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 2](#_Toc512860784)

[2 显示实现 4](#_Toc512860785)

[3 模块实现 7](#_Toc512860786)

[4 仿真实现 10](#_Toc512860787)

[5 性能分析 13](#_Toc512860788)

[6 实验感想 13](#_Toc512860789)

# 总体状况

* 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. **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. 规格

regfile: 32bit\*32

data RAM: 32bit\*64

instr RAM: 32bit\*64

* 1. **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**
  2. 总体结构
     1. 代码结构

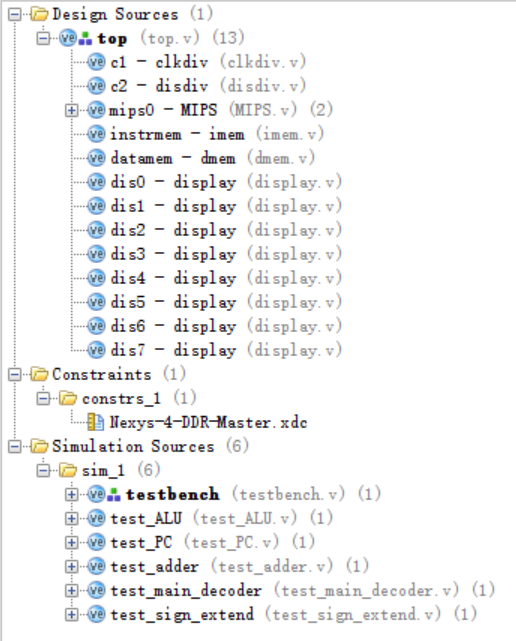


Figure 1 总体代码结构

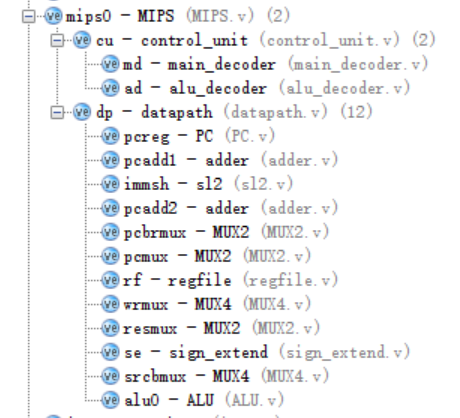


Figure 2 MIPS模块结构

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。

* + 1. 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. 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。

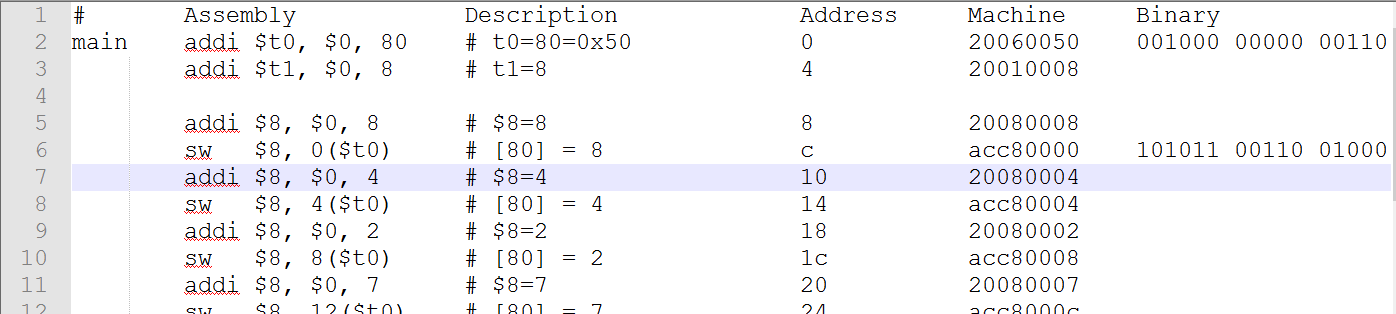


Figure 3 sortfile0.txt部分示例

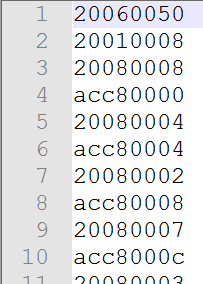
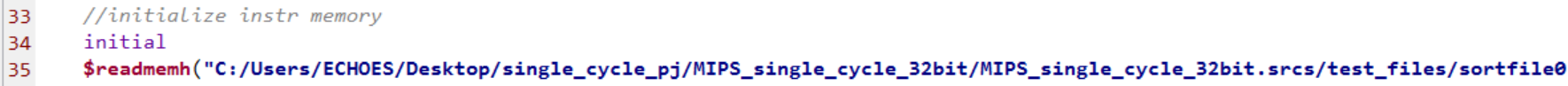


