综合实验 流水线 MIPS 处理器的设计

无 21 苏创椿 2022010512

一、实验目的

1、 复习 5 级流水线处理器架构，加深对冒险处理单元的理解和认识，进一步学习 MIPS 指令集的组成和指令结构，学习具体指令的数据通路实现，以排序算法为基础练习对多周期流水线进行时序分析，计算最高运行频率和 CPI，从而对 CPU 架构的实现和分析有更深入的认识。

2、 训练通过 vivado 进行设计和分析的能力，加强相应的代码和硬件调试能力。

3、 初步接触通过外设存储器对外设功能进行软件实现的方式，训练汇编语言编写。

二、设计方案

1.解决竞争问题

（1）结构冒险

寄存器堆设计采用“先写后读”设计

（2）数据冒险

寄存器写回冒险：采用 EX/MEM 和 MEM/WB 转发回 ID/EX 的转发模式

load-use 冒险：采用 stull 一个周期并从 MEM/WB 转发的模式

（3）控制冒险

分支指令：在 EX 阶段判断，若分支则取消 IF 和 ID 阶段指令

跳转指令：在 ID 阶段判断， 取消 IF 阶段指令

2.数据空间地址和外设空间地址结构

本实验中 0x0000~0x07FF 为数据空间地址，0x40000010 后 12bit 为外设空间地址，在实现上采用直接将 BCD 和 ano 管线直接传入 DataMemory 模块中，并在对应外设地址内容改变时直接修改对应外设资源管线的值。

3.实现的 mips 指令集指令

根据对应指令的数据通路，得出各个指令的控制信号及控制信号的作用范围（不包括转发控制信号和 ALU 控制信号），将控制信号的生成集成在模块 control.v 中，在 IF 阶段实现。

	PCSrc[1:0]	Branch[2:0]	RegWrite	RegDst[1:0]	MemRead	MemWrite	MemtoReg[1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOP
lw	0	0	1	0	1	0	1	0	1	1	0
sw	0	0	0	x	x	1	x	0	1	1	0
lui	0	0	1	0	x	0	0	0	1	x	1
add	0	0	1	1	x	0	0	0	0	x	x
addu	0	0	1	1	x	0	0	0	0	x	x
sub	0	0	1	1	x	0	0	0	0	x	x
subu	0	0	1	1	x	0	0	0	0	x	x
addi	0	0	1	0	x	0	0	0	1	1	0
addiu	0	0	1	0	x	0	0	0	1	1	0
mul	0	0	1	1	x	0	0	0	0	x	x

and	0	0	1	1	x	0	0	0	0	x	x
or	0	0	1	1	x	0	0	0	0	x	x
xor	0	0	1	1	x	0	0	0	0	x	x
nor	0	0	1	1	x	0	0	0	0	x	x
andi	0	0	1	0	x	0	0	0	1	0	0
sll	0	0	1	1	x	0	0	1	0	x	x
srl	0	0	1	1	x	0	0	1	0	x	x
sra	0	0	1	1	x	0	0	1	0	x	x
slt	0	0	1	1	x	0	0	0	0	x	x
sltu	0	0	1	1	x	0	0	0	0	x	x
slti	0	0	1	0	x	0	0	0	1	1	0
sltiu	0	0	1	0	x	0	0	0	1	1	0
beq	0	1	0	x	x	0	x	0	0	1	0
bne	0	2	0	x	x	0	x	0	0	1	0
blez	0	3	0	x	x	0	x	0	0	1	0
bgtz	0	4	0	x	x	0	x	0	0	1	0
bltz	0	5	0	x	x	0	x	0	0	1	0
j	1	0	0	x	x	0	x	x	x	x	x
jal	1	0	1	2	x	0	2	x	x	x	x
jr	2	0	0	x	x	0	x	x	x	x	x
jalr	2	0	1	2	x	0	2	x	x	x	x

