**计算机组成原理**

**课程设计报告**

**学 号\_\_\_\_\_\_\_\_20074214\_\_\_\_\_\_\_\_\_\_**

**姓 名\_\_\_\_\_\_\_\_周子渔\_\_\_\_\_\_\_\_\_\_\_\_**

**指导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_\_\_\_\_\_2022.7.2\_\_\_\_\_\_\_\_\_\_\_\_**

**成绩评价表**

|  |  |  |
| --- | --- | --- |
| **报告内容** | **报告结构** | **报告最终成绩** |
| **□丰富正确**  **□基本正确**  **□有一些问题**  **□问题很大** | **□完全符合要求**  **□基本符合要求**  **□有比较多的缺陷**  **□完全不符合要求** |  |
| **报告与Project功能一致性** | **报告图表** | **总体评价** |
| **□完全一致**  **□基本一致**  **□基本不一致** | **□符合规范**  **□基本符合规范**  **□有一些错误**  **□完全不正确** |  |

**教师签字:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

# 模块定义

**（1）顶层模块定义**

1）基本描述

主要功能是完成取指令功能。IFU内部包括PC、IM(指令存储器)以及其他相关逻辑。IFU除了能执行顺序取值令外，还能根据BEQ指令的执行情况决定顺序取值令还是转移取值令。

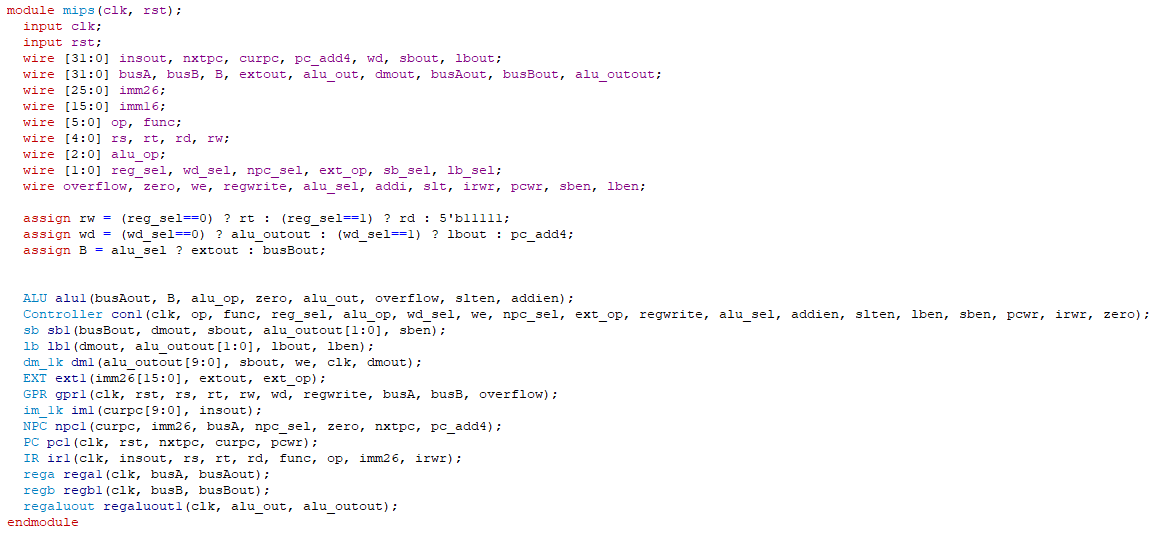
2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号。  1：复位  0：无效 |

3）功能定义

连接所有模块。

4）程序截图



**（2）GPR模块定义**

1）基本描述

GPR包括了所有的寄存器，有两个读端口和一个写端口。寄存器的读输出总是对应于读寄存器号，不需要其他控制信号。但是写寄存器必须明确写使能控制信号。注意写操作是边沿触发的，所以所有的写操作的输入（要写的内容、寄存器号、写控制信号）必须在时钟边沿有效。因为寄存器的写入是边沿触发的，故可以在同一时钟周期内读出和写入同一寄存器：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。寄存器号的输入都是5位的，数据线为32位。

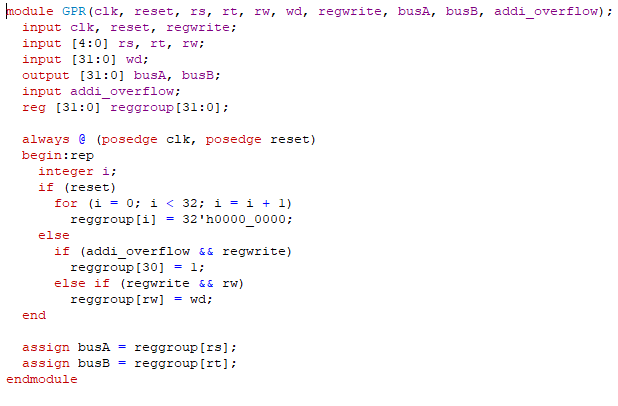
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号 |
| rs | I | rs寄存器号 |
| rt | I | rt寄存器号 |
| rw | I | rw寄存器号 |
| regwrite | I | 寄存器写信号。 0：寄存器写使能无效 1：写使能有效 |
| wd | O | 写入的内容 |
| busA | O | rs寄存器内容 |
| busB | O | rt寄存器内容 |
| addi\_overflow | I | addi是否溢出信号，0为不溢出，1为溢出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | 根据读寄存器的地址从寄存器中读数据，输出 |
| 2 | 写寄存器 | 如果寄存器的写使能有效，且在clk时钟信号触发边沿，则根据寄存器地址，将要写的数据写入指定地址的寄存器当中。读寄存器与写寄存器可以同时进行：读操作读出以前写入的内容，而写入的内容在下一时钟周期才可读。 |

