Project2 VerilogHDL 完成多周期处理器开发

# 设计说明

1. 处理器应实现MIPS-Lite2指令集。
   1. MIPS-Lite2＝{MIPS-Lite1，lb，sb}。
   2. MIPS-Lite1＝{addu，subu，ori，lw，sw，beq，j，lui，addi，addiu，slt，jal，jr }。
   3. addi应支持溢出，溢出标志写入寄存器$30中第0位。
2. 处理器为多周期设计。

# 设计要求

1. 多周期处理器由 datapath(数据通路)和 controller(控制器)组成。
   1. 数据通路应至少包括如下module：PC(程序计数器)、NPC(NextPC 计算单元)、GPR (通用寄存器组，也称为寄存器文件、寄存器堆)、ALU(算术逻辑单元)、EXT(扩展单元)、IM(指令存储器)、 DM(数据存储器)等。
   2. IM：容量为1KB(8bit×1024)。
   3. DM：容量为1KB(8bit×1024) ，采用小端序方式存取数据。
2. Figure1为供你参考的数据通路架构图。
   1. 我们不确保Figure1是完全正确的；我们也不确保Figure1能够满足MIPS-Lite2。
   2. 鼓励你从数据通路的功能合理划分的角度自行设计更好的数据通路架构。

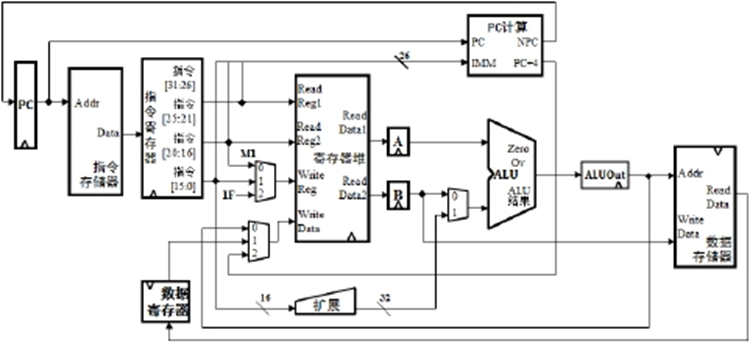


Figure1 数据通路(供参考)

1. 为使得代码更加清晰可读，建议多使用宏定义，并将宏定义组织在合理的头文件中。
2. PC复位后初值为0x0000\_3000，目的是与MARS的Memory Configuration相配合。
   1. 测试程序将通过MARS产生，其配置模式如Figure2所示。



Figure2 MIPS存储配置模式(MARS memory configuration)

1. PPT 中的状态机设计仅供你参考。你可以根据对指令的理解去构造不同的状态机。但仍然建议遵循下述原则：

a) 按指令类别构造状态分支，而不是按每条指令。

b) 状态分支不宜过多。

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、端口各信号以及变量的名称/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 | im\_1k( addr, dout ) ;  input [11:0] addr ; // address bus  output [31:0] dout ; // 32-bit memory output  reg [7:0] im[1023:0] ; |
| dm.v | dm\_1k( addr, din, we, clk, dout ) ;  input [11:0] addr ; // address bus  input [31:0] din ; // 32-bit input data  input we ; // memory write enable  input clk ; // clock  output [31:0] dout ; // 32-bit memory output  reg [7:0] dm[1023:0] ; |

# 测试要求

1. 所有指令都应被测试充分。
2. 构造至少包括40条以上指令的测试程序，并测试通过。
   1. MIPS-Lite2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。

# 其他要求

1. 提交的压缩文件内容：工程中所有.v文件、code.txt、code.asm、项目报告。
2. 时间要求：各班实验指导教师指定。

# 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，你必须已经完成了处理器设计及开发。
3. 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。
4. 实验指导教师会临时增加1～2条指令，你需要在规定时间内完成对原有设计的修改，并通过实验指导教师提供的测试程序。
   1. 考查时，教师将用专用testbench和code.txt检测代码执行情况。

# 开发与调试技巧

1. 对于每条指令，请认真阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》！
   1. 如果测试时，你无法清楚的解释所要求的指令，测试成绩将减一档！
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。

* 1. 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

$monitor("PC = %8X, IR = %8X",

U\_MIPS.datapath.pc.pc,U\_MIPS.datapath.ir.ir );

clk = 0 ;

rst = 0 ;

其他语句