Multi-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 8](#_Toc512860786)

[4 仿真实现 11](#_Toc512860787)

[5 性能分析 13](#_Toc512860788)

[6 实验感想 14](#_Toc512860789)

# 总体状况

* 1. 多周期MIPS处理器

单周期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，但是在任一时刻，处理器中只有部分元件处于使用中的状态，而大部分的元件在静止中，加上时钟周期是由最慢的指令决定的，因此在执行其他指令的时候，还存在一段时间是所有元件都在静止的，这极大地限制了处理器的效率。此外单周期处理器使用了3个加法器、指令存储器和数据存储器分离，过于冗余。

因此有了多周期处理器，通过往处理器中**添加寄存器**，用于储存中间量，以此把处理器划分为多个部分，同时把各个指令分解为多个短步骤，指令运行会更加高效、紧凑。同时还复用**单个ALU**，**合并指令、数据存储器**。此外，还需要**设计FSM**来控制每条指令各部分的运行顺序。

* 1. **指令集**
     1. 书中结构已包含的指令：add, sub, and, or, slt, sw, lw, beq
     2. 书中添加的指令：addi, j
     3. **要求以外添加指令：bne, nop, andi, ori, slti, 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

RAM: 32bit\*64

* 1. **实现功能（均已展示）（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（stop）
     2. 可在暂停的情况下，实现**单条指令运行**功能（next）
     3. 可以使**各个寄存器、regfile、memory同步归零**（reset）
     4. 可以通过16个LED灯、8个7段数码，**直观地查看**电路中各个模块**所有的值**
  2. 总体结构
     1. 代码结构

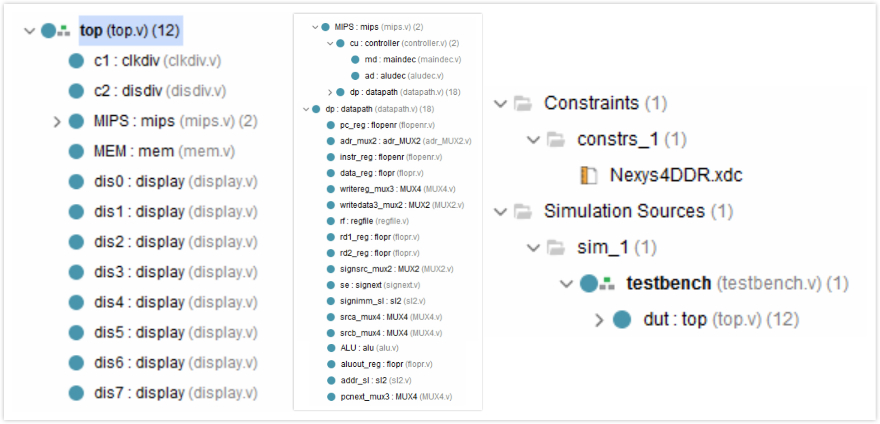


Figure 1 总体代码结构

其中，top为顶层文件，分为MIPS、MEMORY、显示模块三个部分。而testbench为处理器的仿真文件。

* + 1. 硬件结构

由于额外添加了j, addi, **bne, nop, andi, ori, slti, xor, xori, nor, sll, srl, sra, jal, lui共15条指令，**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. **FSM结构**

由于**多支持了15条指令**，因此在原来的基础上把**state**从11种增加到**19种**。在后面会详细介绍。

* 1. 测试代码

位于MIPS\_multi\_cycle\_32bit\MIPS\_ multi \_cycle\_32bit.srcs\test\_files\文件夹中，含有**memfile6（含有所有指令）、sumfile0（1~100的求和）、sortfile0（插入排序）共3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。