4、各阶段功能示意图

Stage	R-Type	Load	Store	Branch
IF	IR<=Meminst[PC],PC<=PC+4			
ID	Control; Databus1<=Reg[IR[25:21]];Databus2<=Reg[IR[20:16]] Jump:if(JUMP)PC<=JumpTarget EX/MEM.ALUOut			
EX	ALUControl ALUOut<=A op B EX/MEM.ALUOut MEM/WB.Databus3	ALUControl ALUOut<=A+signext(IR[15:0]) EX\MEM.ALUOut MEM/WB.Databus3		Branch: If(A==B) PC<=BranchTarget
MEM		Databus3 <= MemData[ALUOut]	MemData[ALUOut] <=Databus2 MEM\WB.Databus3	
WB	Reg[IR[15:11]] <=ALUOut	Reg[IR[15:11]] <=Databus3		

三、文件清单

多周期流水线文件

CPU.v

Control.v

RegisterFile.v

ALUControl.v

ALU.v

InstructionMemory.v

DataMemory.v

Forward.v

Hazard.v

IF_ID.v

ID_EX.v

EX_MEM.v

MEM_WB.v

分频器

Frequency_divider

仿真文件

test_cpu.v

约束文件

xdc_for_both.xdc

四、模块实现及关键代码

（1）ALU 实现

在 ALUControl 单元，根据指令的 OpCode 及 Funct 生成对应的 ALU 控制信号，经级间寄存器输入到 ALU 模块，ALU 模块根据相应的控制信号进行运算并输出。事实上 ALU 提供的运算如下：AND、OR、ADD、SUB、SLT、NOR、XOR、SLL、SRL、SRA、MUL

```
    // funct number for different operation
    parameter aluAND = 5'b00000;
    parameter aluOR  = 5'b00001;
    parameter aluADD = 5'b00010;
    parameter aluSUB = 5'b00110;
    parameter aluSLT = 5'b00111;
    parameter aluNOR = 5'b01100;
    parameter aluXOR = 5'b01101;
    parameter aluSLL = 5'b10000;
    parameter aluSRL = 5'b11000;
    parameter aluSRA = 5'b11001;
    parameter aluMUL = 5'b11010; //mul
```

```
        case (ALUCtl)
            5'b00000: out <= in1 & in2;
            5'b00001: out <= in1 | in2;
            5'b00010: out <= in1 + in2;
```

```
            5'b00110: out <= in1 - in2;
            5'b00111: out <= {31'h00000000, Sign? lt_signed: (in1 <
in2)};
            5'b01100: out <= ~(in1 | in2);
            5'b01101: out <= in1 ^ in2;
            5'b10000: out <= (in2 << in1[4:0]);
            5'b11000: out <= (in2 >> in1[4:0]);
            5'b11001: out <= ({{32{in2[31]}}, in2} >> in1[4:0]);
            5'b11010: out <= in1 * in2; // mul
            default: out <= 32'h00000000;
        endcase
```

另有输出信号 ZERO、LTZero：

```
    // zero means whether the output is zero or not
    // LTZero means whether the output less then zero or not
    assign zero = (out == 0);
    assign LTZero = out[31];
```

（2）寄存器堆实现

支持同一周期先写后读，提供一个写端口和两个读端口。

先写后读具体实现如下：

```
    // read data from RF_data as Read_data1 and Read_data2
    assign Read_data1 = (Read_register1 == 5'b00000) ? 32'h00000000 :
                        (RegWrite && (Read_register1 ==
Write_register)) ? Write_data :
                        RF_data[Read_register1];
    assign Read_data2 = (Read_register2 == 5'b00000) ? 32'h00000000 :
                        (RegWrite && (Read_register2 ==
Write_register)) ? Write_data :
                        RF_data[Read_register2];
```