（4）程序截图



**（3）ALU模块定义**

1）基本描述

ALU算数逻辑单元，是计算机的核心，在本次处理器设计中，ALU可以执行的算数运算包括加法、减法，可以执行的逻辑运算包括或操作。

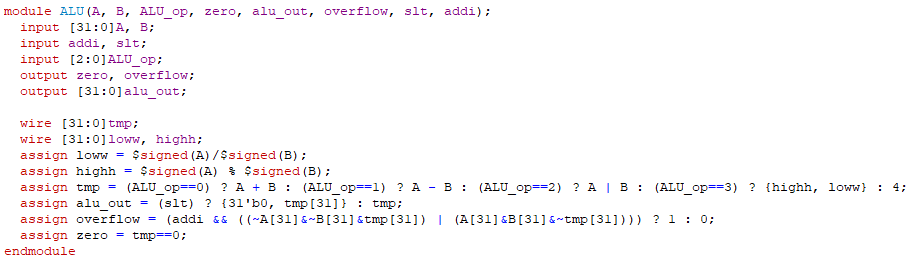
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A | I | 参与运算的输入数据1 |
| B | I | 参与运算的输入数据2 |
| ALU\_op | I | ALU控制信号  Switch (ALUop)  00: alu\_out = A + B  01: alu\_out = A - B  10: alu\_out = A || B |
| alu\_out | O | ALU运算结果 |
| zero | O | 运算结果是否为0的标志位，用于beq相等则分支指令 |
| overflow | O | 判断addi是否溢出 |
| slt | I | 是否为slt指令 |
| addi | I | 是否为addi指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | 若ALUop = 00，则执行加运算。把两个32-bit的输入数据相加，然后将结果Result输出。在本实验中，不支持溢出判断。 |
| 2 | 减运算 | 若ALUop = 01，则执行减运算。把两个32-bit的输入数据相减，然后将结果Result输出。 |
| 3 | 按位或 | 若ALUop = 10，则执行按位或运算。把两个32-bit的输入按位或，然后将结果Result输出。 |

（4）程序截图



**（4）EXT模块定义**

1）基本描述

EXT是符号拓展单元，有一个16位的输入，符号拓展为32位后输出。在lw取数、sw存数、beq相等则分支等指令中的立即数字段包含一个无符号/有符号二进制补码表示的16位数。为了将这个立即数字段加到一个32位的寄存器，计算机必须将这个16位的数转换成数值上相等的32位的数。

对于有符号数来说，这种方法就是将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。

对无符号数来说，将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。

对于lui指令，需要加载至高16位，将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。

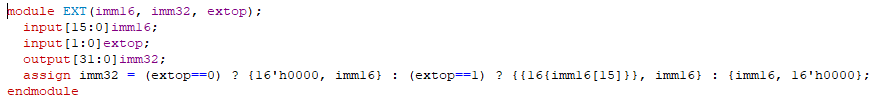
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm16 | I | 16位立即数 |
| extop | I | 符号拓展控制信号  Switch (EXTOp)  00: imm32 = unsign 16 to 32 extend  01: imm32 = sign 16 to 32 extend  10: imm32 = extend to high 16 bit |
| imm32 | O | 符号拓展结果输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 有符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，其最高有效位（符号位）则以复制的方式填满新数的高16位，实现符号拓展。 |
| 2 | 无符号拓展 | 将原有的16位数简单赋值到32位新数的低16位，用0填满新数的高16位。 |
| 3 | 拓展至高16位 | 将原有的16位数简单赋值到32位新数的高16位，低16位用0填充。 |

（4）程序截图



**（5）IM模块定义**

1）基本描述

指令寄存器

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 参与运算的输入数据1 |
| dout | O | 输出指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读指令 | 根据地址Addr从相应位置读出数据，dout为数据输出 |

**（6）DM模块定义**

1）基本描述

数据寄存器

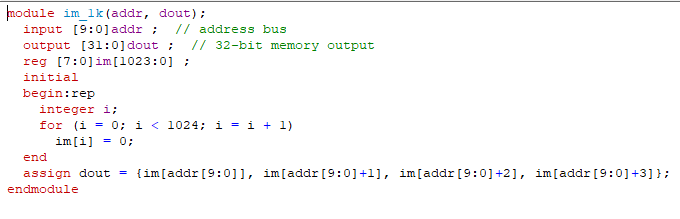
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr | I | 参与运算的输入数据1 |
| din | I | 输出 |
| dout | O | 输出 |
| we | I | 写使能信号 |
| clk | I | 时钟信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 根据地址addr从相应位置读出数据，dout为数据输出 |
| 2 | 写数据 | 如果we写使能有效，根据地址addr将数据din写入存储器的相应位置 |

（4）程序截图



**（7）PC模块定义**

1）基本描述

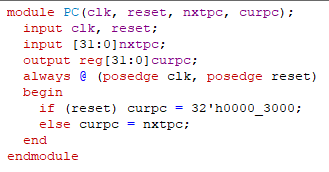
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| curpc | O | 当前pc |
| reset | I | 复位信号 |
| nxtpc | I | 下一个pc |
| clk | I | 时钟信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 改pc | 将当前pc：curpc改为下一条pc：nxtpc |

