计算机组成原理实验报告

1. 数据通路设计

1、 IFU（取指令单元）

IFU由PC与IM两部分构成：

1、PC由一个32位寄存器实现，

2、IM由32个32位寄存器实现

**PC端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | JPC | Input | [31:0] | 输入j型跳转的地址 |
| 2 | Imme | Input | [31:0] | 输入imme |
| 3 | Pcsrc | Input | 1 | 输入是否跳转信号 |
| 4 | Jump | Input | 1 | 输入是否j跳转信号 |
| 5 | Clk | Input | 1 | 时钟输入信号 |
| 6 | Reset | Input | 1 | 复位信号 |
| 7 | PC | Output | [31:0] | 输出当前PC值 |

**IM端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | PC | Input | [31:0] | 输入需要取指令的地址 |
| 2 | Addr | Output | [25:0] | 输出Addr |
| 3 | Imme | Output | [15:0] | 输出imme |
| 4 | Op | Output | [5:0] | 输出op |
| 5 | Rs | Output | [4:0] | 输出rs |
| 6 | Rt | Output | [4:0] | 输出rt |
| 7 | Rd | Output | [4:0] | 输出rd |
| 8 | Shamt | Output | [4:0] | 输出shamt |
| 9 | Func | Output | [5:0] | 输出func |
| 10 | Instr | Output | [31:0] | 根据PC提供的指令地址，输出指令 |

2、GRF(寄存器堆)

GRF由由32个32位寄存器实现

**GRF端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | A1 | Input | [4:0] | 读端口，输入地址，用于指定32个寄存器中的一个作为读入寄存器 |
| 2 | A2 | Input | [4:0] | 读端口，输入地址，用于指定32个寄存器中的一个作为读入寄存器 |
| 3 | A3 | Input | [4:0] | 写端口，输入地址，用于指定32个寄存器中的一个作为写入寄存器 |
| 4 | WD3 | Input | [31:0] | 写端口，输入数据，将数据写入到寄存器 |
| 5 | PC | Input | [31:0] | 读入指令 |
| 5 | WE3 | Input | 1 | 写入使能信号，若为1则写入 |
| 6 | Reset | Input | 1 | 重置使能，若为1，清空所有寄存器 |
| 7 | CLK | Input | 1 | 时钟输入信号 |
| 8 | RD1 | Output | [31:0] | 读入寄存器的数据输出 |
| 9 | RD2 | Output | [31:0] | 读入寄存器的数据输出 |

3、ALU（算术逻辑单元）

**ALU端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | In1 | Input | [31:0] | 写端口，将操作数传入到ALU |
| 2 | In2 | Input | [31:0] | 写端口，将操作数传入到ALU |
| 3 | ALUOp | Input | [1:0] | ALU控制信号，通过信号判断执行指令 |
| 4 | Funct | Input | [5:0] | 写入funct指令段 |
| 5 | Op | Input | [6:0] | 写入Op指令段 |
| 6 | ALUResult | Output | [31:0] | 输出ALU操作结果 |
| 7 | Zero | Output | 1 | PC跳转使能信号 |

1. DM（数据存储器）

DM由2048个32位寄存器实现

**DM端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | Addr | Input | [31:0] | 写端口，输入地址，用于指定DM的数据地址 |
| 2 | Din | Input | [31:0] | 写端口，写入数据到DM |
| 3 | Memwrite | Input | 1 | DM写入使能，若为1，则写入 |
| 4 | Clk | Input | 1 | 时钟输入信号 |
| 5 | Reset | Input | 1 | 重置使能，若为1，清空DM |
| 6 | Pc | Input | [31:0] | 输入pc |
| 7 | Dout | Output | [31:0] | DM读出数据 |

1. EXT

**EXT端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | immediate | Input | [15:0] | 输入要扩展的数据 |
| 2 | ExtOp | Input | 1 | 控制信号输入，选择无符号扩展或符号扩展 |
| 3 | Ext\_out | Output | [31:0] | 输出已扩展的数据 |

二、**控制器设计**

通过输入的op和func来判断输出的使能信号

**控制信号端口功能表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 编号 | 名称 | Intput/Output | 端口位数 | 功能实现 |
| 1 | Op | Input | [5:0] | 写入Op指令段 |
| 2 | Func | Input | [5:0] | 写入func指令段 |
| 3 | Zero | Input | 1 | 输入zero使能信号 |
| 4 | MemtoReg | Output | 1 | 内存写入寄存器使能信号 |
| 5 | RegDst | Output | 1 | 选择目的寄存器由rt字段或rd字段控制 |
| 6 | ALUsrc | Output | 1 | 选择SrcB来自RD2或signimm |
| 7 | Regwrite | Output | 1 | 输出寄存器写入使能信号 |
| 8 | Memwrite | Output | 1 | 输出内存写入使能信号 |
| 9 | Pcsrc | Output | 1 | 输出是否跳转信号 |
| 10 | ALUOp | Output | 1 | 输出ALU选择信号 |
| 11 | ExtOp | Output | 1 | 选择扩展immediate |
| 12 | Jump | Output | 1 | 输出J跳转信号 |
| 13 | Jal | Output | 1 | 输出Jal使能信号 |
| 14 | Jr | Output | 1 | 输出Jr使能信号 |