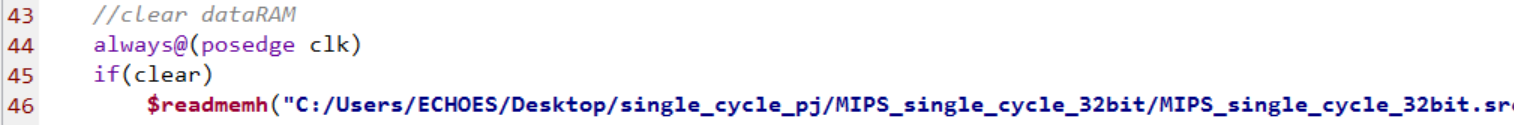
Figure 4 sortfile.dat部分示例

* 1. 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）

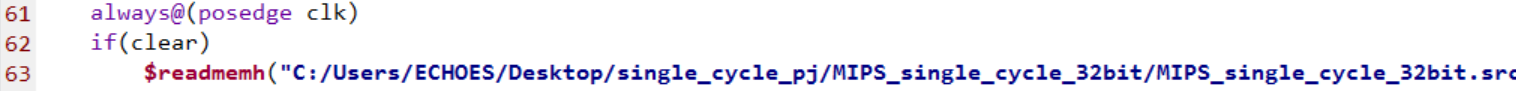
在imem模块，需把如下地址修改为执行代码文件的实际地址。



在dmem模块，需要把如下地址修改为emptyRAM.dat的实际地址。



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



# 显示实现

（代码实现请看工程文件，在此不展示）

* 1. disdiv

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。

* 1. display

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**。

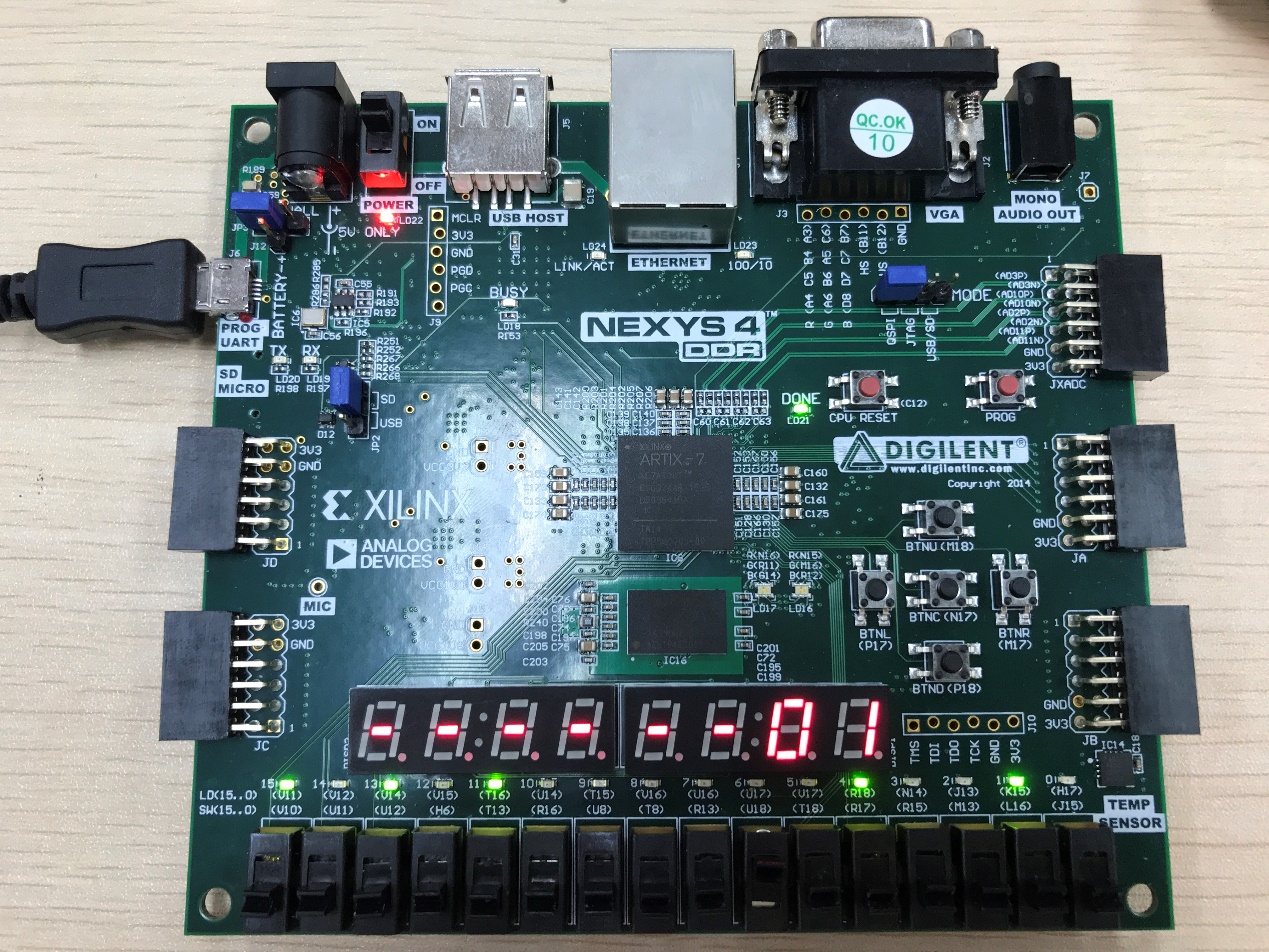


