MHZ：输入时钟，用分频模块产生其他所需频率的时钟
6. 输出模块选择-SW[9:6] + 输出值选择-SW[5:0]：控制 8 个 7 段数码管，按照对

应数据所需位数输出对应的十六进制值，多余的位输出 - 符号。其他没有对应的，则输出 8 个 - 符号。

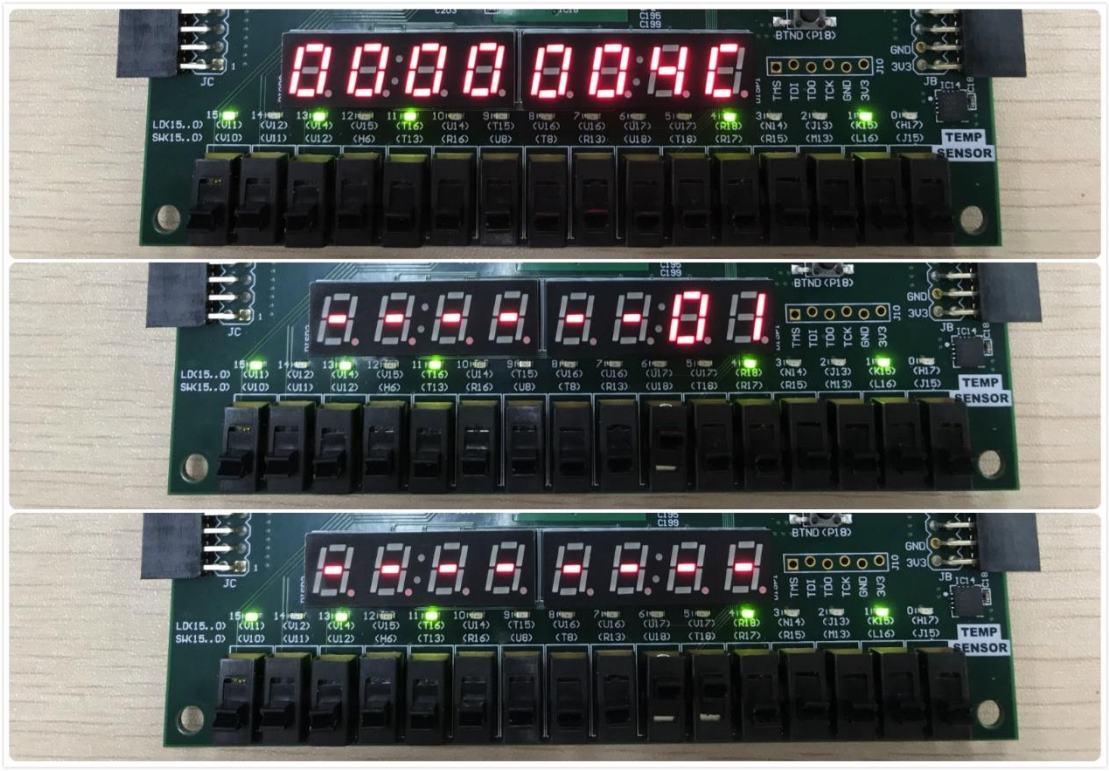

Figure 6 7 段数码管显示示例

2.3.2 输出及显示机制

1. **discu[15:0]-LED[15:0]**:由于 control unit 的输出的调控信号很重要，且大部分位长为 1，所以用 LED 灯显示。在 control unit 模块中添加 output [15:0] discu，**并把全部 signals 赋值到 discu 上，最终在 LED 中显示**。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
regwrite	regdst		alusrca		alusrcb		branch	memwrite	mem2reg	jump	extend	alucont			

Figure 7 LED 显示对应关系

2. C[6:0]:在某一时刻需要输出的 7 段数码数字
3. AN[7:0]:指定 8 个中需要点亮的 7 段数字
4. **8 个 7 段数码管取值机制**：把选择显示模块的 SW[9:6]、选择显示值的 SW[5:0]传入到 MIPS 中的 datapath 模块、dmem 模块、imem 模块，同时向上述模块各传入一对 tx[31:0]、disopx[7:0]值，通过 SW 来选择要显示的值，并加载到 tx 值中，而 disop 则记录要显示的位数（0~8 位）。最后在 top 模块中，按照 SW 值，选取最终要显示的 tx、disopx 值，并传给 top 模块中的 T、disop。另外，SW 产生的多余的选项，则对 T、disop 传入 0 值，也就是显示 0 位数。