（3）存储器实现

将地址 32’h40000010 设为外设地址空间，32’h0 开始到外设地址前为数据存储地址空间

```
    assign ano = Output_RAM_data[11:8];
    assign BCD = Output_RAM_data[7 :0];

always @(posedge reset or posedge clk)begin
        if (reset) begin
            // -------- Paste Data Memory Configuration Below (Data-
q1.txt)
           //数据初始化，详见源代码

            // -------- Paste Data Memory Configuration Above
        end
        else if (MemWrite) begin
            if(Address==32'h40000010) begin
                Output_RAM_data <= Write_data;
```

```
            end
            else begin
                RAM_data[Address[RAM_SIZE_BIT + 1:2]] <= Write_data;
            end
        end
    end
```

(4)转发单元及 Hazard 实现

转发单元 Forward 实现如下

```
    always @(*) begin
        //EX/MEM hazard
        if(MEM_RegWrite&&(MEM_Write_register!=0)&&(MEM_Write_register==EX_RegisterRs))
            ForwardA=2'b10;
        //MEM/WB hazard
        else if(WB_RegWrite&&(WB_Write_register!=0)&&(WB_Write_register==EX_RegisterRs)&&(MEM_Write_register!=EX_RegisterRs||~MEM_RegWrite))
            ForwardA=2'b01;
        else
            ForwardA=2'b00;
        //EX/MEM hazard
        if(MEM_RegWrite&&(MEM_Write_register!=0)&&(MEM_Write_register==EX_RegisterRt))
            ForwardB=2'b10;
        //MEM/WB hazard
        else if(WB_RegWrite&&(WB_Write_register!=0)&&(WB_Write_register==EX_RegisterRt)&&(MEM_Write_register!=EX_RegisterRt||~MEM_RegWrite))
            ForwardB=2'b01;
        else
            ForwardB=2'b00;
    end
```

Hazard(用于 load-use 冒险产生 stall)实现如下

```
//Load-use hazard
    always @(*)begin
        if(EX_MemRead&&((Write_register==ID_registerRs)||(Write_register==ID_registerRt)))
            stall<=1'b1;
        else
            stall<=1'b0;
    end
```

五、汇编指令代码

采用插入排序，软件实现显示。

软件显示思路为先将数码管译码相关内容存于内存中未用到的地址中，译码时通过读取存储中内容来完成索引
汇编代码如下：

```
.text
main:
    addi $s0 $zero 0
    addi $t2 $zero 4
    lw $s1 0($s0) #读出N存于s1
    addi $t4 $s1 -1 #t4初始化为N-1，作为循环loop的计数器
    move $t6 $s1 #t6初始化为N，作为循环for的计数器
    addi $s0 $s0 4   #s0指向需要排序的第一个数
    #执行排序
    insertion_sort:
        #move $t2 $zero #初始化比较次数
        #读出v[1]
        addi $s0 $s0 4  #s0指向需要排序的下一个数
        addi $s2 $t2 0  #记录第一个数据位置
        addi $s3 $t2 4 #s3记录已排数据末尾位置
        #循环执行查找的同时插入
        loop: #比较、后移、插入，均在loop中完成
            #读出新的数
            lw $t0 0($s0)
            addi $s0 $s0 4  #s0指向需要排序的下一个数
            search:
                lw $t1 0($t2)  #读出比较数
                blt $t0 $t1 proceed   #比较大小
                addi $t2 $t2 4
                insert: #插入
                    sw $t0 0($t2)
                    addi $t2 $s3 0#t2重新指回栈底
                    addi $s3 $s3 4
                    j loopend
                proceed: #后移
                    sw $t1 4($t2)
                    beq $s2 $t2 insert
                    addi $t2 $t2 -4
                    j search
            #判断循环是否结束
            loopend:
                addi $t4 $t4 -1 #计数器减1
                bnez $t4 loop #如果计数器不为0循环继续

    output:
        addi $a0 $zero 100 #译码地址
```