（4）程序截图



**（8）NPC模块定义**

1）基本描述

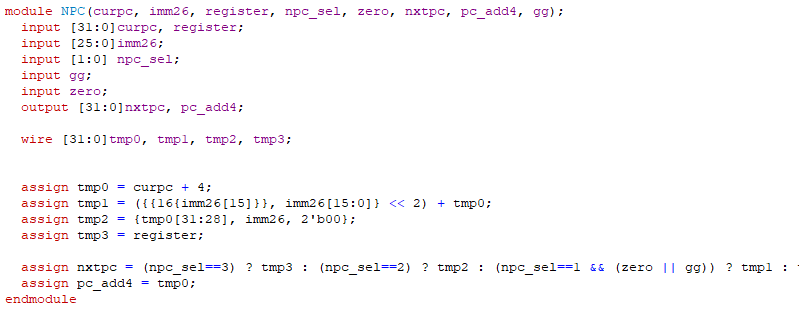
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| curpc | I | 当前pc |
| imm26 | I | 26位立即数 |
| register | I | jr的寄存器输入 |
| npc\_sel | I | npc功能选择  0：pc+4  1：beq  2：j |
| zero | O | 零信号 |
| nxtpc | O | 下一个pc |
| pc\_add4 | O | jal的存储的pc+4 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条pc | 根据npc\_sel计算下一条pc |

（4）程序截图



**（9）sb模块定义**

1）基本描述

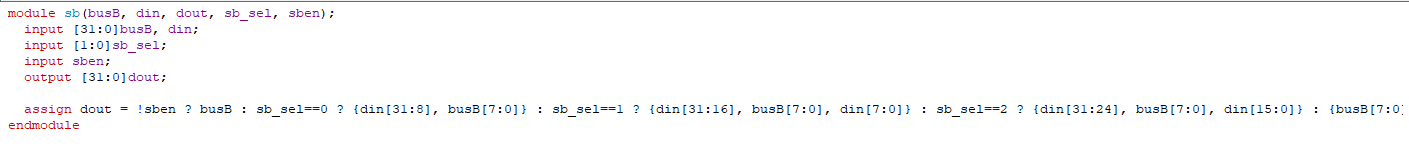
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| busB | I | 输入 |
| din | I | 取出的值 |
| dout | O | 拼接后的输出 |
| sb\_sel | I | sb选择 |
| sben | I | sb使能 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 存byte | 存byte |

（4）程序截图



**（10）lb模块定义**

1）基本描述

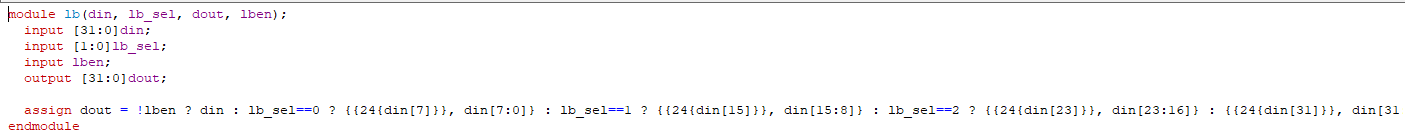
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| busB | I | 输入 |
| din | I | 取出的值 |
| dout | O | 拼接后的输出 |
| lb\_sel | I | sb选择 |
| lben | I | sb使能 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取byte | 取byte |

（4）程序截图



**（10）IR模块定义**

1）基本描述

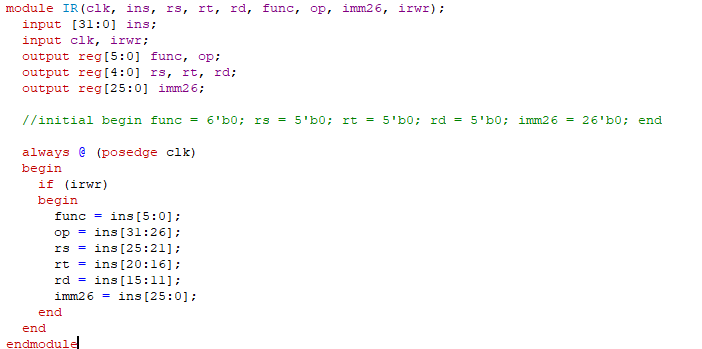
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| ins | I | 指令 |
| rs | O | rs寄存器 |
| rt | O | rt寄存器 |
| rd | O | rd寄存器 |
| func | O | 功能 |
| op | O | op |
| imm26 | O | 26位立即数 |
| irwr | I | 使能 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取指令 | 取指令 |

