**计算机组成原理**

**期末大作业报告**

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

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

**指导教师\_\_\_\_\_\_ 魏坚华\_ \_\_\_\_\_\_\_\_\_\_**

**提交日期\_\_\_ 2022.6.7\_\_\_ \_\_\_\_\_\_\_\_\_**

**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** |  |
| **报告与大作业功能一致性** | **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:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

目录

**一、设计说明1**

**二、数据通路3**

**三、模块定义3**

3.1 PC模块4

3.2 NPC模块4

3.3 im\_1k模块5

3.4 GPR模块5

3.5 ALU模块6

3.6 EXT模块7

3.7 dm\_1k模块7

3.8 MUX模块8

3.9 ctr模块9

3.10 mips模块10

**四、测试要求12**

4.1 测试程序12

4.2 预期测试结果14

4.3 实际测试结果15

**五、心得体会15**

**Project1 VerilogHDL完成单周期处理器开发**

# 一、设计说明

1.处理器应实现MIPS-Lite1指令集。

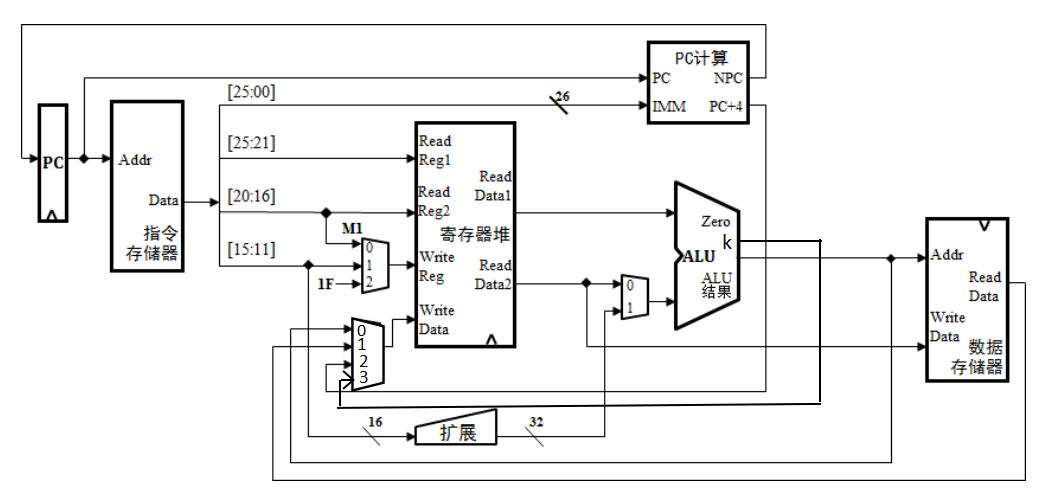
MIPS-Lite1＝{MIPS-Lite，addi，addiu， slt，jal，jr}。

MIPS-Lite指令集：addu，subu，ori，lw，sw，beq，lui，j。

addi应支持溢出，溢出标志写入寄存器$30中第0位。

2.处理器为单周期设计。

# 二、数据通路

****

**三、模块定义**

1.PC模块定义

（1）基本描述

PC主要功能是完成输出当前指令地址并保存下一条指令地址。复位后，PC指向0x0000\_3000，此处为第一条指令的地址。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| npc[31:0] | I | 下条指令的地址 |
| clk | I | 时钟信号 |
| reset | I | 复位信号。  1：复位  0：无效 |
| PC[31:0] | O | 32位指令存储器地址 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x0000\_3000。 |
| 2 | 保存npc并输出pc | 在每个clock的上升沿保存npc，并输出pc。 |

2.NPC模块定义

（1）基本描述

通过不同指令的执行情况计算下一条指令的地址。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| pc[31:0] | I | 32位指令存储器地址 |
| JRaddr[31:0] | I | 执行jr指令,写给pc |
| sel | I | 选择信号，根据指令不同选择pc改写方式 |
| Zero | I | 执行beq指令，pc地址改写信号 |
| Ins[31:0] | I | 32位MIPS指令 |
| npc[31:0] | O | 下条指令的地址 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条指令地址 | ①如果当前指令不是beq、j、jal、jr指令，则PC🡨PC+4  ②如果当前指令是beq指令，且zero为0，则PC🡨PC+4  如果当前指令是beq指令，且zero为1，则PC🡨PC+4+sign\_ext(Instr[15:0])||00  ③如果当前指令是j指令或jal指令，则PC🡨PC+4[31:28] || insout[25:0]||00。  ④如果当前指令是jr指令，则pc🡨JRaddr。 |
| 2 | 分析当前信号 | 根据输入的sel分析出当前信号是j、jal、jr还是beq，从而生成不同的指令地址。 |