Figure 5 7段数码显示示例

* 1. top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。

* + 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。

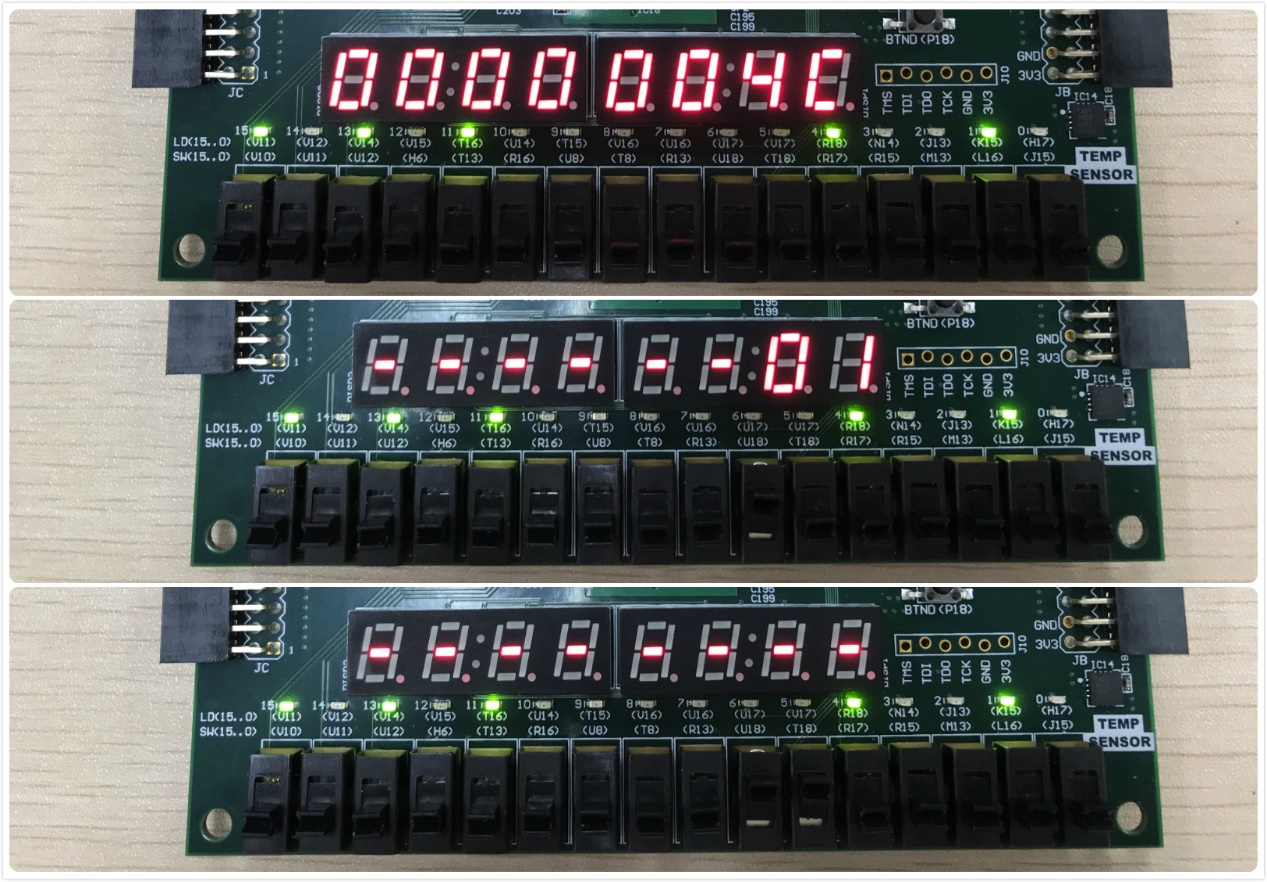


Figure 6 7段数码管显示示例

* + 1. 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

1. C[6:0]:在某一时刻需要输出的7段数码数字
2. AN[7:0]:指定8个中需要点亮的7段数字
3. **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

1. **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。
   * 1. 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。**