SW[9:6]	模块	SW[5:0]	值	位数（0~8 位）
0000	PC 类	000000	pc	8
		000001	pcnext	8

		000010	pcplus4	8
		000011	signimm	8
		000100	signimmsh	8
		000101	pcbranch	8
		000110	pcnextbr	8
		000111	pcsrc	1
		else	0	0
0001	reg 类	000000	writereg	2
		000001	result	8
		000010	srca	8
		000011	writedata	8
		else	0	0
0010	regfile	000000~ 011111	reg[SW[4:0]]	8
		else	0	0
0011	alu 类	000000	srca	8
		000001	srcb	8
		000010	aluout	8
		000011	zero	1
		else	0	0
0100	imem	000000~ 111111	RAM[SW[5:0]] (RAM in imem module)	8
0101	imem 类	000000	addr	2
		000001	instr	8
		else	0	0
1000	dmem	000000~ 111111	RAM[SW[5:0]] (RAM in dmem module)	8
1001	dmem 类	000000	addr	8
		000001	writedata	8
		000010	readdata	8
		else	0	0
else			0	0

Figure 7 开关取值对应关系

5. **8 个 7 段数码管显示机制**：把 32bit 的 T 分开成 8 个十六进制数，分别用 display 模块进行转换，并记录到 C0~C7 中，多余的位会被转化为 – 符号。接着采用合适的频率，扫描显示 8 个数码管，最终会显示出 8 位数。