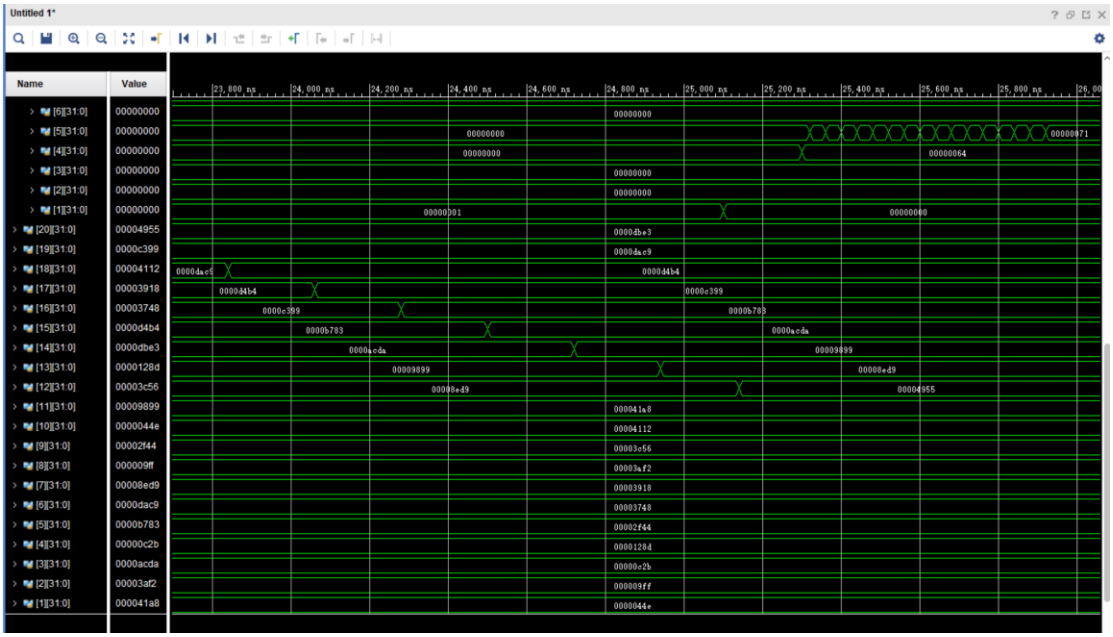
```
        addi $a1 $zero 63  #0
        sw $a1 0($a0)
        addi $a1 $zero 6  #1
        sw $a1 4($a0)
        addi $a1 $zero 91  #2
        sw $a1 8($a0)
        addi $a1 $zero 79  #3
        sw $a1 12($a0)
        addi $a1 $zero 102  #4
        sw $a1 16($a0)
        addi $a1 $zero 109  #5
        sw $a1 20($a0)
        addi $a1 $zero 125  #6
        sw $a1 24($a0)
        addi $a1 $zero 7  #7
        sw $a1 28($a0)
        addi $a1 $zero 127  #8
        sw $a1 32($a0)
        addi $a1 $zero 111  #9
        sw $a1 36($a0)
        addi $a1 $zero 119  #A
        sw $a1 40($a0)
        addi $a1 $zero 124  #b
        sw $a1 44($a0)
        addi $a1 $zero 57  #c
        sw $a1 48($a0)
        addi $a1 $zero 94  #d
        sw $a1 52($a0)
        addi $a1 $zero 121  #e
        sw $a1 56($a0)
        addi $a1 $zero 113  #F
        sw $a1 60($a0)

        addi $t0 $zero 0
        lui $t5 16384
        addi $t5 $t5 16 #oX40000010
    for:
        addi $t0 $t0 4
        addi $s2 $zero 250
        one_s:
            lw $t1 0($t0)
            #最高位
            srl $t2 $t1 12
            sll $t2 $t2 2
```

```
        add $t3 $t2 $a0
        lw $t4 0($t3)
        addi $t4 $t4 2048  #0X 1000 0000 0000
        sw $t4 0($t5)
        addi $t7 $t7 12500
        one_ms_1:
            addi $t7 $t7 -1
        bnez $t7 one_ms_1
        #次高位
        sll $t2 $t1 20
        srl $t2 $t2 28
        sll $t2 $t2 2
        add $t3 $t2 $a0
        lw $t4 0($t3)
        addi $t4 $t4 1024  #0X 0100 0000 0000
        sw $t4 0($t5)
        addi $t7 $t7 12500
        one_ms_2:
            addi $t7 $t7 -1
        bnez $t7 one_ms_2
        #次低位
        sll $t2 $t1 24
        srl $t2 $t2 28
        sll $t2 $t2 2
        add $t3 $t2 $a0
        lw $t4 0($t3)
        addi $t4 $t4 512  #0X 0010 0000 0000
        sw $t4 0($t5)
        addi $t7 $t7 12500
        one_ms_3:
            addi $t7 $t7 -1
        bnez $t7 one_ms_3
        #最低位
        sll $t2 $t1 28
        srl $t2 $t2 28
        sll $t2 $t2 2
        add $t3 $t2 $a0
        lw $t4 0($t3)
        addi $t4 $t4 256  #0X 0001 0000 0000
        sw $t4 0($t5)
        addi $t7 $t7 12500
        one_ms_4:
            addi $t7 $t7 -1
        bnez $t7 one_ms_4
```

```
        addi $s2 $s2 -1
        bnez $s2 one_s
    addi $t6 $t6 -1
    bnez $t6 for
```