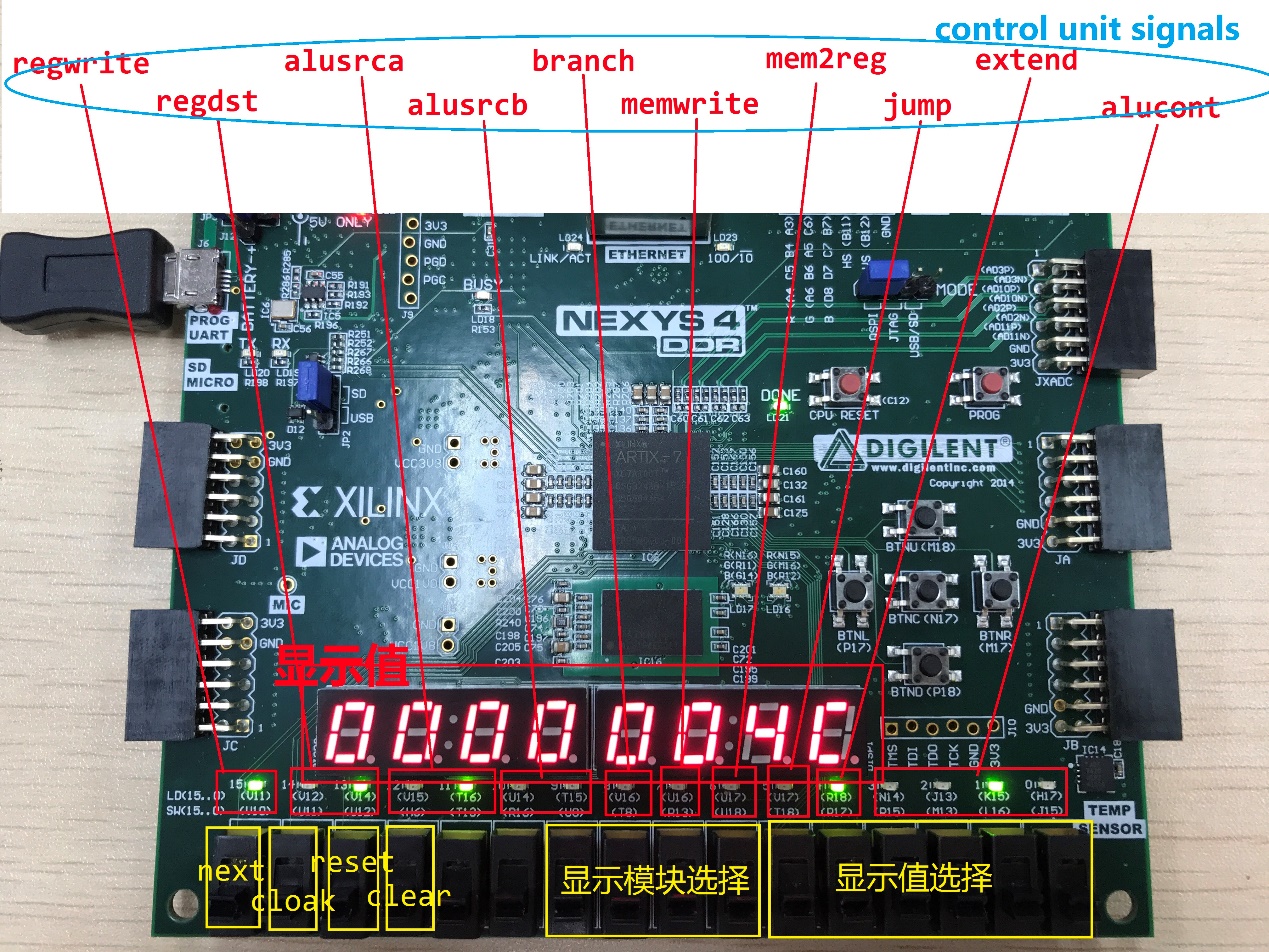


Figure 8 输入输出描述

# 模块实现

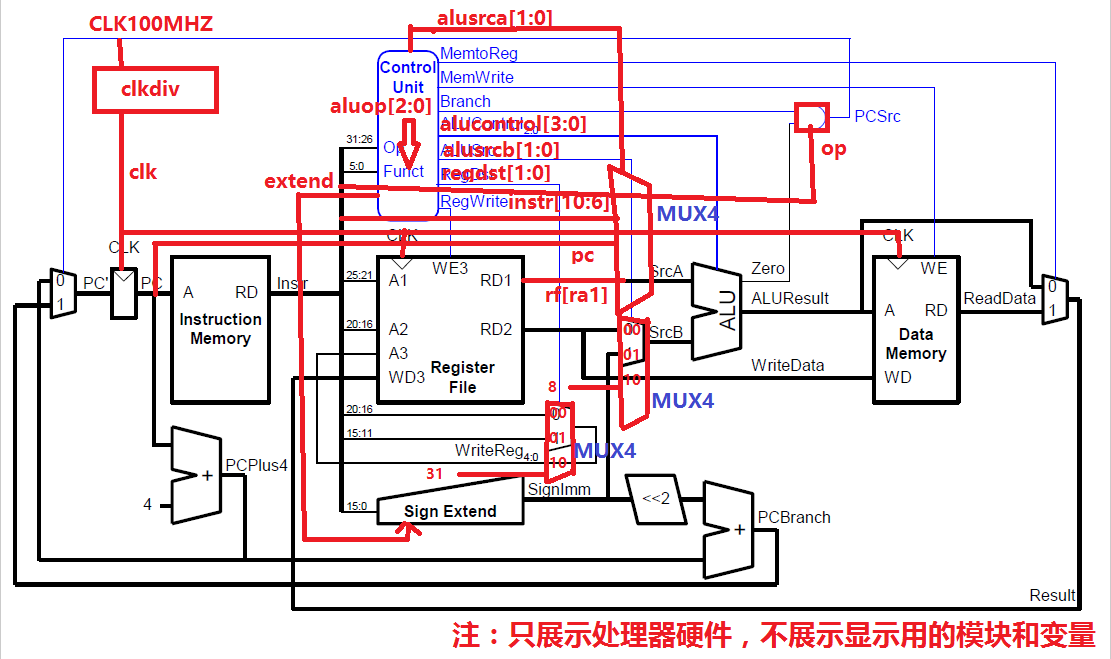


Figure 8 **处理器硬件设计（红色部分为修改部分）**

（代码请看工程文件，在此不展示）

* 1. clkdiv

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。

* 1. MIPS

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。

* + 1. control\_unit

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

instr[5:0]

instr[5:0]

instr[31:26]

mem2reg

regwrite

regdst[1:0]

alusrca[1:0]

alusrcb[1:0]

branch

memwrite

jump

aluop[2:0]

extend

alucontrol[3:0]

main

decoder

ALU

decoder

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**。
    8. datapath
       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。

* + - 1. pcadd1(加法器)

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

* + - 1. 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。

* + - 1. 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。

* + - 1. 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**。

* + - 1. 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**。

* + - 1. 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）。

* + - 1. 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）**。

* + - 1. 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**。

* + - 1. ALU

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

* 1. imem

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。

* 1. dmem

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。**

* 1. top

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）。

# 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。

* 1. **仿真1：memfile6（含有所有指令）**
     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。