2.3.3 输入输出描述

通过上述输入输出，可以**直观地、简易地实现以上功能，并能查看处理器中的所有值**。

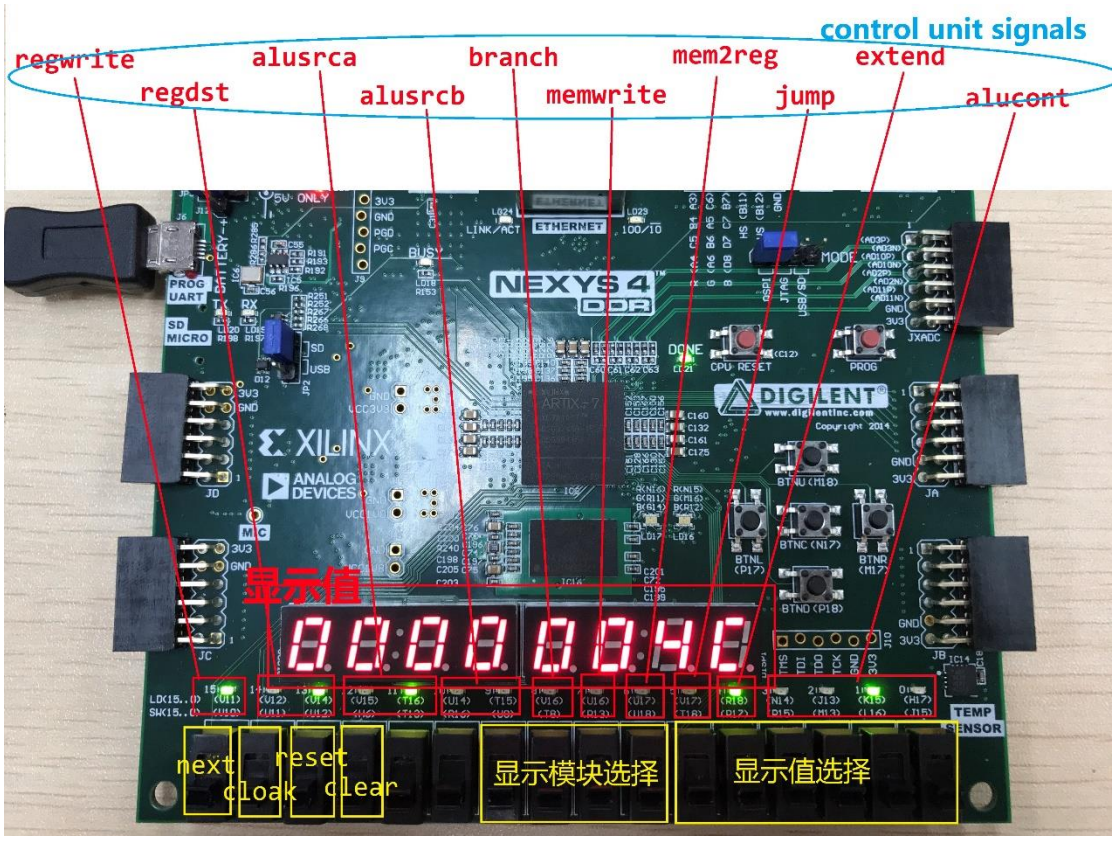

Figure 8 输入输出描述

3 模块实现

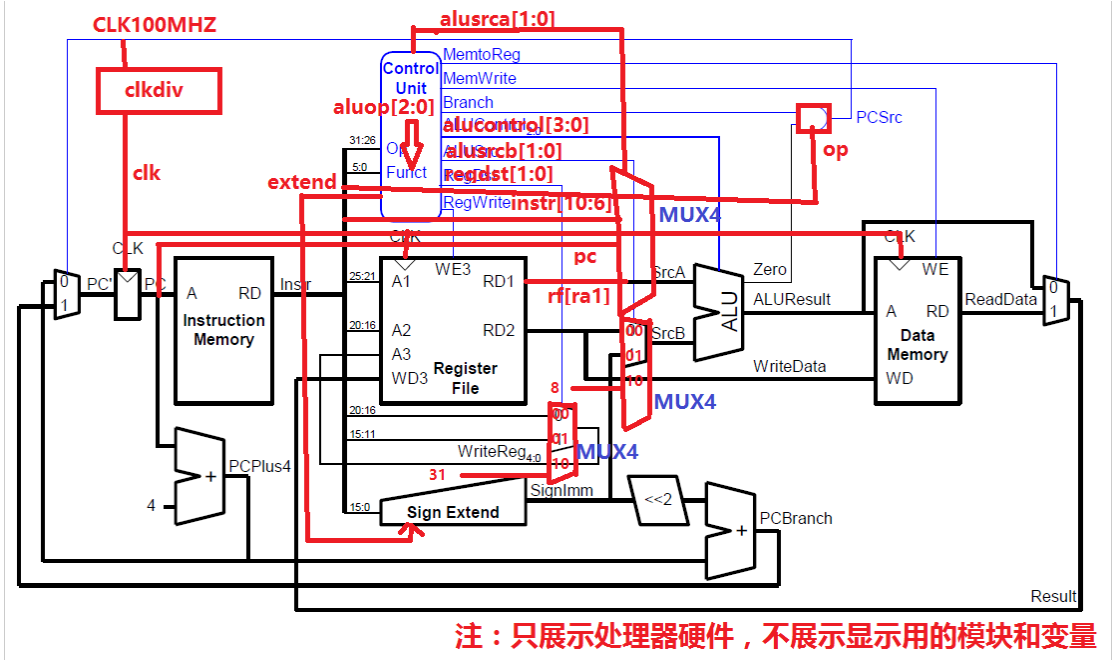

Figure 8 处理器硬件设计（红色部分为修改部分）

（代码请看工程文件，在此不展示）

3.1clkdiv

时钟分频模块，实现原理与上文中的 disdiv 模块一样，但是频率不一样：**用于开发板查看指令运行时，为了能看的清楚，一般频率在 1Hz~1.5Hz，使用 q[26]；在仿真时，一般运行一次有 10ms，为了能在运行 10~20ms 内就能查看所有的指令运行情况，采用 q[0]**，也就是 100MHz。

3.2MIPS

MIPS 是处理器中最重要的模块，**由 control_unit、datapath 两个模块组成**。其中，datapath 含有处理器中的大部分硬件模块，并连接成数据通路；而 control_unit 含有 main_decoder、alu_decoder，共同根据 instr 生成信号，并调控 datapath 中的各个模块以及 dmem 的运行。