此外，由于合并了指令、数据存储器，因此原来的更改了低位数据内容的指令会覆盖指令代码，于是要**把数据存取的位置改到程序指令的后面**，以免影响指令的存储。

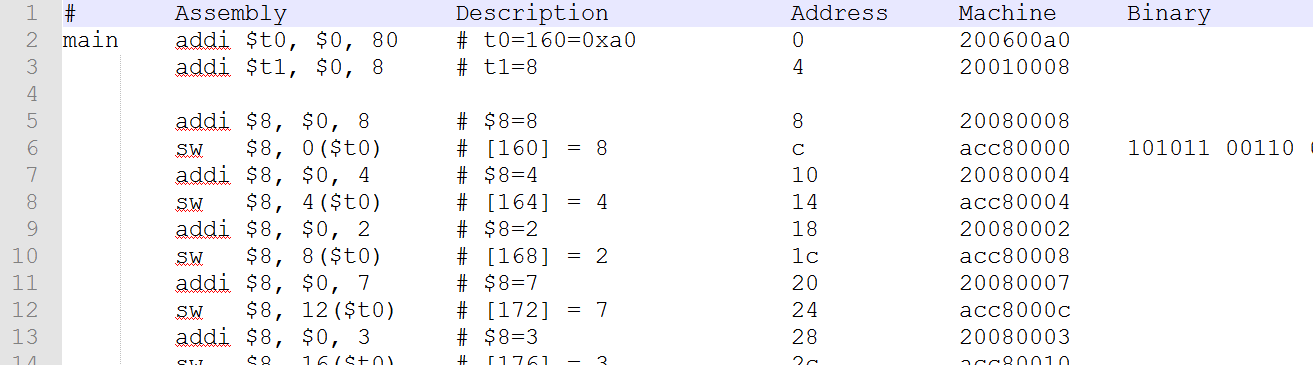


Figure 2 sortfile0.txt部分示例

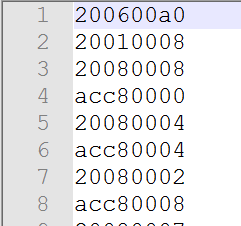
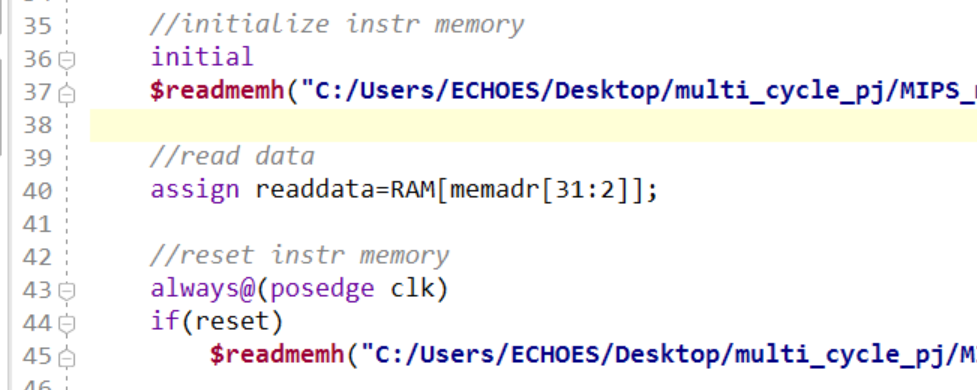


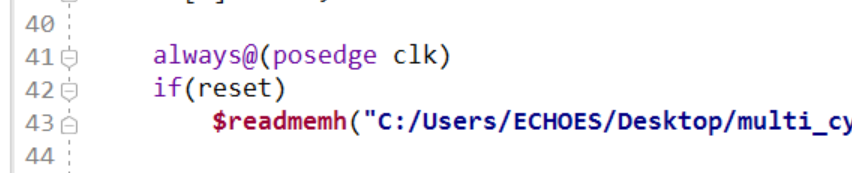
Figure 3 sortfile.dat部分示例

* 1. 文件读入地址修改（以下文件均可在test\_files文件夹中可以找到）

在mem模块，需把如下两行地址修改为待执行代码文件的实际地址。



在regfile模块，需要把如下地址修改为emptyreg.dat（在test\_files文件夹中）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**。

* 1. top（用于显示的部分）及约束文件
     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；stop-SW[14]：为1时，屏蔽时钟，暂停。通过把输入MEM和MIPS的clk信号修饰为next|((~stop)&clk)来实现。
2. lightsrc-SW[13]：由于controller的输出信号很重要，因此用LED灯输出。但信号宽度超过了16，因此加入1位的信号来确定显示前半部分/后半部分的控制信号。
3. reset-SW[12]：为1时，寄存器、mem、regfile同步归零
4. 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。
5. clk-CLK100MHZ：输入时钟，用分频模块产生其他所需频率的时钟

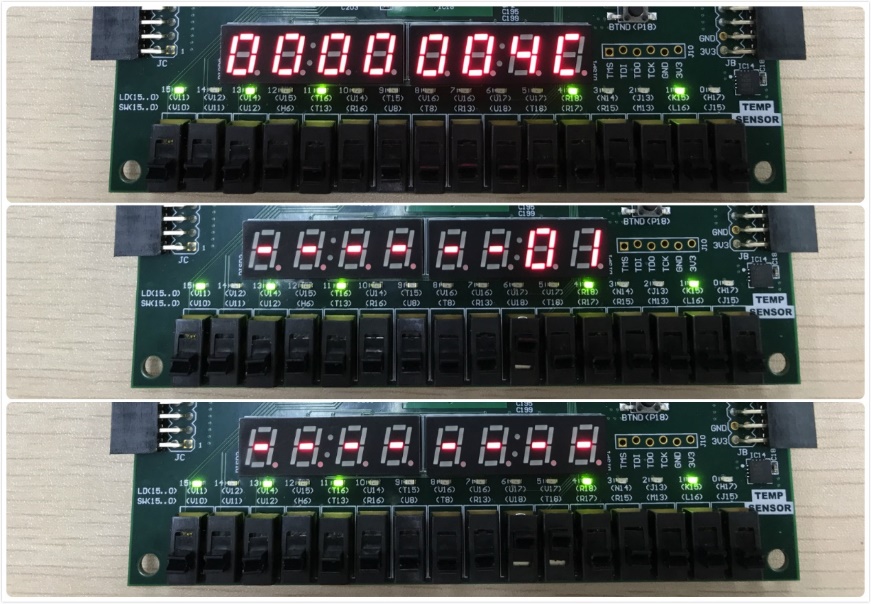


Figure 4 7段数码管显示示例

* + 1. 输出及显示机制

1. **discu[15:0]-LED[15:0]**:由于controller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。根据lightsrc信号来决定输出前半/后半部分的信号。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| light  src | 15 | 14 | 13 | 12 | 11 | 10 | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | | 1 | 0 |
| 0 | alusrca | | branch | iord | memtoreg | regdst | | | alusrcb | | pcsrc | | aluop | | | | extend | extsrc |
| 1 | next | stop | lightsrc | reset | clk |  |  | | pcen | alucontrol | | | | pcwrite | | memwrite | irwrite | regwrite |