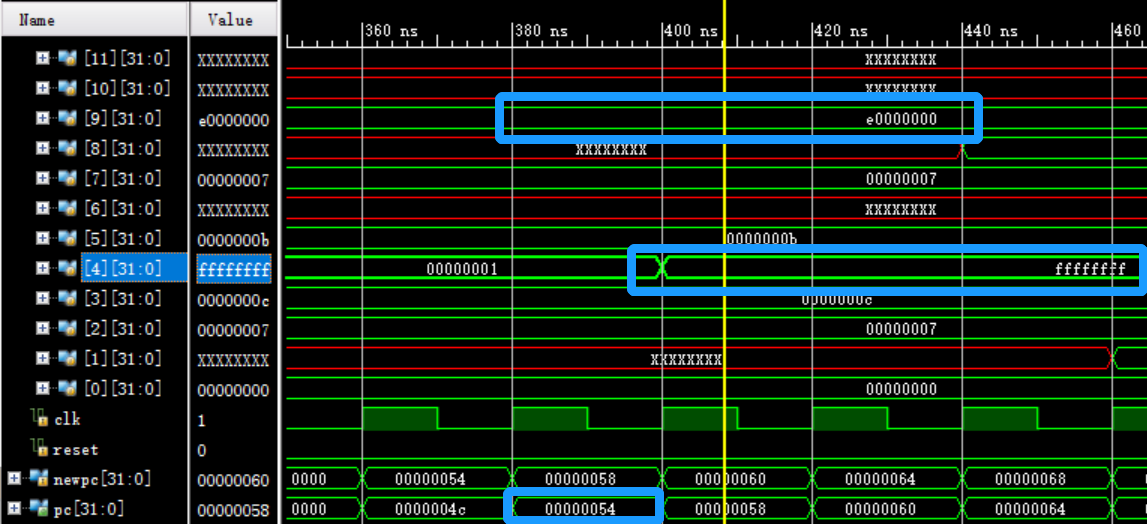


Figure 10 sra: $4 = $9 >> 29 = ffffffff

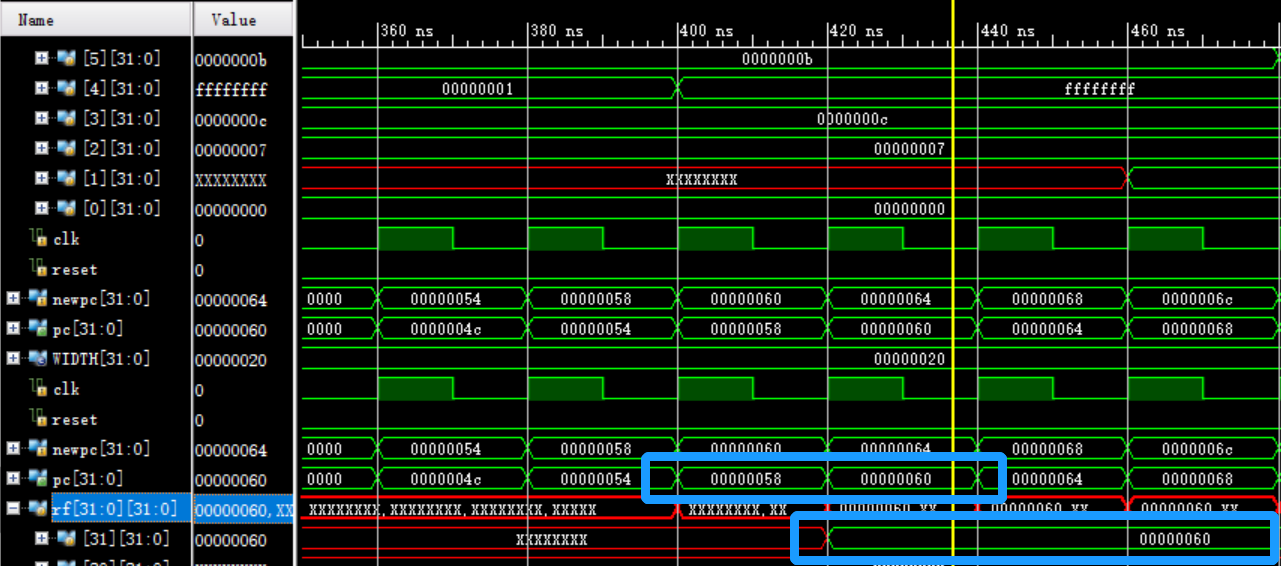


Figure 11 jal: jal 0x60

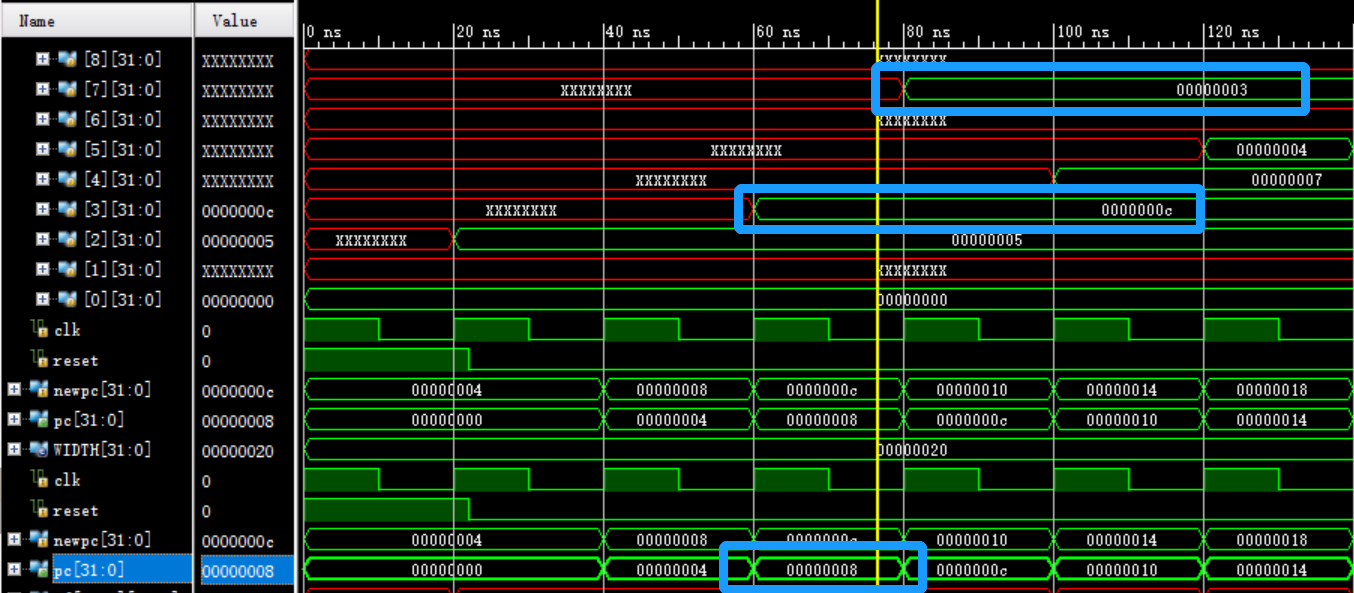


Figure 12 xori: $7 = $3 ^ 15 = 3

* + 1. 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。

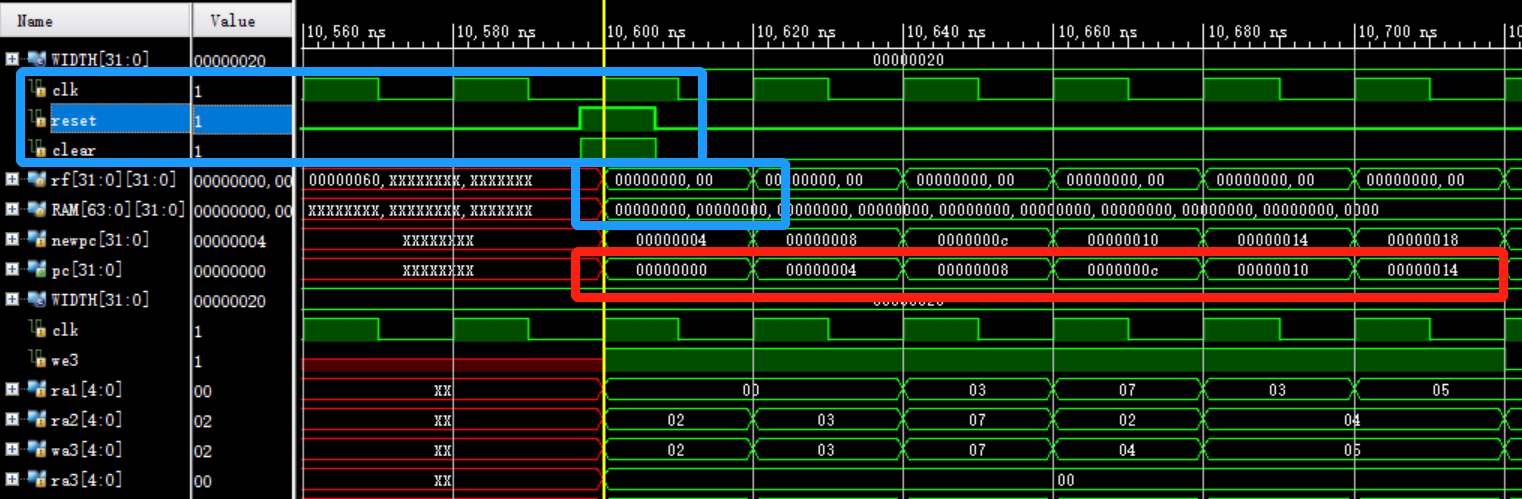


