Project2VerilogHDL开发多周期MIPS-C2

V1.0@2014.07.3

高小鹏

# 设计说明

1. 处理器应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。
2. 建议你用多周期CPU工程化综合方法来构造数据通路和状态机。
   1. 一旦当你运用这个方法后，你会发现设计将被分为2个环节：真正意义上的设计和实现。
   2. 设计：仅仅在表格中就可以完成，而无需在HDL这个层次进行。这一方面避免了HDL的大量细节对设计的干扰，另一方面使得你的设计可以被追朔，从而更高效的开发和准确的发现错误。
   3. 实现：当设计都完成后，可以一次性的完成HDL编码。这个编码仅仅是对表格的翻译，对你的要求仅仅是掌握基本的HDL和认真细心。事实上，如果你是一个优秀的设计人员，你会发现完全可以开发一个软件来将你的设计自动的转化为HDL代码。
   4. 表面上看，直接写HDL代码似乎很快。但根据我们的经验，你的大量时间将花费在反复试错与反复修改的迭代上。一旦你建立了这样的工作方法，你会发现今后再修改设计应该首先从表格开始（即设计），并在设计阶段就可以检查设计正确性。这会使你的再开发效率大幅度提升。

# 模块定义【WORD】

1. 仿照下面给出的PC模块定义，给出所有功能部件的模块定义。
   1. 具体要求请参考上个project的相关要求。
2. 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\_40\_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。
4. 详细说明你的测试程序原理及测试结果。【WORD】
   1. 应明确说明测试程序的测试期望，即应该得到怎样的运行结果。
   2. 每条汇编指令都应该有注释。

# 成绩及实验测试要求

1. 实验成绩包括但不限于如下内容：初始设计的正确性、增加新指令后的正确性、实验报告等。
2. 实验测试时，你需要展示你的设计并证明其正确性。
   1. 应简洁的描述你的验证思路，并尽可能予以直观展示。

# 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 |

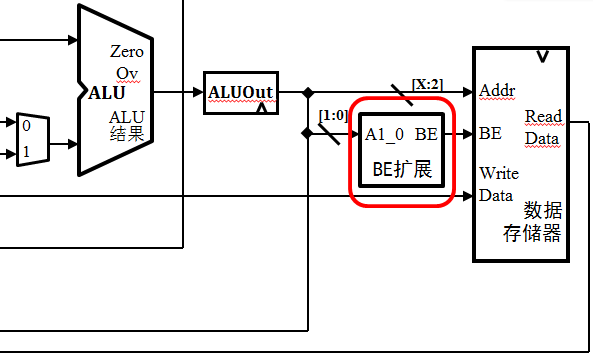
* 1. sh指令：GPR[rt]15:0写入对应的半字。

|  |  |  |
| --- | --- | --- |
| 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. 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. 由于MIPS-C2支持了更多的指令，因此我们已经基本可以用GCC来编译用C写的简单测试程序了。我们提供了基于cygwin(windows下模拟linux)的GCC编译环境，这样你可以更快速的编写测试程序了。Cygwin压缩包及简要使用说明很快提供。假设使用gcc，那么你的测试程序开发流程可以如下：
   1. S1：验证C功能正确性。用windows上的某个C开发环境(CodeBlocks、VisualStudio等)先编写并测试C代码。注意：由于你的mips系统还非常的简陋，还不能运行复杂系统，因此类似printf这样的库函数只能在windows上测试时使用，在gcc编译时需要剔除。
   2. S2：生成MIPS汇编。用cygwin下的gcc编译测试程序。
   3. S3：提取测试用MIPS汇编。由于生成的汇编包含部分无关代码(由于系统的原因)，因此你需要提取出你真正用于测试的MIPS汇编部分。
   4. S4：修正测试用MIPS汇编。打开gcc生成的bin.elf.txt，你能看到C代码与汇编指令的对应关系。1）看看有没有不支持的指令；2）看看有没有需要支持但没有产生的指令。3）调整数据的基地址、堆栈的基地址(通常都只涉及几条指令)。如果出现了任意一种情况，你可能都需要在这段MIPS汇编基础上改写一些内容。
   5. S5：MARS上验证改写的MIPS汇编。与windows的C环境进行对比(建议对比DM数据)。如果验证是一致的，那么恭喜你，你得到了可以用来测试你自己的处理器的mips代码了。
3. 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、项目报告。
2. 时间要求：实验指导教师指定。
3. 本实验要求文档中凡是出现了【WORD】字样，就意味着该条目需要在实验报告中清晰表达。
4. 实验报告请按照《计算机组成原理实验报告撰写规则.doc》要求排版。