（4）程序截图



**（12）Controller模块定义**

1）基本描述

Controller是控制单元。控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。

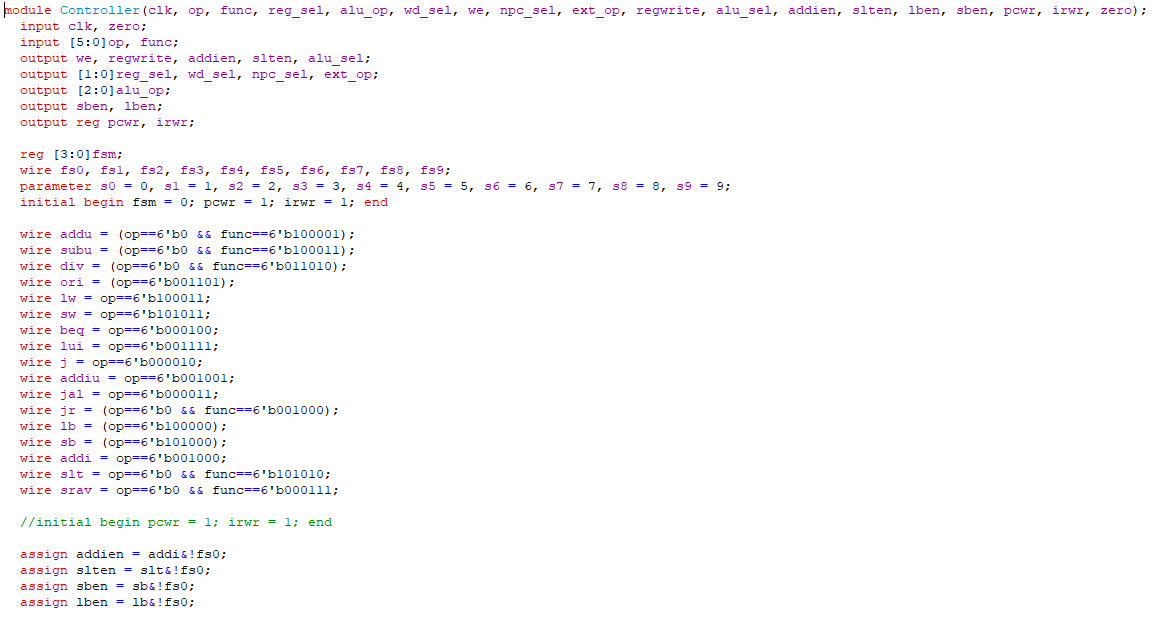
（2）模块接口

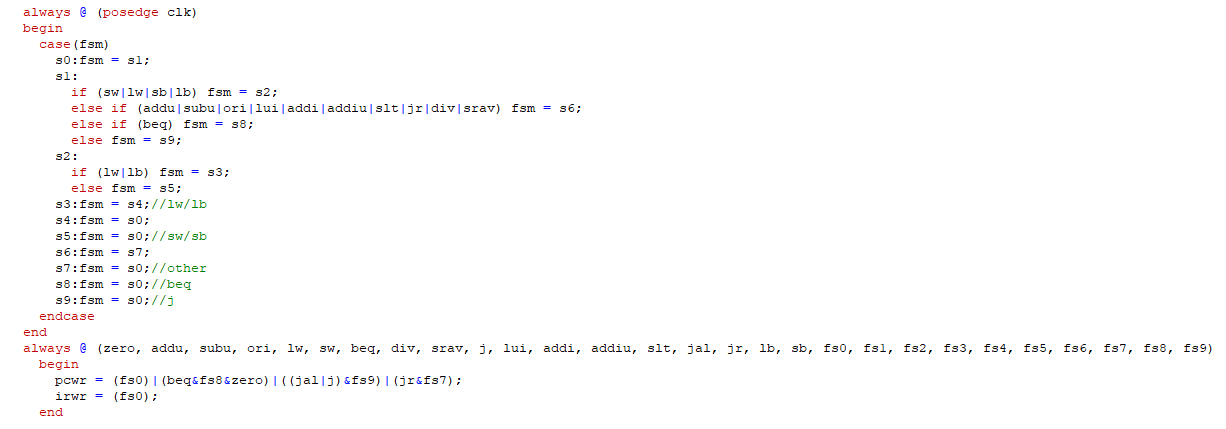
|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| func | I | 指令的func字段 |
| op | I | 指令的opcode字段 |
| reg\_sel | O | 寄存器选择 |
| wd\_sel | O | 写入数据选择 |
| alu\_op | O | 控制ALU执行的运算 |
| we | O | 数据寄存器写使能 |
| npc\_sel | O | npc选择 |
| ext\_op | O | ext选择 |
| regwrite | O | 寄存器写使能 |
| alu\_sel | O | 第二个ALU操作数的来源 |
| addi | O | 是否为addi |
| slt | O | 是否为slt |

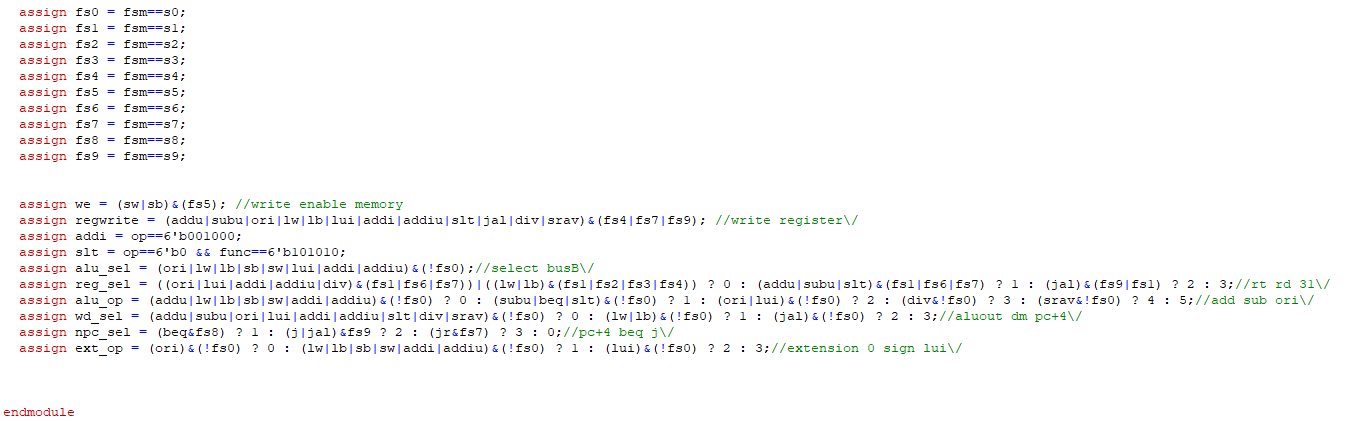
（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 产生控制信号 | 控制单元以指令为输入，能够产生每个状态单元的写信号，每个多选器的选择信号和ALU的控制信号。 |