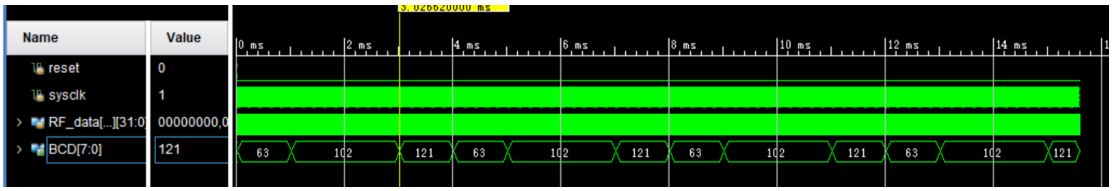
六、仿真结果及分析

1.排序



排序数据储存在内存地址 4’h8-4’h84 中（即 RAM_data[2]-RAM_data[21]),从仿真波形可以看出数据成功按照插入排序的方式进行排序重新后写入对应地址。

2、软件实现 led 显示



图中数为十进制，对应显示的数字字母可对照汇编代码的译码部分

排序完成后第一个数据应该为 0x044e，

对应的最低位为 e，对应 BCD 二进制转为十进制的数字为 121，

第二位和第三位为 4，对应 BCD 二进制转为十进制的数字为 102，

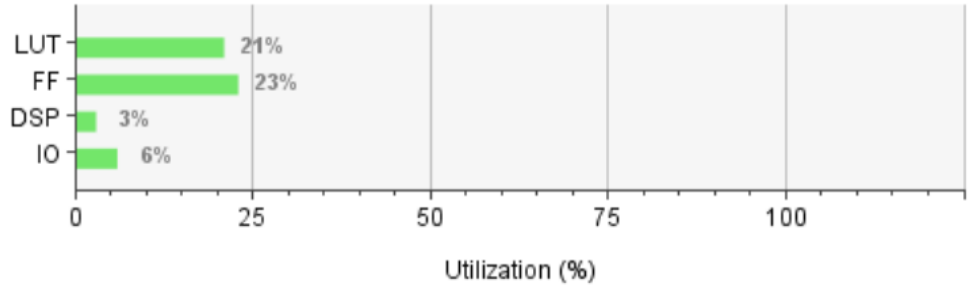
最高位为 0，对应 BCD 二进制转为十进制的数字为 63，

从仿真波形可以看出实现了从高到低位逐位显示，也可看出每位显示约 1ms

七、CPI 计算

1、逻辑资源与面积占用

Resource	Utilization	Available	Utilization %
LUT	4272	20800	20.54
FF	9675	41600	23.26
DSP	3	90	3.33
IO	14	250	5.60



Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT Flip Flop Pairs (20800)	DSPs (90)	Bonded IOB (250)	BUFGCTRL (32)
CPU	4272	9675	1152	384	3931	4272	218	3	14	2
alu1 (ALU)	464	0	0	0	133	464	0	3	0	0
alu_control1 (ALUCont...	10	0	0	0	4	10	0	0	0	0
control1 (Control)	23	0	0	0	9	23	0	0	0	0
data_memory1 (DataM...	2621	8204	1024	320	3264	2621	0	0	0	0
EX_MEM1 (EX_MEM)	0	106	0	0	63	0	0	0	0	0
f_divider1 (f_divider)	1	1	0	0	1	1	1	0	0	1
Forward1 (Forward)	13	0	0	0	7	13	0	0	0	0
Hazard1 (Hazard)	5	0	0	0	2	5	0	0	0	0
ID_EX1 (ID_EX)	88	172	0	0	58	88	87	0	0	0
IF_ID1 (IF_ID)	33	64	0	0	17	33	32	0	0	0
instruction_memory1 (I...	79	0	0	0	27	79	0	0	0	0
MEM_WB1 (MEM_WB)	0	104	0	0	47	0	0	0	0	0
register_file1 (Register...	692	992	128	64	490	692	0	0	0	0

2、时序性能与最大时钟频率

重新创建一个 vivado 项目，删除分频代码，进行时序分析

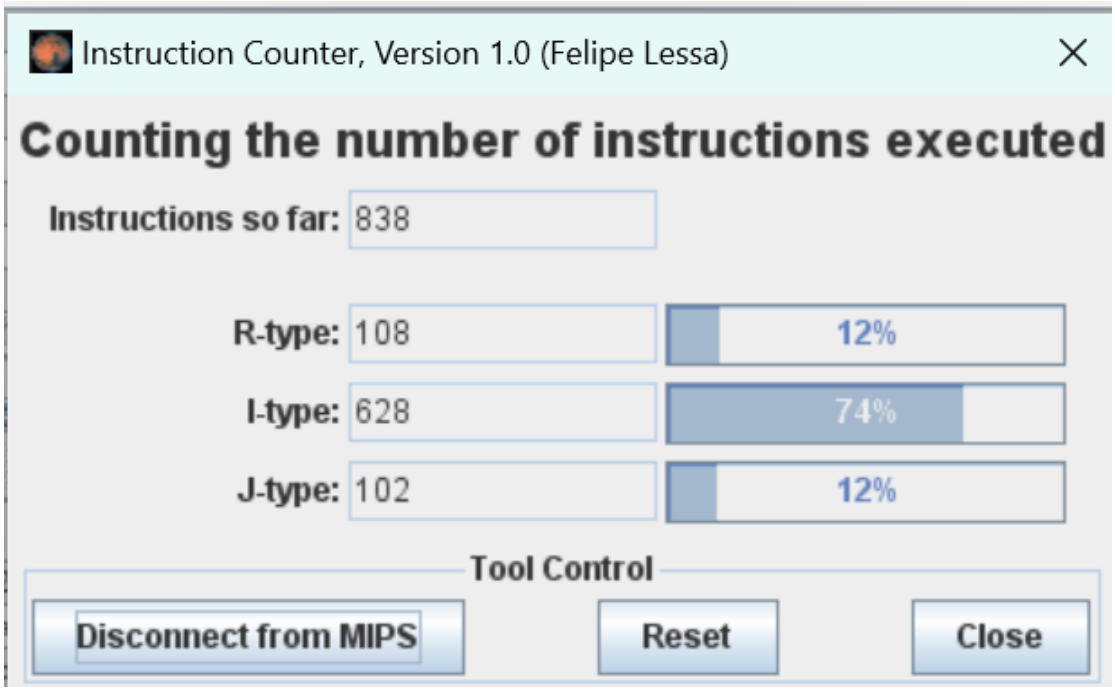
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-5.218 ns	Worst Hold Slack (WHS):	0.168 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	-2412.672 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	672	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	19134	Total Number of Endpoints:	19134	Total Number of Endpoints:	9675