3.im\_1k模块定义

（1）基本描述

保存指令并根据输入的地址，从指令寄存器中取出相应的指令。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[9:0] | I | 指令存储器地址 |
| dout[31:0] | O | 32位MIPS指令 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 保存指令 | 将生成的16进制保存到指令寄存器里。 |

4.GPR模块定义

（1）基本描述

寄存器组内含32个32位寄存器、写信号及其他相关逻辑。GPR按照rs和rt提供的编号读取内容，按照rt或者rd提供的编号写入内容。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| we | I | 读写控制信号  1：写操作  0：读操作 |
| rw[4:0] | I | 写寄存器地址 |
| ra[4:0] | I | 数据1寄存器地址 |
| rb[4:0] | I | 数据2寄存器地址 |
| clk | I | 时钟信号 |
| rst | I | 复位信号。  1：复位  0：无效 |
| busw[31:0] | I | 写入数据的输入 |
| busa[31:0] | O | 32位数据1 |
| busb[31:0] | O | 32位数据2 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，寄存器被设置为0x00000000。 |
| 2 | 读写操作 | 读取：ra作为编号的寄存器内容输出到busA，rb作为编号的寄存器内容输出到busB。  写入：当信号we=1时，busW上的数据被写入ra作为编号的寄存器内。 |
| 3 | 时钟信号 | clk仅仅写操作时有效，表明写入时刻  当读操作时，clk失效，转变为组合逻辑操作，只要ra或rb有效，经过一定的寄存器存取时间，数据就会出现在busA和busB上。 |

5.ALU模块定义

（1）基本描述

ALU内含无符号加、减及或运算，当输入两路数据后，输出相应的运算结果，如果是lw或sw指令，则计算访存地址。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 32位输入数据1 |
| B[31:0] | I | 32位输入数据2 |
| ALUctr[1:0] | I | 控制信号  00：加运算  01：减运算  10：或运算  11：输出B(lui指令) |
| addi | I | 控制信号  1：指令为addi  0：指令不是addi |
| ALUout[31:0] | O | 32位数据输出 |
| Zero | O | ALU计算结果为0标志。(A=B)  1：计算结果为0  0：计算结果非0 |
| OF | O | addi溢出标志  1：溢出  0：未溢出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加 | A+B |
| 2 | 减 | A-B |
| 3 | 或 | A | B |
| 4 | 判断溢出 | addi指令时，加数与被加数最高位相同，且与结果最高位相反，则发生溢出，溢出标志OF为1。 |

6.EXT模块定义

（1）基本描述

将16位立即数实现零扩展、符号扩展、低位补0而扩展为32位。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imma[31:0] | I | 32位MIPS指令 |
| ExtOp[1:0] | I | 控制信号  0：零扩展  1:符号扩展 |
| Luisel | I | 控制信号  1：指令为lui  0：指令不是lui |
| immb[31:0] | O | 扩展后的32位数据输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零扩展 | 高16位补0 |
| 2 | 符号扩展 | 高16位符号扩展 |
| 4 | Lui扩展 | 16立即数置于高16位，低16位补0 |

7.dm\_1k模块定义

（1）基本描述

实现相应的数据存储器的写入及输出功能。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr[9:0] | I | 数据存储器的地址 |
| din[31:0] | I | 写入数据的输入 |
| we | I | 读写控制信号  1：写操作 |
| clk | I | 时钟信号 |
| dout[31:0] | O | 32位数据输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，寄存器被设置为0x00000000。 |
| 2 | 读操作 | 根据输入的寄存器地址读出数据 |
| 3 | 写操作 | 根据输入的地址，将输入的数据写入相应的寄存器 |

8.Mux模块定义

（1）基本描述

①Memsel:回写数据的选择

②regctr:回写地址的选择

③ALUsel:写入ALU运算操作的数据2的选择

④sltsel:ALU运算结果ALUout的选择

（2）模块接口

①Memsel:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| slt\_result  [31:0] | I | ALU运算经过slt选择后32位结果 |
| dout[31:0] | I | 数据存储器输出数据 |
| jalpc[31:0] | I | pc+4 |
| MemtoReg[1:0] | I | DM读控制信号  00：slt\_result回写  01：dout回写  10：jalpc回写 |
| busw | O | 回写数据 |