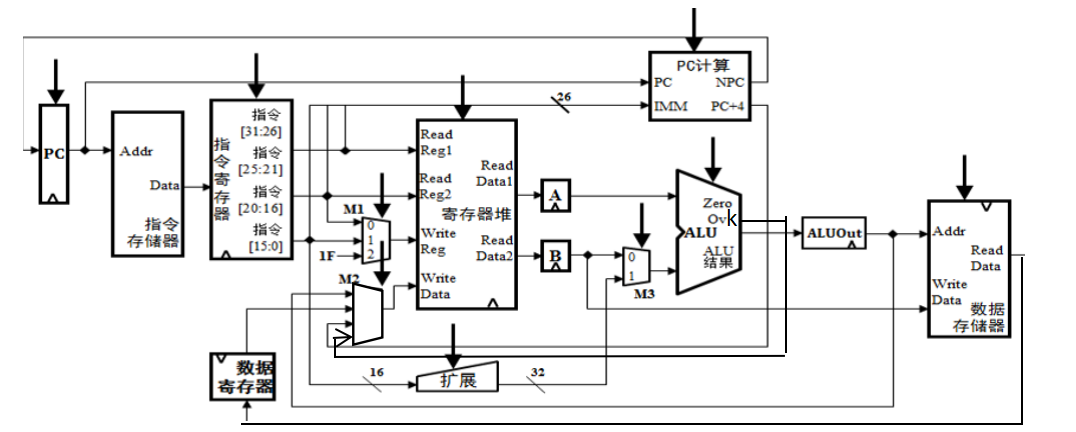
（4）程序截图







二、通路设计



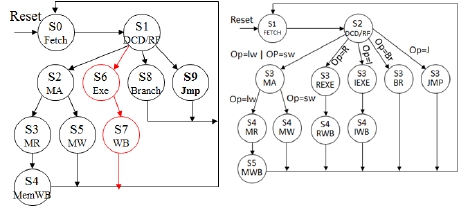
**在p1通路的基础上，将slt放进alu**

**同时加入lb sb两个模块分别接在alu dm来完成lb和sb指令**

三、指令集说明及状态机描述

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令名称 | 操作码opcode | 功能码funct | 功能 | 指令功能描述 |
| addu | 000000 | 100001 | 无符号数加 | R[rd]🡨R[rs]+R[rt] |
| subu | 000000 | 100011 | 无符号减 | R[rd]🡨R[rs]-R[rt] |
| slt | 000000 | 101010 | 小于时置位 | R[rd] ← (GPR[rs] < GPR[rt]) |
| addi | 001000 |  | 立即数加法（支持溢出） | 未溢出时：R[rt]🡨R[rs]+immdiate  溢出时：不改变寄存器内的值 |
| addiu | 001001 |  | 立即数加法（不支持溢出） | R[rt]🡨R[rs]+immdiate |
| ori | 001101 |  | 立即数或 | R[rt]🡨R[rs]|ZeroExt(imm16) |
| lw | 100011 |  | 取字 | R[rt]🡨MEM{R[rs]+SignExt[imm16]} |
| sw | 101011 |  | 存字 | MEM{R[rs]+SignExt[imm16]}🡨R[rt] |
| beq | 000100 |  | 相等时跳转 | if(R[rs]==R[rt]) then PC🡨PC+4+(signExt(imm16)||00) |
| lui | 001111 |  | 取立即数的高位 | R[rt]🡨imm16||016 |
| j | 000010 |  | 无条件跳转 | PC🡨PC ← PC+4[31..28] || instr || 02 |
| jal | 100000 |  | 跳转并链接 | $31=pc+4 |
| jr | 000000 | 001000 | 跳转至寄存器所指位置 | PC ← R[rs] |
| sb | 101000 |  | 字节存入存储器的相应字节位置 | memory[GPR[base] + offset] ←GPR[rt] |
| lb | 100000 |  | 取出存储器相应字节 | GPR[rt] ←memory[GPR[base] + offset] |
| bltz | 000001 |  | 当GPR[rs]小于0跳转执行分支 | if GPR[rs] < 0 then branch |

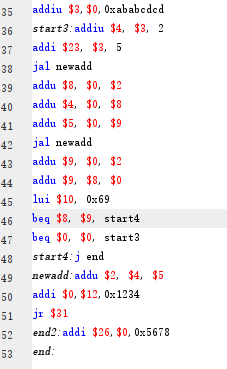
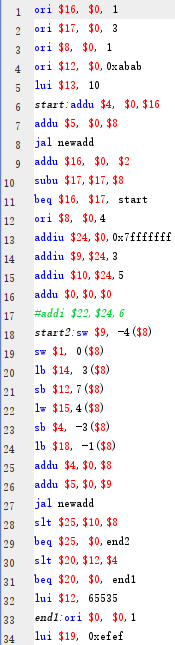
|  |  |
| --- | --- |
| 指令 | 状态 |
| addu | s0->s1->s6->s7->s0 |
| subu | s0->s1->s6->s7->s0 |
| slt | s0->s1->s6->s7->s0 |
| addi | s0->s1->s6->s7->s0 |
| addiu | s0->s1->s6->s7->s0 |
| ori | s0->s1->s6->s7->s0 |
| lw | s0->s1->s2->s3->s4->s0 |
| sw | s0->s1->s2->s5->s0 |
| lb | s0->s1->s2->s3->s4->s0 |
| sb | s0->s1->s2->s5->s0 |
| beq | s0->s1->s8->s0 |
| lui | s0->s1->s6->s7->s0 |
| j | s0->s1->s9->s0 |
| jal | s0->s1->s9->s0 |
| jr | s0->s1->s6->s7->s0 |