Figure 5 LED显示对应关系

1. C[6:0]:在某一时刻需要输出的1个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中的各个模块、mem模块，同时向上述模块各传入一对输出值tx[31:0]、disopx[7:0]值，通过SW来选择要显示的数据，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。**以上内容均通过嵌套的case语句来实现。（由于按照模块来划分，因此部分数值是有重复的、有分割的，虽然有些冗余，但是能方便对应某一模块来查看其全部输入输出值，易于debug）**
4. **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，最终会显示出对应的8个数字。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SW[9:6] | 模块 | SW[5:0] | 值 | 十六进制位数（0~8位） |
| 0000 | FSM | 000000 | state | 2 |
| 000001 | nextstate | 2 |
| else | 0 | 0 |
| 0001 | PC | 000000 | pc | 8 |
| 000001 | pcnext | 8 |
| 000010 | pcen | 1 |
| 000011 | aluresult | 8 |
| 000100 | aluout | 8 |
| 000101 | pcjump | 8 |
| 000110 | pcsrc | 1 |
| else | 0 | 0 |
| 0010 | adr mux2 | 000000 | pc[7:2] | 2 |
| 000001 | aluout | 8 |
| 000010 | iord | 1 |
| 000011 | adr | 8 |
| else | 0 | 0 |
| 0011 | instr reg | 000000 | irwrite | 1 |
| 000001 | readdate | 8 |
| 000010 | instr | 8 |
| else | 0 | 0 |
| 0100 | data reg, op, funct | 000000 | readdate | 8 |
| 000001 | data | 8 |
| 000010 | op | 2 |
| 000011 | funct | 2 |
| else | 0 | 0 |
| 0101 | writereg mux3 | 000000 | instr[20:16] | 2 |
| 000001 | instr[15:11] | 2 |
| 000010 | 5’b11111 | 2 |
| 000011 | regdst | 1 |
| 000100 | writereg | 2 |
| else | 0 | 0 |
| 0110 | writedate3 mux2 | 000000 | aluout | 8 |
| 000001 | data | 8 |
| 000010 | memtoreg | 1 |
| 000011 | writedate3 | 8 |
| else | 0 | 0 |
| 0111 | reg | 000000 | instr[25:21] | 2 |
| 000001 | instr[20:16] | 2 |
| 000010 | writereg | 2 |
| 000011 | writedata3 | 2 |
| 000100 | regwrite | 1 |
| 000101 | regout1 | 8 |
| 000110 | regout2 | 8 |
| 000111 | regouta | 8 |
| 001000 | regoutb | 8 |
| 001001 | writedata | 8 |
| else | 0 | 0 |
| 1000 | regfile | 000000~  011111 | rf[SW[4:0]] | 8 |
| else | 0 | 0 |
| 1001 | signext | 000000 | instr[15:0] | 4 |
| 000001 | instr[10:6] | 2 |
| 000010 | extsrc | 1 |
| 000011 | extend | 1 |
| 000100 | signimm | 8 |
| 000101 | signimmsh | 8 |
| else | 0 | 0 |
| 1010 | ALU | 000000 | pc | 8 |
| 000001 | regouta | 8 |
| 000010 | 16 | 8 |
| 000011 | signimm | 8 |
| 000100 | alusrca | 1 |
| 000101 | srca | 8 |
| 000110 | regoutb | 8 |
| 000111 | 4 | 8 |
| 001000 | signimmsh | 8 |
| 001001 | alusrcb | 1 |
| 001010 | srcb | 8 |
| 001011 | alucontrol | 1 |
| 001100 | aluresult | 8 |
| 001101 | zero | 1 |
| 001110 | aluout | 8 |
| else | 0 | 0 |
| 1011 | addr sl | 000000 | instr[25:0] | 8 |
| 000001 | addrsl | 8 |
| 000010 | pc[31:28] | 1 |
| 000011 | pcjump | 1 |
| else | 0 | 0 |
| 1100 | mem | 000000 | memwrite | 1 |
| 000001 | memadr | 8 |
| 000010 | writedata | 8 |
| 000011 | readdata | 8 |
| else | 0 | 0 |
| 1101 | memfile | 000000~  111111 | RAM[swt[5:0]] | 8 |
| else |  |  | 0 | 0 |

Figure 6 开关取值对应关系

* + 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，并能按照模块分类，直观查看处理器中任意模块的所有值。**