3.2.1 control_unit

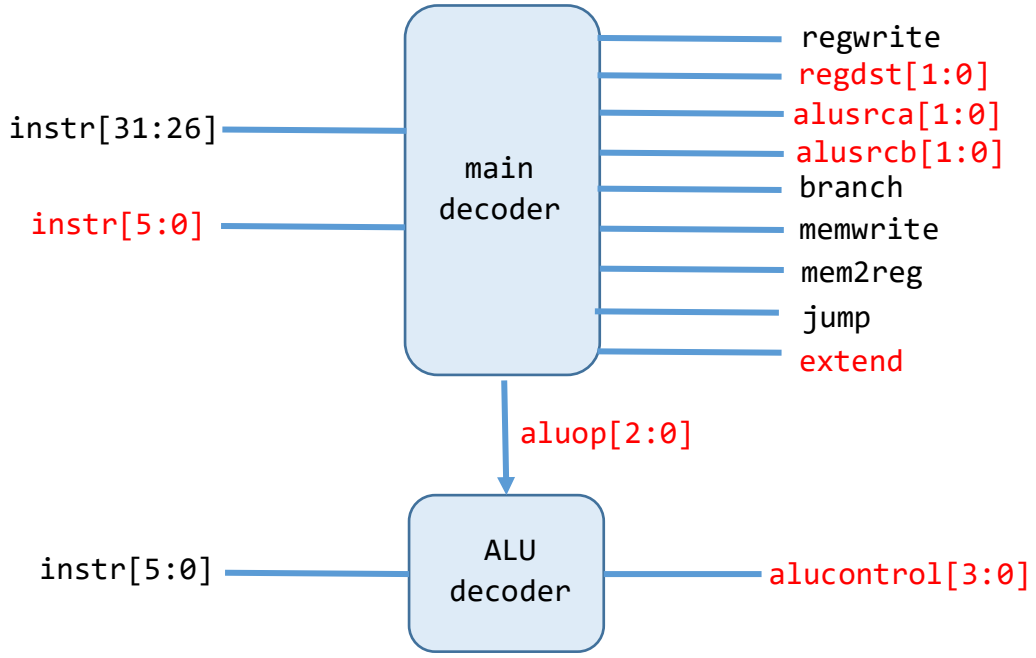

Figure 9 control_unit 硬件结构**（红色为修改/增加的接口）**

在《数字设计和计算机体系结构》书中的基础上，做了以下**修改：**

1. 添加的 jal 指令中有 GPR[31]<-PC+8 的操作，于是**拓宽 regdst 到 2 位长**。在原来的 instr[20:16]、instr[15:11]两个选择上，增加了 5’b11111、5’b00000（暂时没用）两个选择。
2. 添加的 sll、srl、sra 指令中需要用 instr[10:6]作为移位位数，故添加 alusrca 信号，在原来只能取 rf[ra1]的基础下，添加取 instr[10:6]的选择；另外，添加的 jal 命令中有 GPR<-PC+8 的操作，再添加取 PC 的选择。因此需要添加 **2 位长的 alusrca，来控制 ALU 的 srca 输入的取值**。此外，sll、srl、sra 指令虽然是 R type，但是在 instr 中两个 reg 的位置却与其他 R type 指令不一样，于是传入 **instr[5:0]**(funct)，用以判断 reg 应该从指令哪一部分取。
3. 把原来的 alusrc 改为 **alusrcb，来控制 ALU 的 srcb 输入的取值**。由于 jal 命令要取 8 值，故**拓宽为 2 位**，添加 4’b1000 的选择。
4. 由于 andi、ori、xori 需要对立即数进行零拓展，而其他 R type、ls、sw 等指令采用符号拓展，因此添加 **extend** 信号，用以判断拓展方式。
5. 由于添加了 andi、ori、xori、slti、lui 等运算，因此算上 default，

alu_decoder 共需要 7 种 aluop，故把 **aluop 从 2 位拓展为 3 位**。

6. 由于添加了 or、slt、nor、xor、sll、srl、sra、lui 等运算，因此控制 ALU 选择运算方式的 **alucontrol 要从 3 位拓宽为 4 位**。
7. 由于有 beq 和 bne 两个指令，一个在 zero 为 1 时有效，另一个在 zero 为 0 时有效，因此**计算 pcsrc 时要根据 op 来判断取哪种**。