|  |  |
| --- | --- |
| 状态名 | 编号 |
| S0 | 0000 |
| S1 | 0001 |
| S2 | 0010 |
| S3 | 0011 |
| S4 | 0100 |
| S5 | 0101 |
| S6 | 0110 |
| S7 | 0111 |
| S8 | 1000 |
| S9 | 1001 |

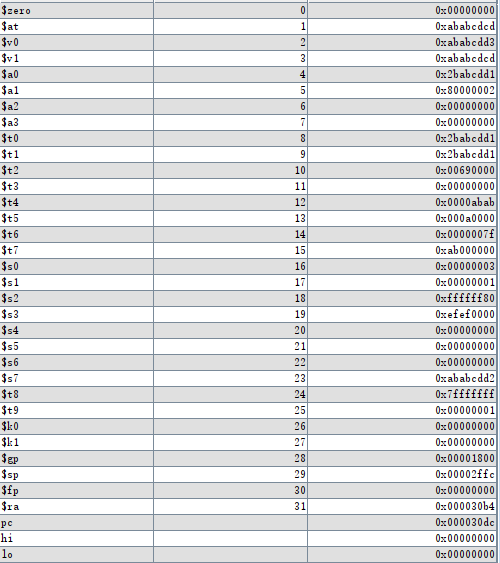
# 四、程序测试

**1.测试程序**

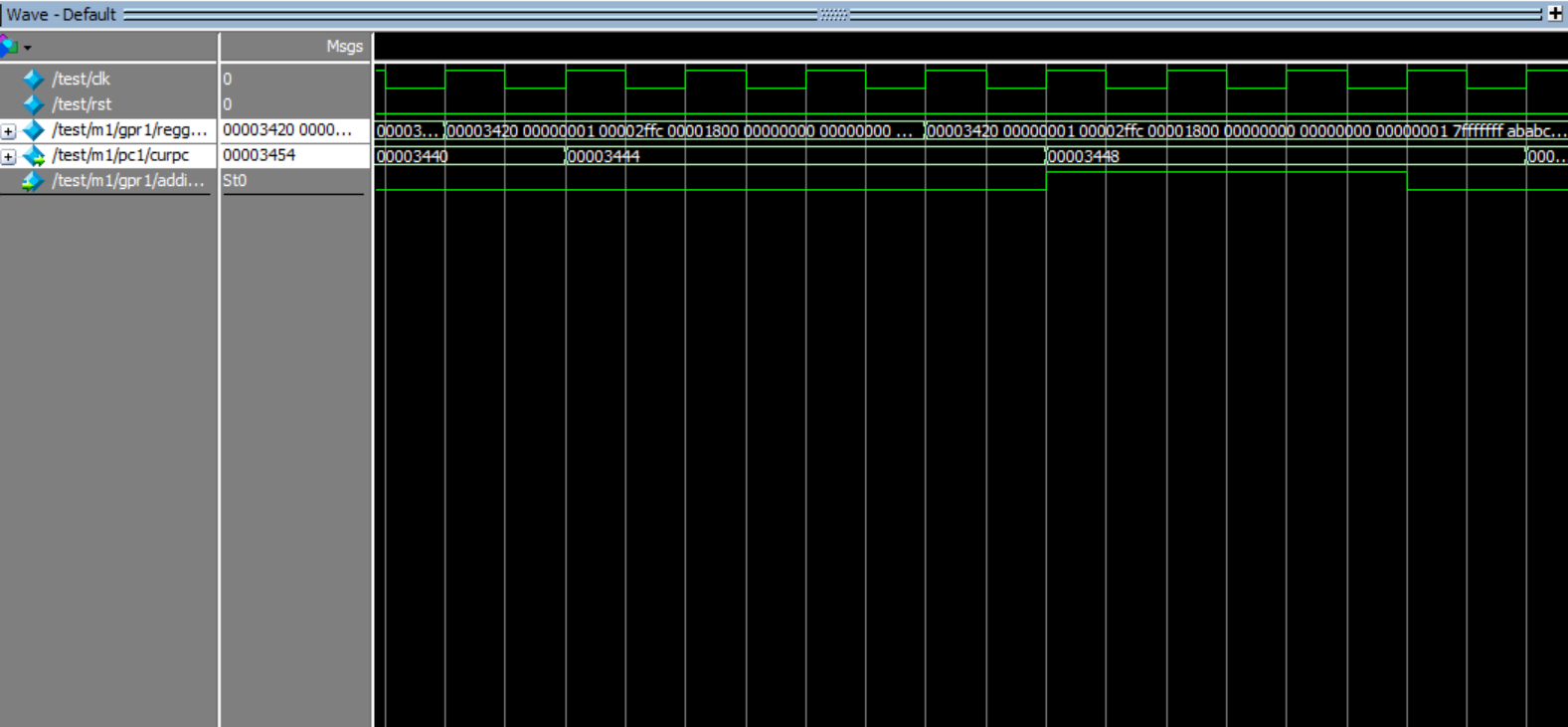


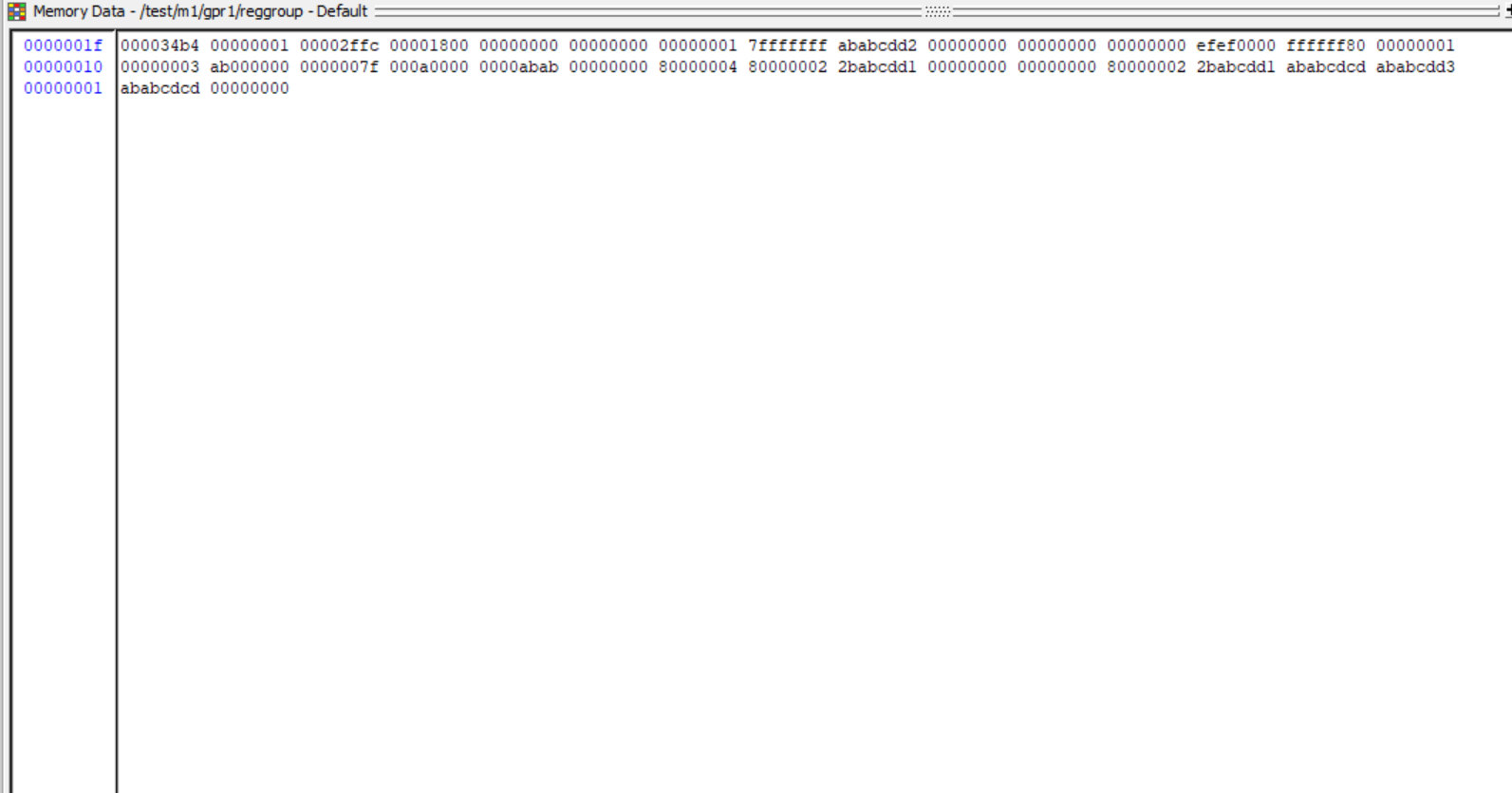
|  |  |  |
| --- | --- | --- |
| 机器码 | 指令 | 注释 |
| 34100001 | ori $16, $0, 1 | #将1和$0内容做或运算放入$16 |
| 34110003 | ori $17, $0, 3 | #将3和$0内容做或运算放入$17 |
| 34080001 | ori $8, $0, 1 | #将1和$0内容做或运算放入$8 |
| 340cabab | ori $12, $0,0xabab | #将0xabab和$0内容做或运算放入$12 |
| 3c0d000a | lui $13, 10 | #将10放到$13中的高16位，低16位补0 |
| 00102021 | start: addu $4, $0,$16 | #start：将$0内容和$16内容无符号相加放入$4 |
| 00082821 | addu $5, $0,$8 | #将$0内容和$8内容无符号相加放入$5 |
| 0c000c32 | jal newadd | #将下一条指令的地址放入$31并跳转到newadd |
| 00028021 | addu $16, $0, $2 | #将$0内容和$2内容无符号相加放入$16 |
| 02288823 | subu $17,$17,$8 | #将$17内容和$8内容无符号相减放入$17 |
| 01211fffa | beq $16, $17, start | #将$16内容和$17内容比较，若相等则跳转到start |
| 34080004 | ori $8, $0,4 | #将4和$0内容做或运算放入$8 |
| 03c017fff | addiu $24,$0,0x7fffffff | #将$0内容和0x7fffffff带符号相加放入$24 |
| 03421ffff |
| 0001c021 |
| 27090003 | addiu $9,$24,3 | #将$24内容和3带符号相加放入$9 |
| 270a0005 | addiu $10,$24,5 | #将$24内容和5带符号相加放入$10 |
| 23160006 | addi $22,$24,6 | #将$24内容和6带符号相加放入$22 |