# 模块实现

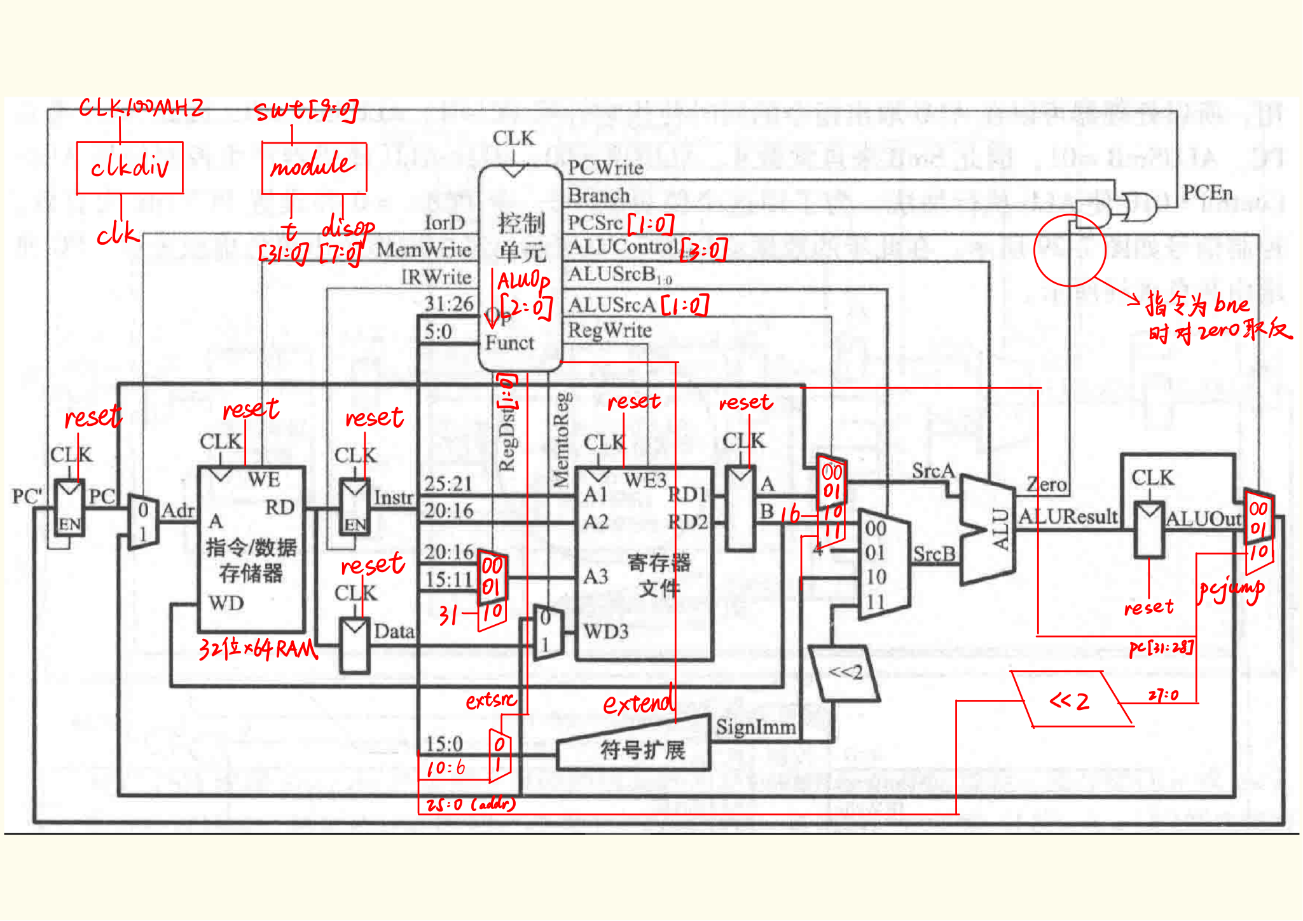


Figure 7 **处理器硬件设计（红色部分：修改了的硬件、增加/拓宽的变量）**

**（左上角：分频模块、7段数码取值模块设计）**

（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controller、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controller含有**main\_decoder、alu\_decoder**，以及main\_decoder含有的**FSM**，共同根据instr的内部顺序，来生成对应的控制信号，并调控**datapath**中的各个模块以及mem的运行。

* + 1. controller

controller解码instr，而FSM控制每条指令内部的运行顺序并生成对应的控制信号。FSM按照下图逻辑，通过state register更新state，并通过next state logic来获得next state，并输出当前state对应的控制信号，用于调控datapath和mem模块的运行。

