Project1VerilogHDL开发多周期处理器-MIPS-Lite1

V1.0@2014.7.3

高小鹏

# 设计说明

1. 处理器应支持MIPS-Lite1指令集。
   1. MIPS-Lite1＝{addu，subu，ori，lw，sw，beq，jal}。
   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. 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)、RF (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. 为供你参考的数据通路架构图。该图更多的是让你对多周期数据通路有认识。
   1. 该图支持MIPS-Lite1指令集。
   2. 如果你对数据通路做了比较大的调整，请注意务必不要与要求13矛盾。

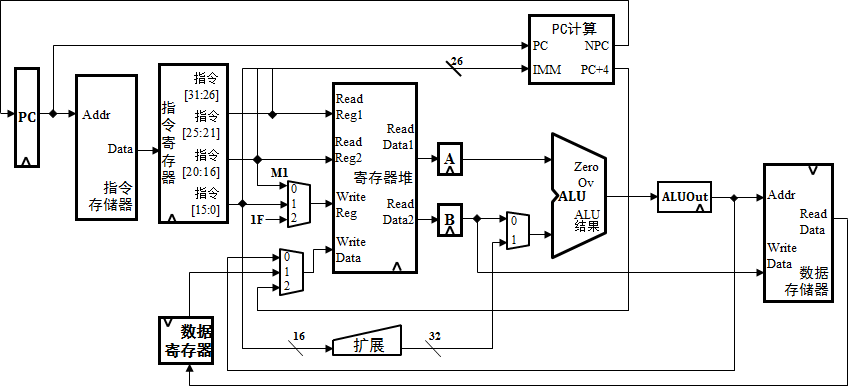


Figure1多周期数据通路(供参考)

1. 多周期数据通路应必须包括PC、NPC、IM、DM这4个独立模块。其中：
   1. IM：容量为4KB(32bit×1024字)。
   2. DM：容量为4KB(32bit×1024字)。
2. 层次及模块实例化命名必须满足下列要求：
   1. 本project的顶层设计文件命名：mips.v。
   2. PC必须被实例化命名：U\_PC。下面代码为示例。

|  |
| --- |
| pc U\_PC(…) ; // 实例化PC(程序计数器) |

* 1. 指令存储器必须被实例化命名：U\_IM。
  2. 数据存储器必须被实例化命名：U\_DM。
  3. 寄存器文件必须被实例化命名：U\_RF。

1. 建议datapath中的每个module都由一个独立的VerilogHDL文件组成。
   1. 建议所有mux（包括不同位数、不同端口数等）都建模在一个mux.v中。同一个.v文件中可以有多个module。
2. 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1个或多个头文件中。
3. PC复位后初值为0x0000\_3000，目的是与MARS的Memory Configuration相配合。
   1. 现场测试用的测试程序将通过MARS产生，其配置模式如所示。

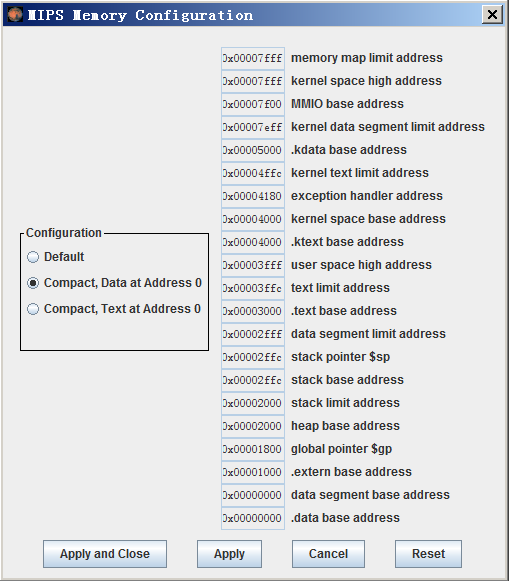


Figure2 MARS存储配置模式(MARS memory configuration)

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：
   1. 按类构造状态分支，而不是按指令。
   2. 状态分支不宜过多。

# 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。
2. PC模块定义(参考样例)
3. 基本描述

PC的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。

1. 模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| NPC[31:2] | I | 下条指令的地址 |
| PCWr | I | PC写使能  1：允许NPC写入PC内部寄存器  0：禁止NPC写入PC内部寄存器 |
| clk | I | 时钟信号 |
| rst | I | 复位信号。  1：复位  0：无效 |
| PC[31:2] | O | 30位指令存储器地址(最低2位省略) |

1. 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 | 在每个clock的上升沿保存NPC，并输出。 |

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, din, DMWr, clk, dout ) ;  input [11:2] addr ; // address bus  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. 所有指令都应被测试充分。
2. 本project提供Test\_7\_Instr.asm的测试程序，可在MARS中运行观察结果。
3. 也可使用MARS自行构造测试程序，并进行测试。要求每条指令至少出现1次以上。
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。

# 问答【WORD】

1. 状态机设计通常没有唯一答案。为2个均可行的状态机。状态机设计思路的主要差异在于在译码状态后，根据指令的性质设置了不同的状态分支。每位设计者的设计构思可能都不尽相同。请详细描述你的设计构思，特别是描述你为什么要这样设计状态分支。

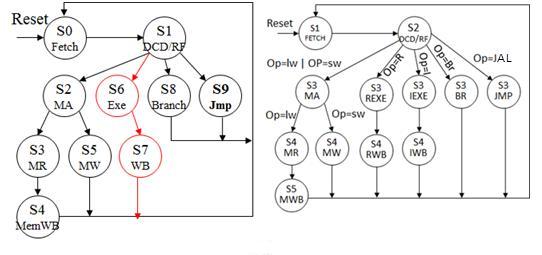


Figure3多周期控制器状态机参考设计

# 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、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。

# 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。

# 开发与调试技巧

1. 由于MIPS部分指令涉及非常复杂的运行模式，故你在阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》时可能存在困难。为此我们针对本课程定义的MIPS-C指令集，从原文手册中提取了必要的内容，编写成《MIPS-C指令集.pdf》以便于你理解。简化主要是去除所有与exception、delay slot有关的描述。
   1. MIPS-C指令集描述见《MIPS-C指令集.pdf》。
   2. 《MIPS-C指令集.pdf》是对《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》解读，我们不能保证100%正确。你应该对照着阅读。
2. 建议先在MARS中编写测试程序并调试通过。注意memory configuration的具体设置。
3. 由于MIPS-Lite4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。
4. 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.U\_pc.pc, U\_MIPS. instr );

clk = 0 ;

rst = 0 ;

其他语句

end