| ad09fffc | start2: sw $9, -4($8) | #start2：$9内容放到以$8内容为基地址偏移-4个字节地址指向的存储器单元 |
| ad010000 | sw $1, 0($8) | #$1内容放到以$8内容为基地址偏移0个字节地址指向的存储器单元 |
| 810e0003 | lb $14, 3($8) | #将以$8内容为基地址偏移3个字指向的存储器单元存放数据放入$14 |
| a10c0007 | sb $12,7($8) | #$12内容放到以$8内容为基地址偏移3个字地址指向的存储器单元 |
| 8d0f0004 | lw $15,4($8) | #将以$8内容为基地址偏移4个字节指向的存储器单元存放数据放入$15 |
| a104fffd | sb $4, -3($8) | #$4内容放到以$8内容为基地址偏移-3个字地址指向的存储器单元 |
| 8112ffff | lb $18, -1($8) | #将以$8内容为基地址偏移-1个字指向的存储器单元存放数据放入$18 |
| 00082021 | addu $4,$0,$8 | #将$0内容和$8内容无符号相加放入$4 |
| 00092821 | addu $5,$0,$9 | #将$0内容和$9内容无符号相加放入$5 |
| 0c000c33 | jal newadd | #将下一条指令的地址放入$31并跳转到newadd |
| 0148c82a | slt $25,$10,$8 | #将$10内容和$8内容比较，$10内容小于$8内容则$25存1，否则存0 |
| 13200018 | beq $25, $0,end2 | #将$25内容和$0内容比较，若相等则跳转到end2 |
| 0184a02a | slt $20,$12,$4 | #将$12内容和$4内容比较，$12内容小于$4内容则$20存1，否则存0 |
| 12800001 | beq $20, $0, end1 | #将$20内容和$0内容比较，若相等则跳转到end1 |
| 3c0cffff | lui $12, 65535 | #将65535放到$12中的高16位，低16位补0 |
| 34000001 | end1: ori $0, $0,1 | #end1：将1和$0内容做或运算放入$0 |
| 3c13efef | lui $19, 0xefef | #将0xefef放到$19中的高16位，低16位补0 |
| 3c01abab | addiu $3,$0,0xababcdcd | #将$0内容和0xababcdcd带符号相加放入$3 |
| 3421cdcd |
| 00011821 |
| 24640002 | start3: addiu $4, $3, 2 | #start3：将$3内容和2带符号相加放入$4 |
| 20770005 | addi $23, $3, 5 | #将$3内容和5带符号相加放入$23 |
| 0c000c33 | jal newadd | #将下一条指令的地址放入$31并跳转到newadd |