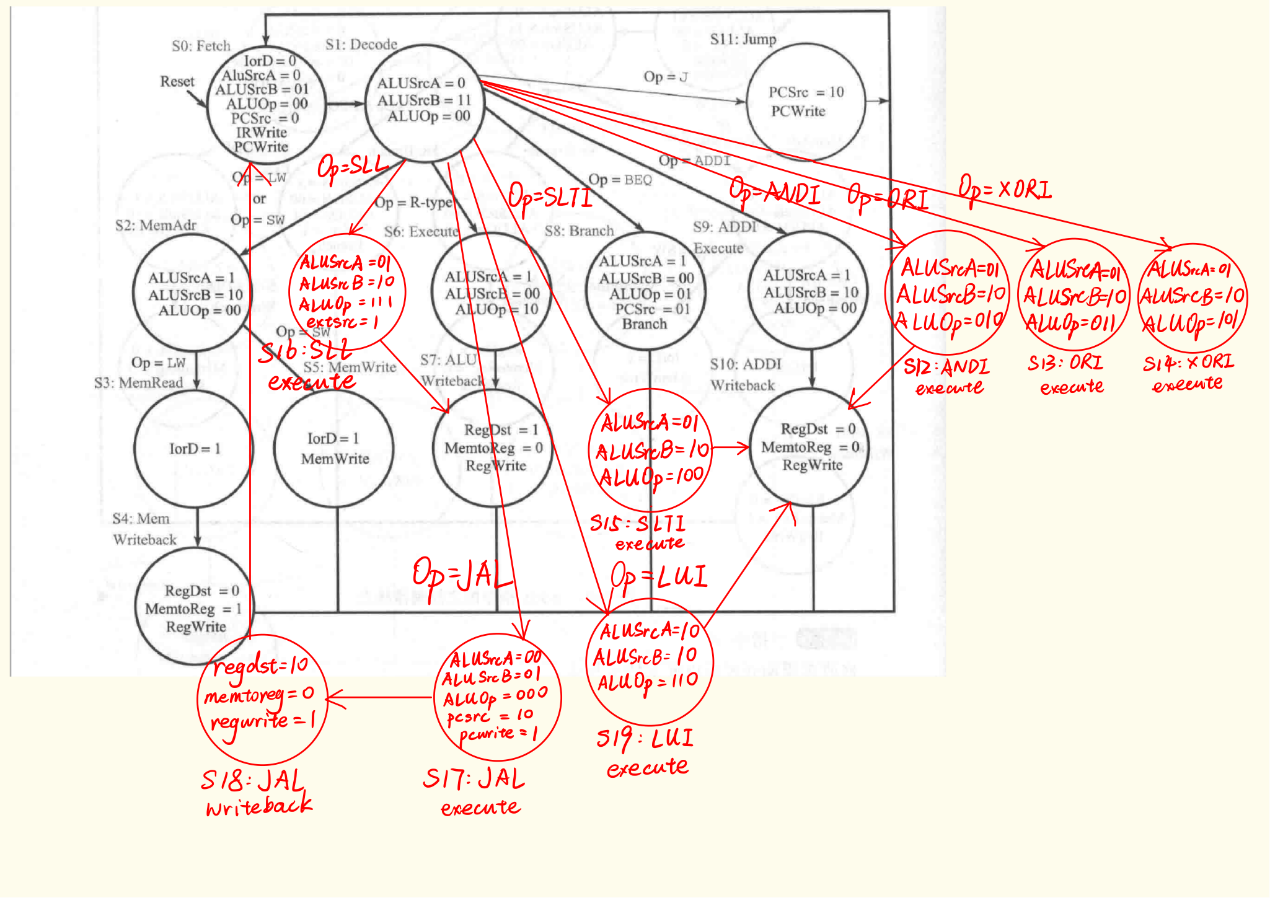


Figure 8 FSM控制逻辑**（红色部分：增加了的状态）**

在书中设计的的基础上，做了以下**修改：**

* + 1. 由于添加了15条指令，**FSM额外增加了8个state**，此外还有部分新指令的state可以与原来的state复用，就不再冗余地额外添加。
    2. 添加的jal指令中有GPR[31]<-PC+8的操作，reg需要取常数31，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31的选择。
    3. 添加的sll、srl、sra指令中需要用instr[10:6]作为移位位数，且需要进行位拓展，故**添加extsrc信号**，用来选择向符号扩展模块输入instr[15:0]或者instr[10:6]。
    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pcen时要根据op来判断取哪种**。
    8. 由于加入了j指令，在获得pcnext的时候要增加pcjump这一个选项，于是**拓宽pcsrc到2位长**，并加入pcjump的选择。
    9. 由于lui指令是左移16位，而被左移数instr[15:0]作为srcb的选择，则需要在srca的选择中添加常数16。且由于sll、sra、srl指令需要取instr[20:16]寄存器的值，因此从中取出的值作为srcb，但同时还需要移位数，就只能作为srca，因此要把移位结果signimm多连一根线到srca的选择中。因此srca要多两个选择，因此**alusrca要拓宽到2位**。
    10. datapath

datapath中各个部分受到controller发来的控制信号控制着，并随着时钟上升沿，依次执行指令的一个个部分，并一条条指令地执行下去。

* + 1. enable – resettable flop：PC、instr寄存器。在上升沿时，若reset为1，则加载0，若为0，则看enable信号，若为1则更新寄存器的值，否则保持不变。
    2. resettable flop：MEM输出数据、寄存器输出数据、ALUresult寄存器。相当于enable信号总是为1.
    3. 符号扩展

**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. 移位器

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. 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regdst选择要写入的寄存器（如lui则要选择常熟31），通过memtoreg来选择要写入的数据（如一般的运算指令则要选择aluout作为写入的数据）。从寄存器中读出来的两个数据存储在寄存器中，等待下一个时钟上升沿时取用。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. 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alusrca来控制取用的srca值、通过alusrcb来控制取用的srcb（如add，则要取寄存器文件读出来两个数值分别作为srca、srcb）。通过alucontrol来控制ALU的运算方式，并输出计算结果（aluresult）和ZF（zero）。

* + 1. 为了获得pcjump，还需要多**增加一个左移两位的移位模块**，对j指令instr[25:0]处的地址x4，并拼接上pc的前4位来生成pcjump值。
    2. 通过pcsrc来选择pcnext取用的值（如为j指令，则取pcjump值）。
  1. mem

mem用于储存、读取指令、数据，由reg型数组实现。传入adr作为地址，读出指令或数据，并根据memwrite信号决定是否要写入数据。**此外，当clear为1时，将会同步读入当前的指令文件，实现mem的还原。**

* 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、mem的结构。MIPS进行正常的处理器运行，并从mem中读取instr指令来执行，从mem中读取数据（如lw等）或者向mem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、mem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mem模块中待运行的代码地址改好。另外，**testbench.v为整个处理器的仿真文件**。