3.2.2 datapath

3.2.2.1 PC(寄存器)

在上升沿时根据 reset 信号，选择把 pcnext 或 0 加载到 pc 上。

3.2.2.2 pcadd1(加法器)

计算 pc+4，获得 pcplus4，一般的指令都会取 pcplus4 作为 pcnext。

3.2.2.3 immsh(移位器) + pcadd2(加法器)

把拓展为 32 位的 instr[15:0]左移 2 位，即 x4。beq、bne 指令都是指定向前/后跳的指令数目，因此需要把数目 x4 获得 signimmsh，并和(pc+4)相加，形成最终要跳到的 pc 位置 pcbranch 作为 pcnext 的选项之一。

3.2.2.4 pcbrmux(MUX2) + pcmux(MUX2)

根据{pcsrc,jump}选择最终的 pcnext。若为 j、jal 等指令，则为 2'b1x，选择{pcplus4[31:28],instr[25:0],2'b00}作为 pcnext；若为 beq、bne 等指令，则为 2'b10，取 pcbranch；而一般指令则为 2'b00，取 pcplus4 作为 pcnext。

3.2.2.5 regfile

由 reg 数组实现 32 个 32bit 的 reg，并能通过 regwrite 控制是否写入数据（如 lw 则要写入），**通过 alusrca 控制是否进行 rd1 数据读取（控制方式上文已叙述）**，同时还能读取 rd2 数据。此外，**在 clear 为 1 时，会同步读入 emptyreg.dat 文件，实现 regfile 的清零**。

3.2.2.6 wrmux(MUX4)

选择 writereg，即要写入的 reg。由于 jal 指令中有 GPR[31]<-PC+8 的操作，所以**拓宽了 regdst，添加了 5'b11111 选择，并把原来的 MUX2 改为 MUX4**。

3.2.2.7 resmux(MUX2)

通过 mem2reg 选择可能写入 regfile 的数据，若为 0 则是 alu 计算结果 aluout（如 addi），为 1 则为 dmem 读出数据 readdata（如 lw）。

3.2.2.8 se(sign_extend)

添加了 extend 信号，来判断对立即数采用零拓展（如 andi、ori、xori）或者符号拓展（如 lw、sw）。

3.2.2.9 srcbmux(MUX4)

通过 alusrcb 控制 ALU 的 srcb 输入的取值，由于 jal 指令**把 alusrcb 拓宽为 2 位，同时把 MUX2 改为 MUX4**。

3.2.2.10 ALU

通过 alucontrol 来控制 ALU 的运算方式，并输出计算结果（aluout）和 ZF（zero）。

3.3imem

imem 用于加载、储存指令，由 reg 数组实现。在程序初始化时，通过$readmemh 从.dat 文件读入全部指令并储存。程序运行中时，传入 pc[7:2]作为 addr，在每一个上升沿读取指令。

3.4dmem

dmem 用于储存、读取数据，由 reg 数组实现。传入 dataadr 作为地址，读出数据，并根据 memwrite 信号决定是否要写入数据。**此外，当 clear 为 1 时，将会同步读入 emptyRAM.dat 文件，实现 dmem 的清零。**

3.5top

top 为顶层模块，除了用于在开发板上输出数据的内容外，就是组织 MIPS、dmem、imem 的结构。MIPS 进行正常的处理器运行，并从 imem 中读取 instr 指令来执行，从 dmem 中读取数据（如 lw 等）或者向 dmem 写入数据（如 sw 等）。

4 仿真实现

仿真用于检验处理器是否能按照所执行的代码要求运行、能否实现相关功能，**检验时主要通过观察 pc、regfile、dmem 等值来判断是否正确**。注意在仿真前，需要将 clkdiv 中的分频取 outclk=q[0]，使其能在短时间内运行完全部指令，便于观察；并将 imem 模块中待运行的代码地址改好。