| 00024021 | addu $8, $0, $2 | #将$0内容和$2内容无符号相加放入$8 |
| 00082021 | addu $4, $0, $8 | #将$0内容和$8内容无符号相加放入$4 |
| 00092821 | addu $5, $0, $9 | #将$0内容和$9内容无符号相加放入$5 |
| 0c000c33 | jal newadd | #将下一条指令的地址放入$31并跳转到newadd |
| 00024821 | addu $9, $0, $2 | #将$0内容和$2内容无符号相加放入$9 |
| 01004821 | addu $9, $8, $0 | #将$8内容和$0内容无符号相加放入$9 |
| 3c0a0069 | lui $10, 0x69 | #将0x69放到$9中的高10位，低16位补0 |
| 11090001 | beq $8, $9, start4 | #将$8内容和$9内容比较，若相等则跳转到start4 |
| 1000fff4 | beq $0, $0, start3 | #将$0内容和$0内容比较，若相等则跳转到start3 |
| 08000c37 | start4: j end | #start4：无条件跳转到end |
| 00851021 | newadd: addu $2, $4, $5 | #newadd：将$4内容和$5内容无符号相加放入$2 |
| 21801234 | addi $0,$12,0x1234 | #将$12内容和0x1234带符号相加放入$0 |
| 03e00008 | jr $31 | #返回到$31储存的地址所指向指令 |
| 201a5678 | end2: addi $26,$0,0x5678 | #end2：将$0内容和0x5678带符号相加放入$26 |
|  | end: | #end： |

**2.mars运行结果**



**3.实际测试波形图及测试结果**





# 五、心得体会

在CPU执行的过程中，每条指令只执行它们所拥有的状态，而跳过它们没有的状态，不但可以提高运行速度，而且可以提高运行效率，运用这种状态，使得执行每一条指令的过程更加清晰。在P1的基础上更改P2，使我更加清楚的认识了两种不同处理方式的差异性，以及一些方面的共通性。在本次课设中我更加了解了MIPS多周期控制，对于课上没有了解的内容进行了进一步的补充，对于每一条指令的工作状态都有了进一步的了解。通过对状态机部分的编程，使我对几个状态，几种阶段有了更加清晰的认识，并绘制了流程图，每一条指令执行的状态过程；而每一个状态对应着不同译码，从而达到控制CPU器件的工作过程；每一条指令对应的译码控制不同的使能信号。由于做P2的时间较短，因此在编写过程中没有考虑其结构化，本次课设将lb\sb单独作为两个模块去书写，有利于增加代码的可重用性。总之P2的课设收获颇丰，更加有助于我对课上掌握不充分部分的进一步理解。