* 1. **仿真1：memfile6（含有所有指令）**

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

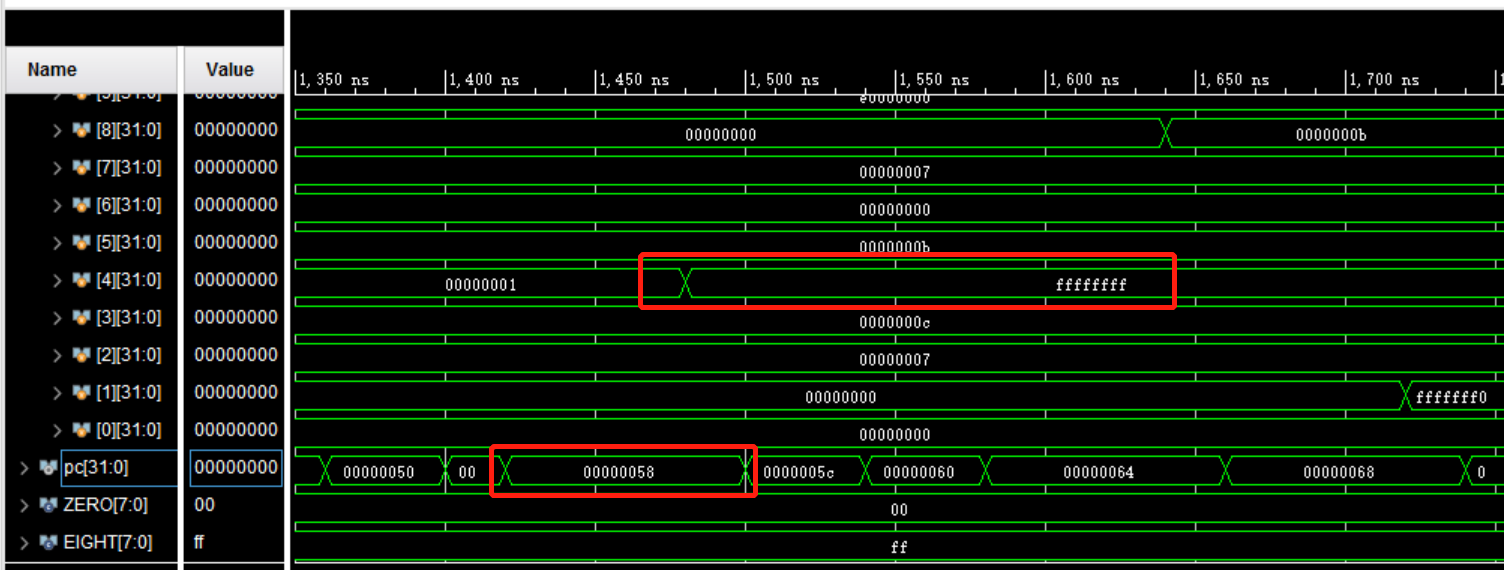


Figure 9 sra: $4 = e0000000 >> 29 = ffffffff

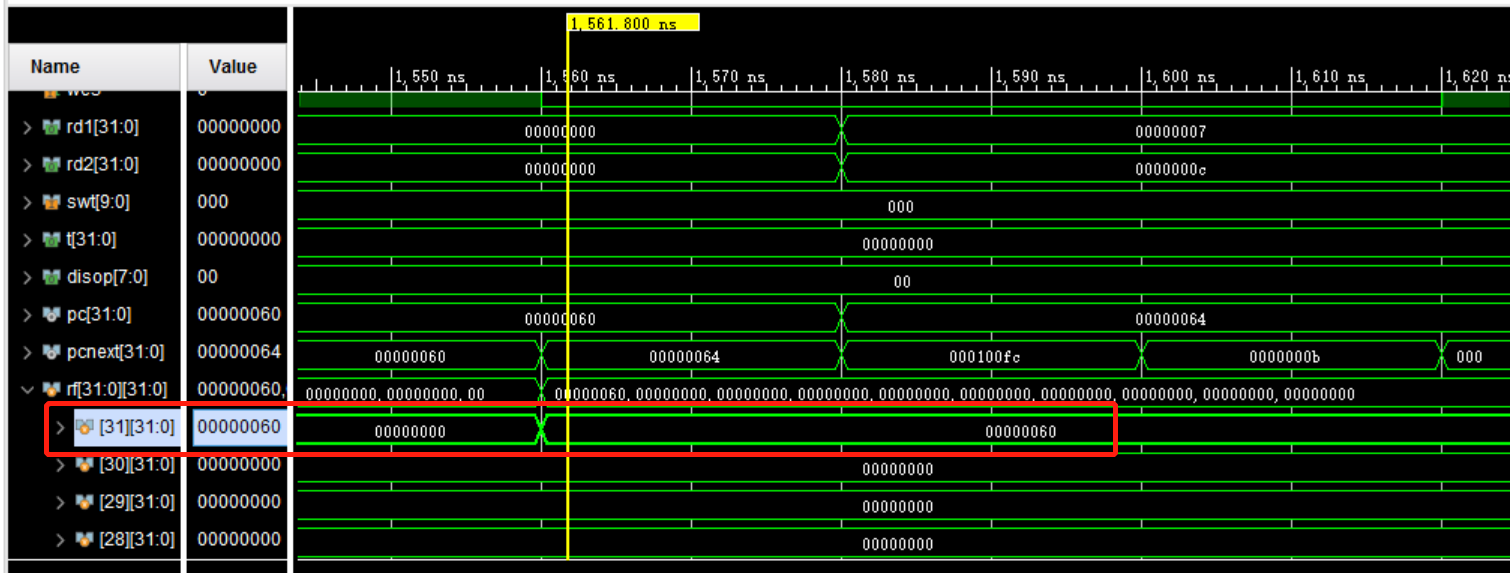


Figure 10 jal: jal 0x60

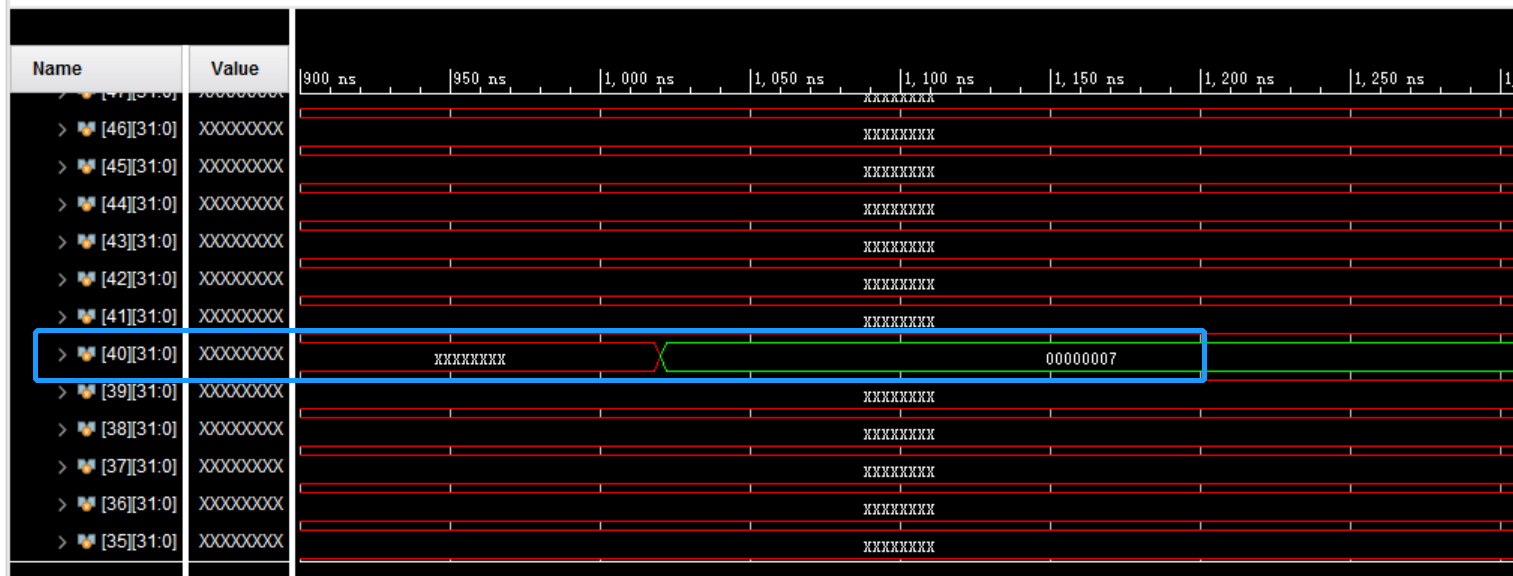


Figure 6 sw: [148 + $3] = $7 = 7