Timing constraints are not met.

$$f_{max} = \frac{1}{T - WNS} = \frac{1}{10ns + 5.218ns} = 65.45MHz$$

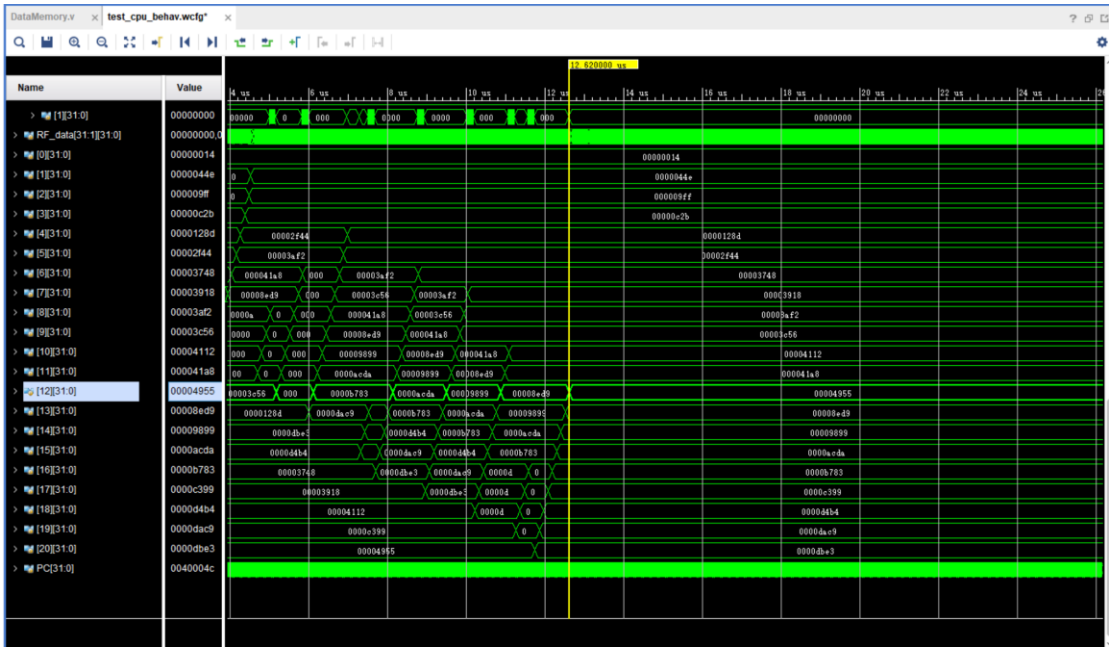
3、CPI 计算

由于其他部分过于庞杂，计算 CPI 时仅计算排序部分



减去加载文件指令 15 条实际应为 823 条指令

在 vivado 仿真得到指令数



所用时钟周期数为$\frac{12620000}{10000}=1262$，因此$CPI=\frac{1262}{823}=1.533$

八、硬件调试及实验总结

第一次成功烧录后，发现数据刷新过快，但仿真时明明没问题，检查发现仿真文件中用的时间刻度与实际不一致，且由于一开始没计算直接按仿真来同时导致了汇编代码的问题，于是重新计算计算显示 1s 实际需要的周期数，更改汇编代码，才最终成功完成要求的效果，经检查数字字母的显示以及排序结果的顺序也没有问题。至此，连续几天的流水线之旅也就进入了尾声。

回顾完成这次实验的整个过程，一连几天全身心地投入，我感觉到自己对流水线有了更

深的理解，verilog 代码的编写和 vivado 的操作也愈发熟练，第一次完成如此大工作量的任务，我感到自豪。当然过程中少不了困难和挫折，但这也让我的心态得到了磨砺。总而言之，这次实验对我甚至是对大部分同学来说都是意义非凡且收获颇丰的。