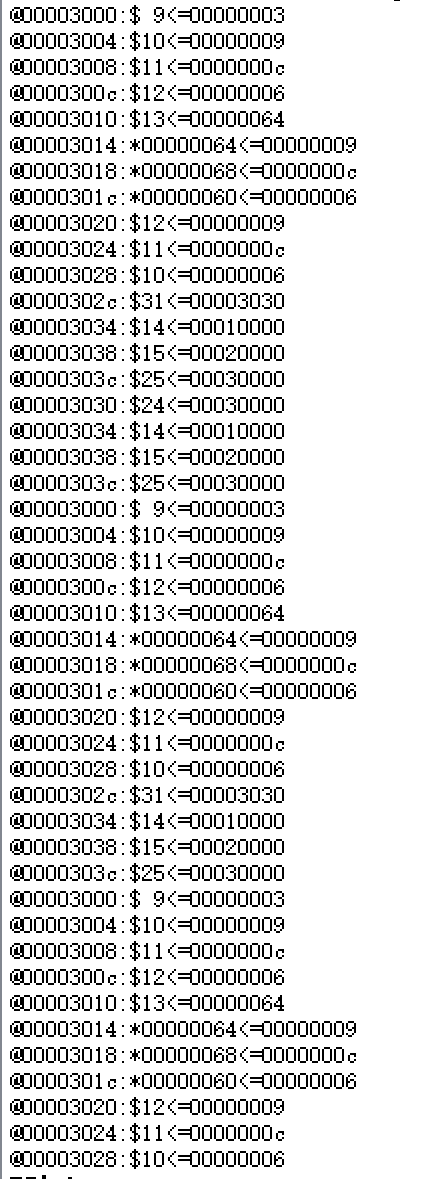
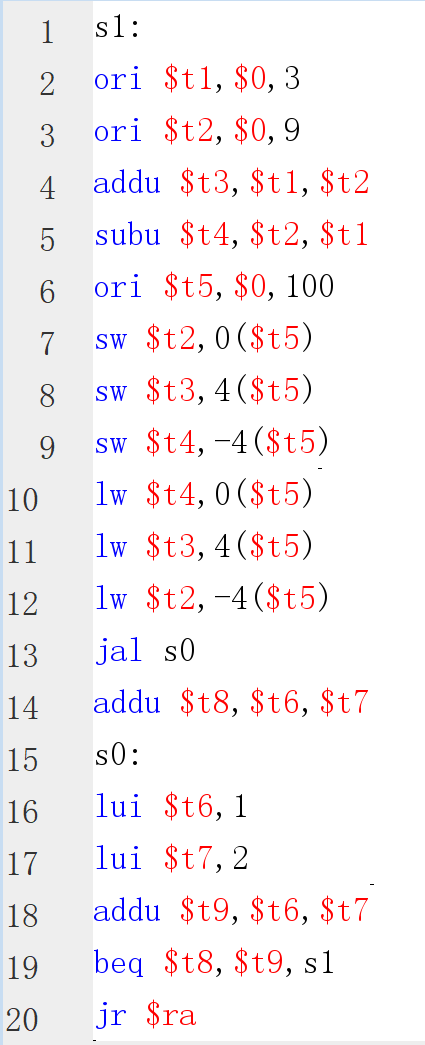
**控制信号真值表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | 100001 | 100011 | N/A | | | | | 000000 |
| op | 000000 | 000000 | 001101 | 001111 | 100011 | 101011 | 000100 | 000000 |
| 名称 | addu | subu | ori | lui | lw | sw | beq | nop |
| RegDst | 1 | 1 | 0 | 0 | 0 | X | X | X |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 1 | 0 | X |
| MemtoReg | 0 | 0 | 0 | 0 | 1 | X | X | X |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 0 | 0 | X |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 1 | 0 | X |
| PCSrc | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X |
| ExtOp | X | X | 0 | X | 1 | 1 | X | X |
| ALUop  <2.0> | 10 | 10 | 00 | 00 | 00 | 00 | 01 | XX |
| Jump | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Jal | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Jr | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**控制信号真值表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | N/A | | 001000 |  |  |  |  |  |
| op | 000010 | 000011 | 000000 |  |  |  |  |  |
| 名称 | J | Jal | jr |  |  |  |  |  |
| RegDst | X | X | X |  |  |  |  |  |
| ALUSrc | X | X | X |  |  |  |  |  |
| MemtoReg | X | X | X |  |  |  |  |  |
| RegWrite | X | 1 | X |  |  |  |  |  |
| MemWrite | X | X | X |  |  |  |  |  |
| PCSrc | X | X | X |  |  |  |  |  |
| ExtOp | X | X | X |  |  |  |  |  |
| ALUop  <2.0> | XX | XX | X |  |  |  |  |  |
| Jump | 1 | 1 | 1 |  |  |  |  |  |
| Jal | 0 | 1 | 0 |  |  |  |  |  |
| Jr | 0 | 0 | 1 |  |  |  |  |  |