* 1. **仿真2：sortfile0（8个数的插入排序）**

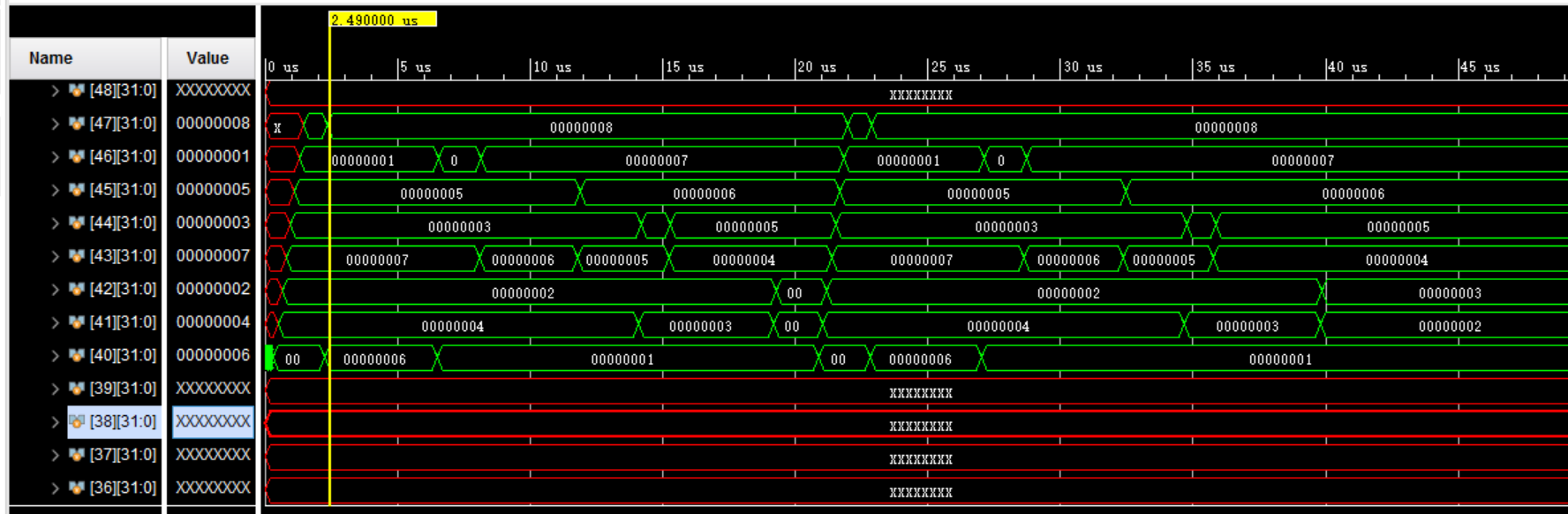


Figure 7 sortfile0验证­­

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）**

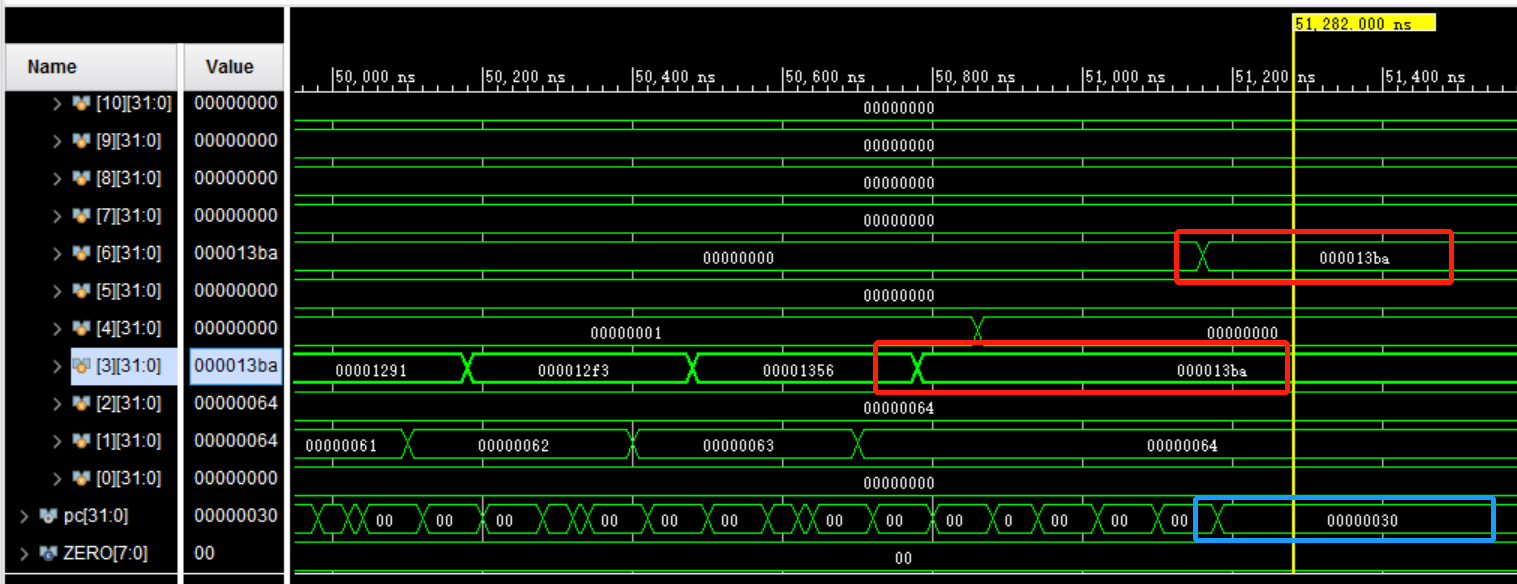


Figure 8 1~100求和

# 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

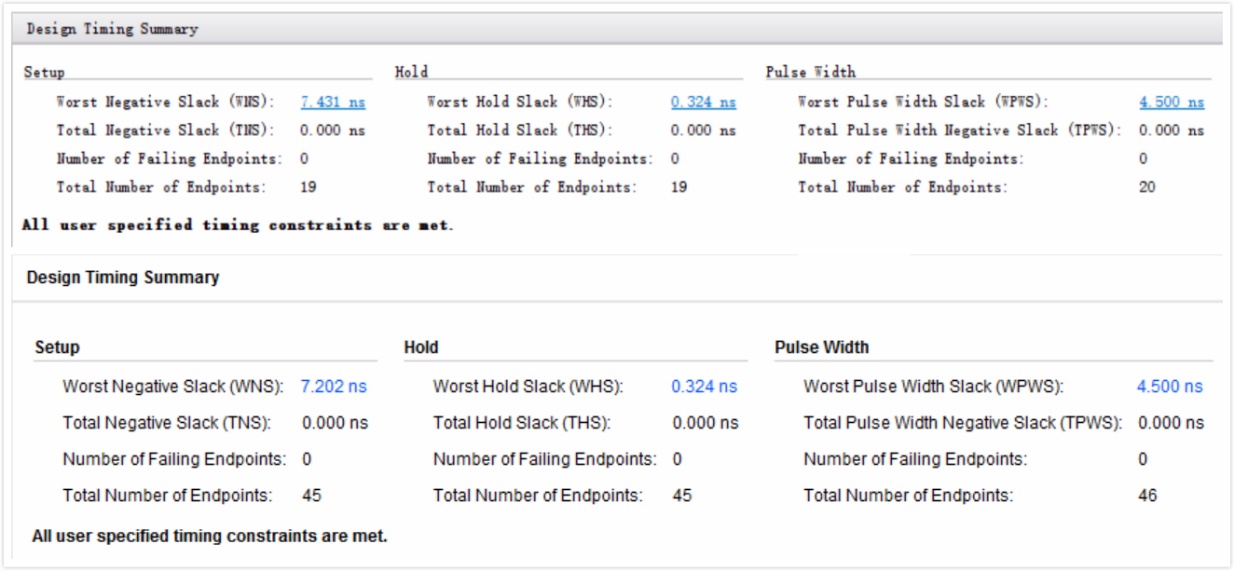


Figure 9 时钟占用（上为单周期，下为多周期）