其中 **testbench.v 为整个处理器的仿真文件**，另外还有 ALU、PC、adder、main_decoder、sign_extend 模块的仿真文件，在此不一一叙述了。

4.1仿真 1：memfile6（含有所有指令）

4.1.1 指令验证

memfile6 含有实现的所有指令，可以根据 pc 值来一条条对应检验指令是否正确。在这里简单指出 sra、jal、xori 指令的正确性。

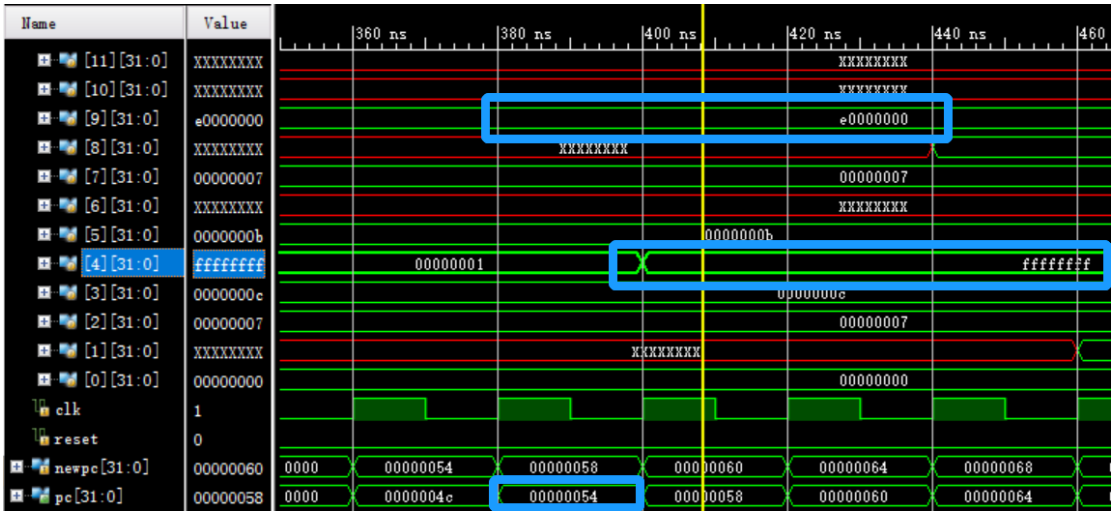

Figure 10 sra: $4 = $9 >> 29 = ffffffff

Figure 11 jal: jal 0x60

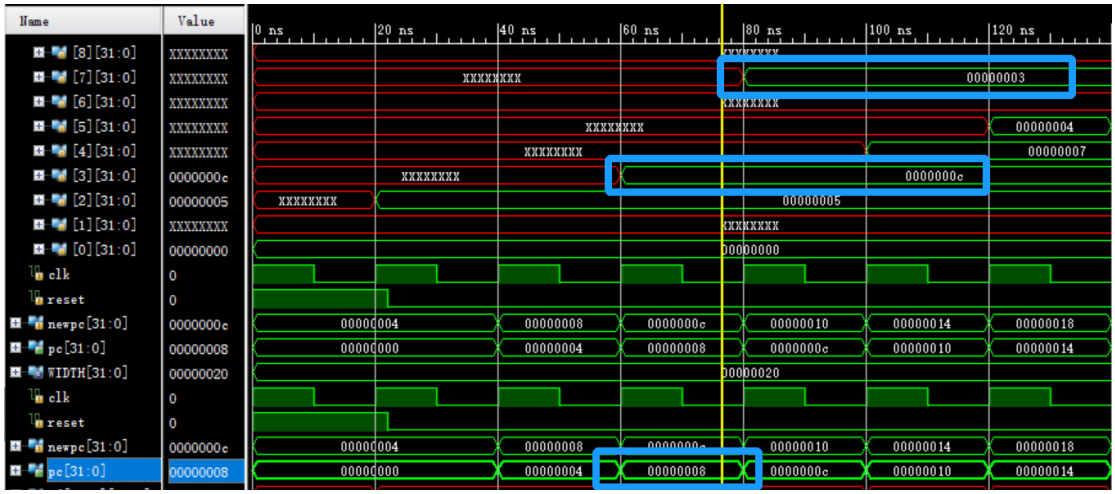

Figure 12 xori: $7 = $3 ^ 15 = 3

4.1.2 功能验证

可以验证在 SW 调控下是否能达到对应的效果。由于所有功能都已经向陈老师演示，在这里只简单验证 reset、clear 的正确性。

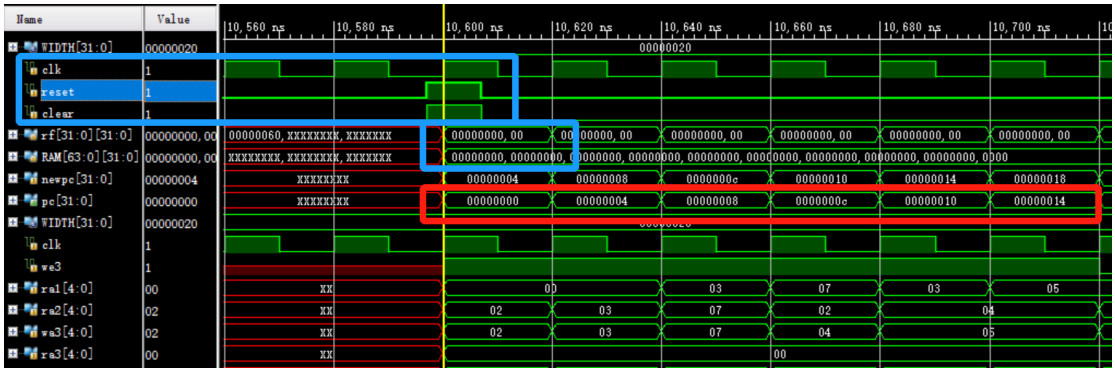

Figure 13 reset 和 reset 功能验证

可以看见，在 reset 为 1，clear 为 1 时，且当时钟上升沿到来时（大蓝框），**会同步实现 pc、regfile 和 RAM 的清零（小蓝框），即可以使程序回到最初状态，并且重新开始运行**