②regctr:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| insout[31:0] | I | 32位MIPS指令 |
| jalreg | I | 31号寄存器 |
| ofreg | I | 30号寄存器 |
| RegDst[1:0] | I | 写地址控制信号  00：写到rt对应的寄存器  01：写到rd对应的寄存器  10：写到31号寄存器  11:写到30号寄存器 |
| rw[4:0] | O | 写寄存器地址 |

③ALUsel:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ALUSrc | I | alu32位输入数据选择信号  0：busb  1:Extout |
| busb[31:0] | I | 寄存器32位数据2 |
| Extout[31:0] | I | 扩展后的32位数据 |
| aluin[31:0] | O | alu32位输入数据2 |

④sltsel:

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| slt | I | slt指令信号  0：指令不为slt，输出aluout  1:指令为slt，输出sltout |
| aluout[31:0] | I | alu运算结果 |
| {31’b0,aluout  [31]}[31:0] | I | aluout结果符号位 |
| slt\_result | O | alu32位输入数据2 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 回写数据的选择 | 00：运算器结果回写  01：数据存储器输出数据回写  10：pc+4回写 |
| 2 | 回写地址的选择 | 00：写到rt对应的寄存器里  01：写到rd对应的寄存器里  10：写到31号寄存器里  11：写到30号寄存器里 |
| 3 | 写入ALU运算操作的数据2的选择 | 0：寄存器输出的32位数据2  1：扩展后的32位数据 |
| 4 | ALU运算结果ALUout的选择 | 0：指令不为slt，输出aluout  1:指令为slt，输出sltout |

9.ctr模块定义

（1）基本描述

根据32位指令分析出的opcode和funct对应于相应的指令，分析出每个指令执行的数据通路的选择信号。

（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 |
| opcode[5:0] | I | 32位MIPS指令中的opcode |
| funct[5:0] | I | 32位MIPS指令中的funct |
| OF | I | addi运算的溢出标志 |
| RegDst | O | 写地址控制信号  00：写到rt对应的寄存器里  01：写到rd对应的寄存器里  10：写到31号寄存器里 |
| RegWrite | O | 读写控制信号  1：写操作  0：读操作 |
| ALUSrc | O | alu32位输入数据选择信号  0：寄存器输出的32位数据2  1：扩展后的32位数据 |
| ALUctr | O | 控制信号  00：加运算  01：减运算  10：或运算  11：输出B |
| MemWrite | O | 读写控制信号  1：写操作 |
| MemtoReg | O | DM读控制信号  00：运算器结果回写  01：数据存储器输出数据回写  10：pc+4回写 |
| N\_pcsel | O | pc地址改写信号 |
| ExtOp | O | 控制信号  00：零扩展  01：符号扩展  10:低16位补0 |
| slt | O | slt指令信号 |
| addi | O | addi指令信号 |
| luisel | O | lui指令信号 |

（3）指令功能定义

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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] |

（4）单周期控制器真值表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | addu | subu | ori | lw | sw | beq | lui | j | addi | addiu | slt | jal | jr |
| RegDst | 01 | 01 | 00 | 00 | x | x | 00 | x | 00 | 00 | 01 | 10 | x |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 | x | 1 | 1 | 0 | x | x |
| MemtoReg | 00 | 00 | 00 | 01 | x | x | 00 | x | 00 | 00 | 11 | 10 | x |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| NPC\_sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ExtOp | x | x | 00 | 01 | 01 | x | 10 | x | 01 | 01 | x | x | x |
| ALUctr<1:0> | 00 | 01 | 10 | 00 | 00 | 01 | 11 | xx | 00 | 00 | xx | xx | xx |

10.mips模块定义

（1）基本描述

起各模块的连接作用。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| rst | I | 置位信号 |