* 1. 资源占用

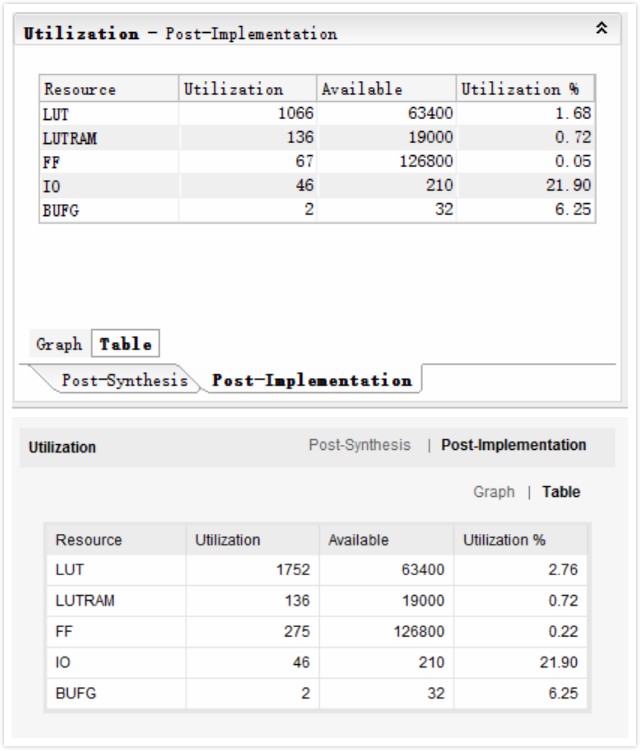


Figure 10 资源占用（上为单周期，下为多周期）

# 实验感想

* 1. 对多周期MIPS处理器的运行机制更加了解，是单周期处理器的升级版，也为之后流水线MIPS处理器的实现打下基础；
  2. 更加熟悉FSM的控制和实现机制，学会设计控制指令的FSM状态；
  3. 对MIPS指令集有了更深的了解，并学会书写MIPS指令程序；
  4. 对verilog语言更加深入；
  5. **能熟练运用开关来实现功能，能使用LED灯以及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。**