三、测试程序



s1:

ori $t1,$0,3

ori $t2,$0,9

addu $t3,$t1,$t2

subu $t4,$t2,$t1

ori $t5,$0,100

sw $t2,0($t5)

sw $t3,4($t5)

sw $t4,-4($t5)

lw $t4,0($t5)

lw $t3,4($t5)

lw $t2,-4($t5)

jal s0

addu $t8,$t6,$t7

s0:

lui $t6,1

lui $t7,2

addu $t9,$t6,$t7

beq $t8,$t9,s1

jr $ra

四、思考题

T2：

1、根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

答：

因为在mars当中为按字节寻址，verilog中为按字寻址。addr信号通过ALU计算得来。

2、在相应的部件中，**reset的优先级**比其他控制信号（不包括clk信号）都要**高**，且相应的设计都是**同步复位**。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

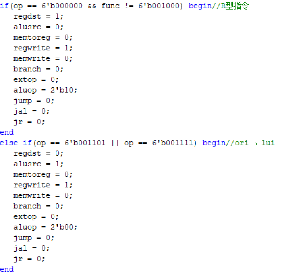
答：

会对GRF、DM进行清零操作，对PC进行复位操作，在未清零时，电路板为未知状态。

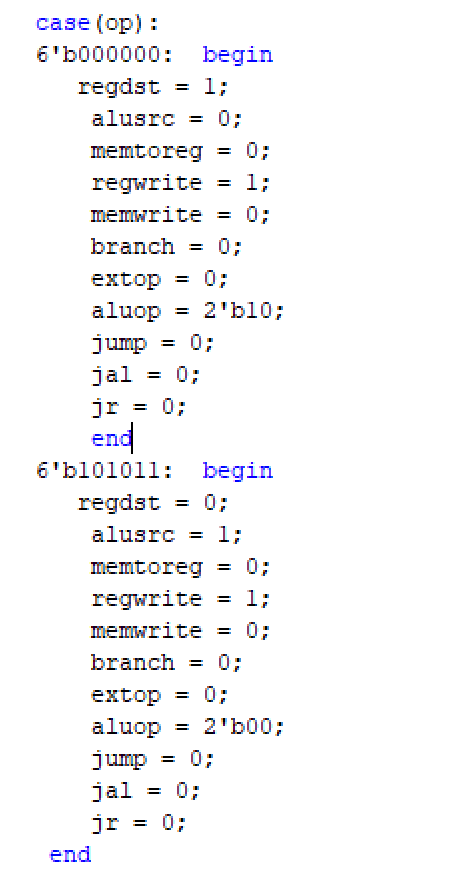
T3:

1. 列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

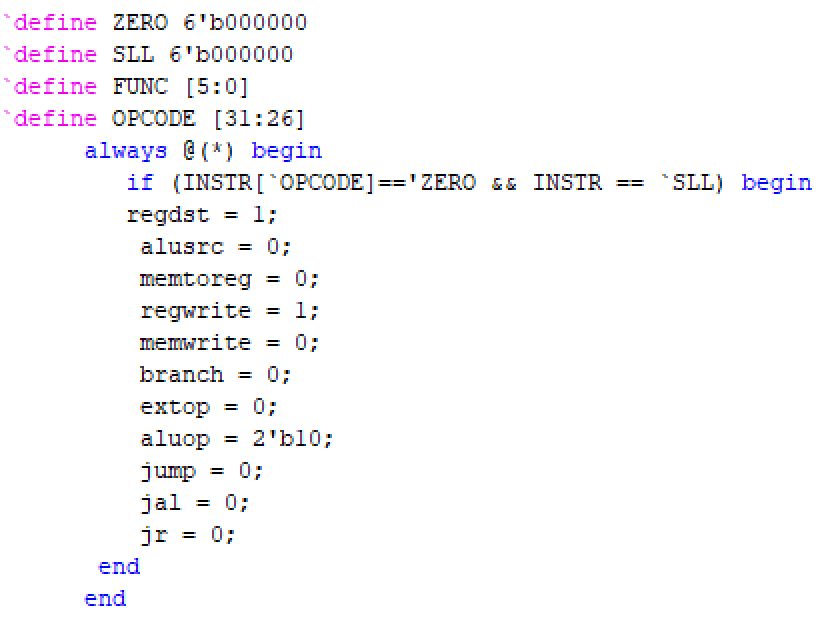
答：

1、if-else：

2、case：



3、`define



2.根据你所列举的编码方式，说明他们的优缺点。

答：

If else 思路比较简单，做法比较麻烦

Case 思路较为简单，也比较好做

` define 可以是代码清晰简洁

T5:

1、C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分 。

答：因为addi和addiu与add和addu只有在有溢出的情况下，对cpu的操作才不一样，在无溢出的情况下，操作是一样的。

2、根据自己的设计说明单周期处理器的优缺点。

答：单周期cpu易于实现，但跑得慢。

3、简要说明jal、jr和堆栈的关系。

答：在调用jal函数的时候，要进行的操作是保存当前环境，即将当前环境入栈，而当调用jr函数的时候，要进行的操作是出栈，将当时存的环境弹出栈。