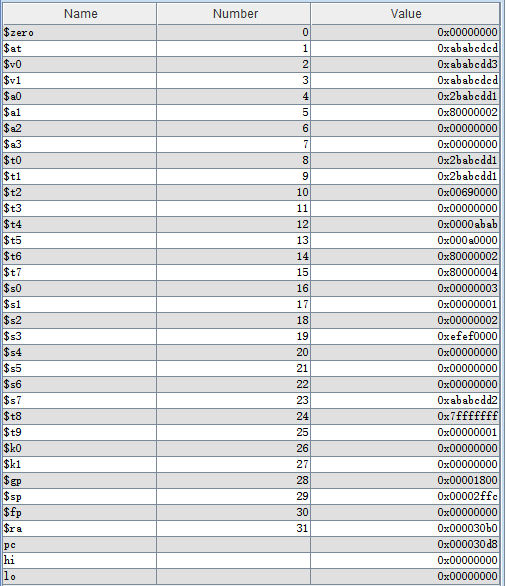
**四、测试要求**

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 |
| 1211fffa | 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 |
| 3c017fff | addiu $24,$0,0x7fffffff | #将$0内容和0x7fffffff带符号相加放入$24 |
| 3421ffff |
| 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 |
| ad090000 | start2: sw $9, 0($8) | #start2：$9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 |
| 8d0e0000 | lw $14, 0($8) | #将以$8内容为基地址偏移0个字节指向的存储器单元存放数据放入$14 |
| ad0a0004 | sw $10,4($8) | #$10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 |
| 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 |
| ad04fffc | sw $4, -4($8) | #$4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 |
| 8d12fffc | lw $18, -4($8) | #将以$8内容为基地址偏移0-4个字节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 |
| 0c000c32 | 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 |
| 0c000c32 | 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 |
| 0c000c32 | 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 |
| 08000c36 | 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： |

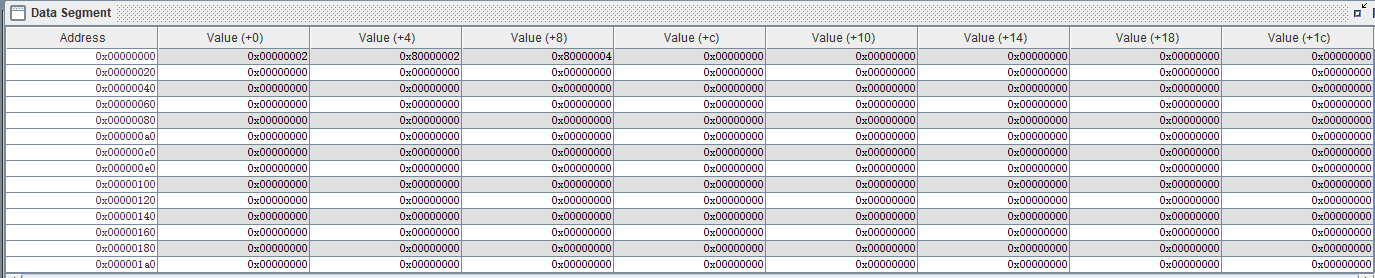
2.预期测试结果

①寄存器值：



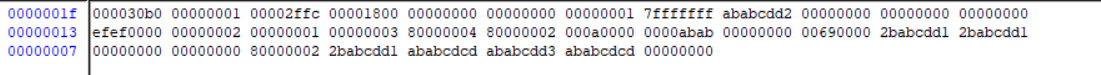
mars运行寄存器值结果

②数据存储器值：



mars运行数据存储器值结果

3.实际测试结果



modelsim运行寄存器值结果



modelsim运行数据存储器值结果

**五、心得体会**

这次课设是在使用Logisim之后第一次编程，第一次使用Modelsim，首先对于被verilog的使用更加的熟练，第一次进行调试，虽然经历了十分痛苦的过程，但掌握了更多的功能，代码的编写更加的熟练，在调试的过程中，对于几个模块之间的相互作用，模块的调用有了更加深刻的理解和体会。这次课设使我更加熟练的掌握mips指令，同时弥补了之前的空缺，对于一些指令更加熟练，更加了解他们的操作方法。对于每一条的单周期循环通路，也能熟练掌握，可以随机应变的添加各种功能的新指令，这对我学习计算机组成原理这门课也是十分有帮助的。

相比较第一个大作业，它主要是将logisim中通过具体的连线搭建的的数据通路，变成了用verilog语言编程的形式搭建出来，同时增加了addi的溢出检测。这就需要我在设计的时候不能简单地把模块之间相互连接，而是要搞清楚每个模块之间的内在联系，包括溢出的判断和设置也是之前用logisim完成单周期处理器开发时所没有细致考虑过的点。在解决这些问题的过程中，我逐渐理解了硬件描述语言的编程思路，结合上个学期数字逻辑实验的相关内容，我更加深了对计算机中硬件调控的理解。我认识到只有绝对正确的程序才可以得到想要的结果，而但凡某个模块出现一点差错，即使其他模块都非常正确，也很有可能导致完全错误的结果。这就需要我们在编程的时候熟悉自己要实现的具体内容，了解单周期处理器的每个模块的具体功能，准确无误地实现他们并合理地连接起来，只有这样才能确保万无一失，无论是寄存器结果还是仿真结果都可以达到要求。这段时期的学习让我受益匪浅，更加深了自己对之后两个课设的期待，希望自己可以取得好的结果。