Project3VerilogHDL开发多周期MIPS-C2

# 设计说明

1. 处理器MIPS-C2指令集。
   1. MIPS-C2＝{LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、SLT、SLTU、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLTI、SLTIU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR}
   2. 所有运算类指令均可以不支持溢出。
2. 处理器为多周期设计。

# 设计要求

1. PC复位后初值为0x0000\_3000，目的是与MARS的Memory Configuration相配合。具体配置方法请参考上个project。

# 模块定义

1. 下列模块必须严格满足如下的接口定义：
   1. 你必须在VerilogHDL设计中建模这3个模块。
   2. 不允许修改模块名称、端口各信号的名称/类型/位宽。

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| mips.v | module mips(,clk, rst) ;  input clk ; // clock  input rst ;// reset |
| im.v | module im\_4k( addr, dout ) ;  input [11:2] addr ; // address bus  output [31:0] dout ; // 32-bit memory output |
| dm.v | module dm\_4k( addr, be, din, DMWr, clk, dout ) ;  input [11:2] addr ; // address bus  input [3:0] be ; // byte enables  input [31:0] din ; // 32-bit input data  input DMWr ; // memory write enable  input clk ; // clock  output [31:0] dout ; // 32-bit memory output |

# 测设要求

1. 本project提供Test\_42\_Instr.asm测试程序。可使用MARS运行该测试程序，观察结果进行验证。
2. 也可构造至少包括40条以上指令的测试程序。
   1. MIPS-C2定义的每条指令至少出现1次以上。
   2. 必须有函数，并至少1次函数调用。
3. 函数相关指令(jal和jr)是较为复杂的指令，其正确性不仅涉及到自身的正确性，还与堆栈调整等操作相关。因此为了更充分的测试，你必须在测试程序中组织一个循环，并在循环中多次函数调用，以确保正确实现了这2条指令。

# 开发与调试技巧

1. be[3:0]是字节使能，分别与din[31:24]、din[23:16]、din[15:8]及din[7:0]对应。当we有效时，对于addr寻址的那个word来说，be[3]为1则din[31:24]被写入byte3，类似的be[2]对应din[23:16]和byte2，依此类推。
   1. be[3:0]主要用于支持sb、sh、sw这3条指令。当处理器执行sb、sh、sw指令时，通过对ALUOut[1:0](ALUOut保存了ALU计算的32位地址)的解读后产生相应的be[3:0]，处理器就可以“通知”DM该写入哪些字节。
   2. sw指令：GPR[rt]写入对应的字。

|  |  |  |
| --- | --- | --- |
| ALUOut[1:0] | BE[3:0] | 用途 |
| XX | 1111 | din[31:24]写入byte3  din[23:16]写入byte2  din[15:8]写入byte1  din[7:0]写入byte0 |

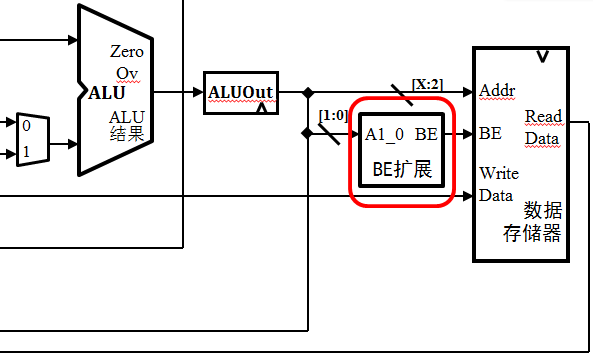
* 1. sh指令：GPR[rt]15:0写入对应的半字。

|  |  |  |
| --- | --- | --- |
| ALUOut[1:0] | BE[3:0] | 用途 |
| 0X | 0011 | din[15:8]写入byte1  din[7:0]写入byte0 |
| 1X | 1100 | din[15:8]写入byte3  din[7:0]写入byte2 |

* 1. sb指令：GPR[rt]7:0写入对应的字节。

|  |  |  |
| --- | --- | --- |
| ALUOut[1:0] | BE[3:0] | 用途 |
| 00 | 0001 | din[7:0]写入byte0 |
| 01 | 0010 | din[7:0]写入byte1 |
| 10 | 0100 | din[7:0]写入byte2 |
| 11 | 1000 | din[7:0]写入byte3 |

* 1. Figure1给出了增加BE扩展的数据通路局部参考设计。显然，BE扩展功能部件还需要有来自控制器的控制信号。注意：由于DM容量有限，因此ALUOut计算出来的32位地址没有必要也不可能都用上，故接入DM的地址总线位数为[X:2](假设DM容量为4KB，那么X就是11；假设DM容量为1MB，那么X就是19；依次类推)。



FigureBE扩展

1. 对于lb、lbu、lh、lhu来说，P1.docx中的数据通路无法支持这些指令。为此你必须增加一个数据扩展模块。这个模块将从数据寄存器读出的数据再进行一次位扩展。
   1. 以lb为例，数据扩展模块输入数据寄存器的32位数据，根据ALUOut[1:0]从中取出特定的字节并以该字节的最高位为符号位做符号扩展。
2. 用$display和$monitor来监控重要变量会提高你的调试效率。如果之前的project都是你自己独立完成的，那么我认为你已经具有很好的工作基础了。换句话说，你已经基本上能驾驭设计了。这时除了看波形外，你需要更加高效的调试方法了。进入这个project后，很多时候我们可以通过观察寄存器来判断程序的正确性了。下面我们通过举一个非常实用的例子来展示$monitor的调试价值。
   1. 现在，我们往往需要观察寄存器的变化来判断处理器设计是否正确。那么请观察下面这段代码。

if ( RegWrite\_I )

begin

rf[j] <= WData\_I ; // 写入寄存器

`ifdef DEBUG

$display("R[00-07]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

$display("R[08-15]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

$display("R[16-23]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

$display("R[24-31]=%8X, %8X, %8X, %8X, %8X, %8X, %8X, %8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

`endif

end

* 1. 这段代码是寄存器文件的片段。我们在写寄存器之后用ifdef引导了4个$display。每当有寄存器被写入后，32个寄存器就都被显示在Modelsim的调试窗口中。显然，通过这种方式，我们可以很容易的发现哪个寄存器被修改了。
  2. 如果再利用$monitor把PC和IR也都监控起来，那么整个CPU的运行状态就非常清晰了。参考代码如下：

mips U\_MIPS( clk, rst ) ;

initial

begin

$monitor("PC = %8X, IR = %8X", U\_MIPS.datapath.pc.pc, U\_MIPS.datapath.ir.ir );

clk = 0 ;

rst = 0 ;

其他语句

end

# 其他要求

1. 打包文件：VerilogHDL工程文件、code.txt、code.txt所对应的汇编程序
2. 时间要求：截止12月23日晚24点
3. 压缩文件命名：“#-学号-姓名”
4. 邮件主题：“#-学号-姓名”
5. 邮箱：zeyicang@foxmail.com