Figure 13 reset和reset功能验证

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。**

* 1. **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**

* 1. **仿真3：sumfile0（1~100求和）**

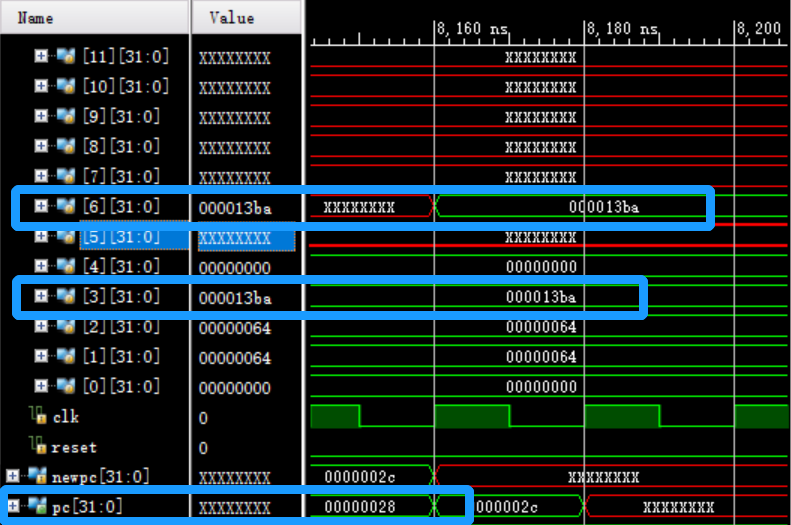


Figure 15 1~100求和

# 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。）

* 1. 时钟占用