程序（红框）。有了这个功能，就可以随时重新开始程序的运行，运行了一遍还不够的话，可以在板子上重新再来。

4.2仿真 2：sortfile0（8 个数的插入排序）

Figure 14 sortfile0 验证

可以看到插入排序的变化过程：

84273516->64273518->14273568->14263578->14253678->13254678->12345678

4.3仿真 3：sumfile0（1~100 求和）

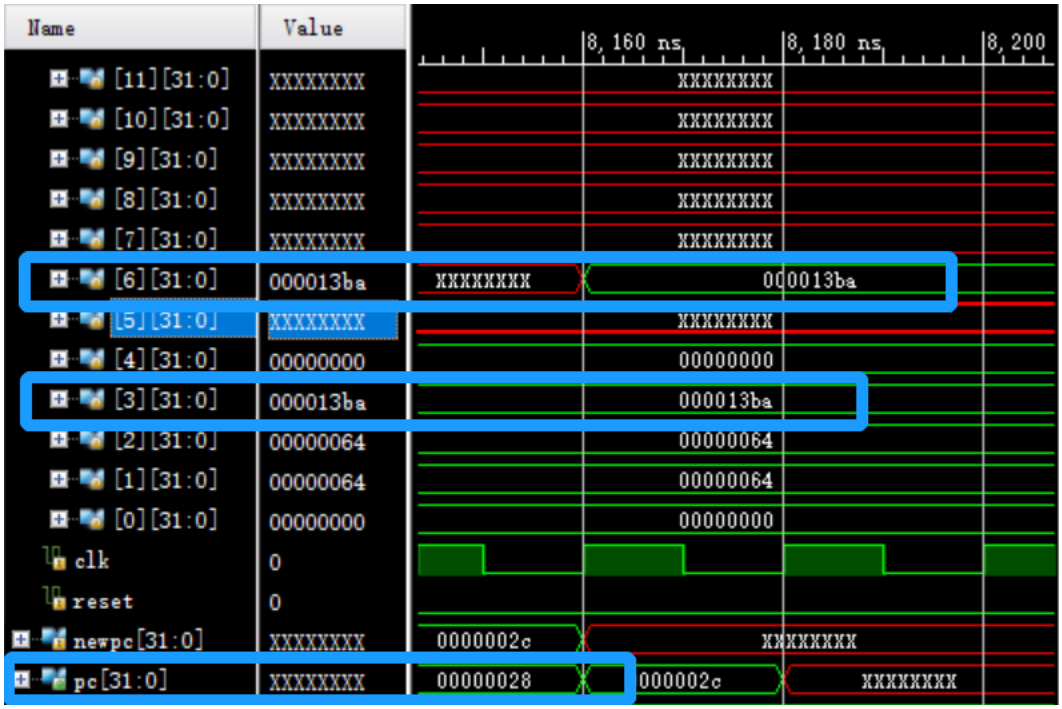

Figure 15 1~100 求和

5 性能分析

（以下性能在运行 sortfile0 程序、clkdiv 分频取 q[0]的情况下测定。）

5.1时钟占用

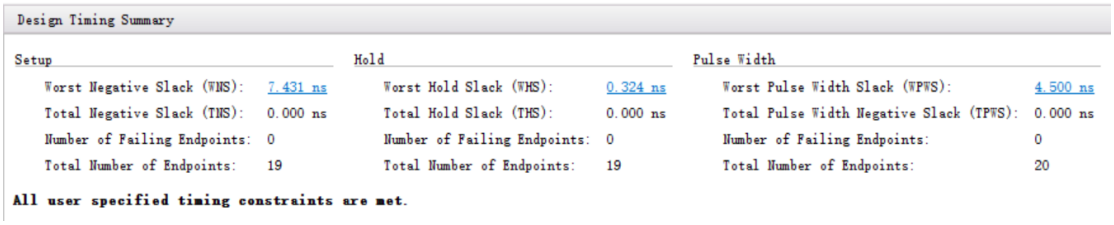

Figure 16 时钟占用

5.2资源占用

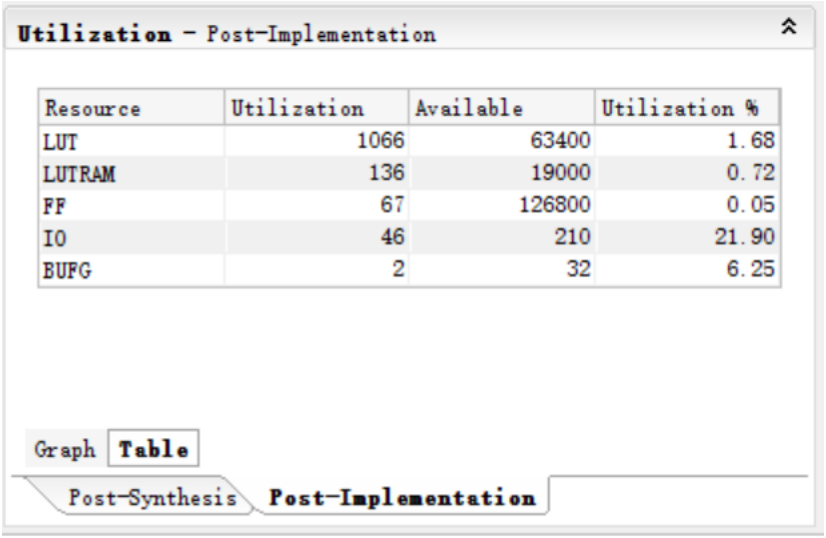

Figure 17 资源占用

6 实验感想

6.1对单周期 MIPS 处理器的运行机制更加了解，为之后多周期、流水线 MIPS 处理器的实现打下基础。

6.2对 MIPS 指令集有了更深的了解，并学会书写 MIPS 指令程序。

6.3对 verilog 语言更加深入。

6.4**能熟练运用 SW 来实现功能，能使用 LED 以及 8 个 7 段显示数码来直观地显示所有数值，能极大地方便 debug 及正确性验证。**