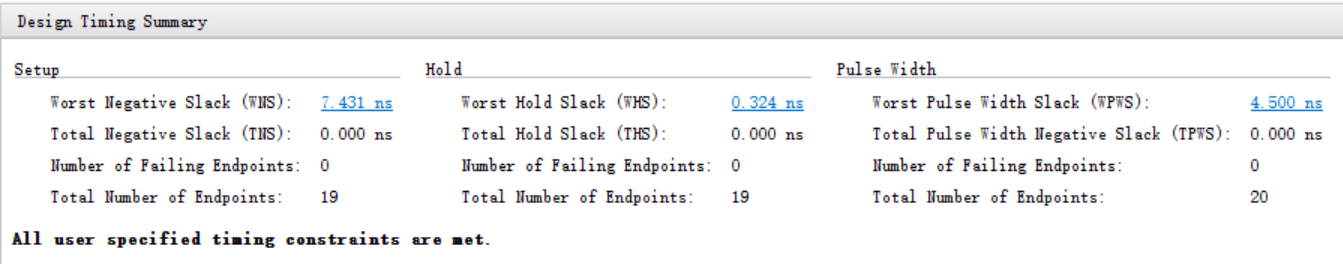


Figure 16 时钟占用

* 1. 资源占用

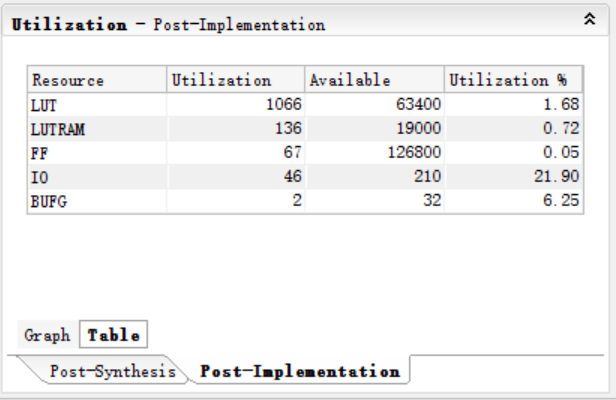


Figure 17 资源占用

# 实验感想

* 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。
  2. 对MIPS指令集有了更深的了解，并学会书写MIPS指令程序。
  3. 对verilog语言更加深入。
  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。**