Project5 VerilogHDL 开发多周期处理器(1)

V1.1@2013.12.6

高小鹏

一、 设计说明

1. 处理器应 MIPS-Lite4 指令集。

   a) MIPS-Lite4＝{MIPS-Lite3，lb，lbu，lh，lhu，sb，sh，slti }。

   b) MIPS-Lite3＝{MIPS-Lite2，addi，addiu， slt，j，jal，jr}。

   c) MIPS-Lite2 指令集：addu，subu，ori，lw，sw，beq，lui。

   d) 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。

批注 [GXP1]: 1.支持了这些指令，基本上就可以用 C 语言编写简单的测试程序，然后用 GCC 编译 C 程序产生 MIPS 汇编了。具体内容参见第 31 条。

二、 设计要求

3. 多周期处理器由 datapath(数据通路)和 controller(控制器)组成。

   a) 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。

4. Figure1 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。

   a) 该图仅能支持{ addu，subu，ori，lw，sw，beq，jal }。以 lb 指令为例，由于需要将从存储器中读出的 32 位数据中的某个特定字节提取并做符号扩展为 32 位数据后才能写入 GPR，因此需要在数据寄存器和 GPR 之间再设置一个新的功能单元—存储器数据扩展单元。

   b) 如果你做了比较大的调整，请注意务必不要与要求 13 矛盾。

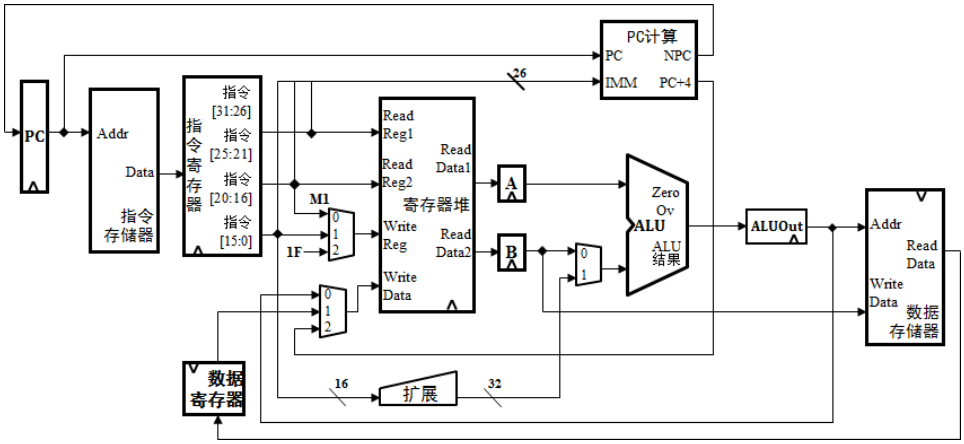

Figure1 多周期数据通路(供参考)

5. 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：
   a) IM：容量为 4KB(32bit×1024 字)。
   b) DM：容量为 4KB(32bit×1024 字)。
6. 层次及模块实例化命名必须满足下列要求：
   a) 本 project 的顶层设计文件命名：mips.v。
   b) 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)
```

   c) 指令存储器必须被实例化命名：U_IM。
   d) 数据存储器必须被实例化命名：U_DM。
   e) 寄存器文件必须被实例化命名：U_RF。
7. 建议 datapath 中的每个 module 都由一个独立的 VerilogHDL 文件组成。
   a) 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。可以有多个 module。
8. 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。
9. PC 复位后初值为 0x0000_3000，目的是与 MARS 的 Memory Configuration 相配合。
   a) 现场测试用的测试程序将通过 MARS 产生，其配置模式如 Figure2 所示。

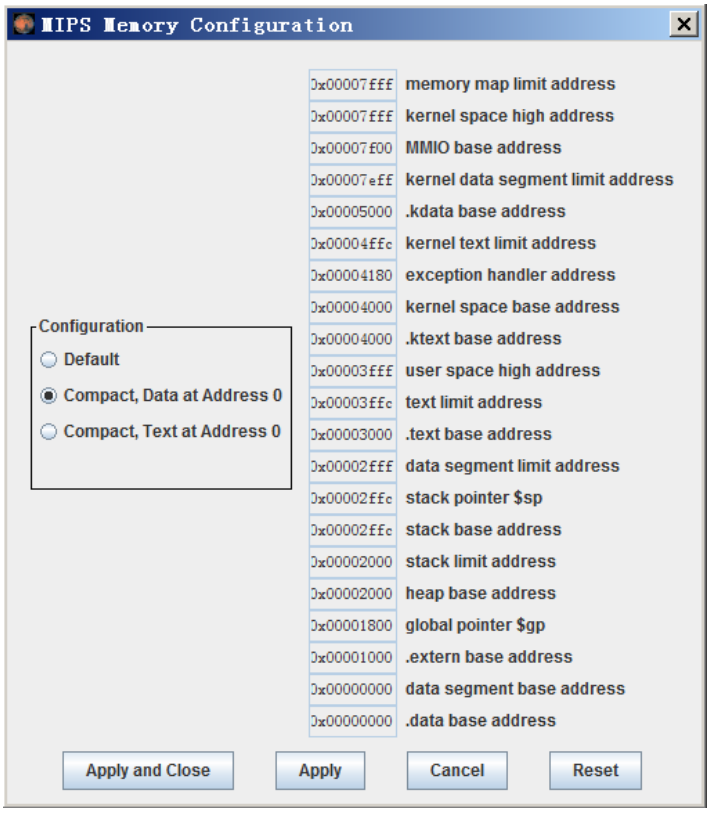

Figure2 MIPS 存储配置模式(MARS memory configuration)

10. 建议你用工程化方法来构造数据通路和状态机。具体内容请参见随 project 提供的 PPT。

    a) 你初次接触工程化方法时可能不是非常理解其价值。但一旦当你运用这个方法后(特别是在后续 project 中)，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。

    b) 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。

    c) 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。

三、 模块定义【WORD】

11. 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。

12. PC 模块定义(参考样例)

(1) 基本描述

PC 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。

(2) 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
Reset	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)

批注 [GXP2]: 注意：PC 有了新增信号。原来在单周期中的某些功能模块也可能需要做针对性调整。

(3) 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。

13. 下列模块必须严格满足如下的接口定义：

a) 你必须在 VerilogHDL 设计中建模这 3 个模块。

b) 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, we, clk, dout ) ;`
	`input [11:2] addr ; // address bus`
	`input [3:0] be ; // byte enables`
	`input [31:0] din ; // 32-bit input data`
	`input we ; // memory write enable`

批注 [GXP3]: 增加了 4 位字节使能！

```
    input           clk ;   // clock
    output  [31:0]  dout ;  // 32-bit memory output
```

四、 测设要求

14. 所有指令都应被测试充分。
15. 本 project 不提供基准测试程序了。
    a) 你可以在 project4 提供的基准测试程序上改造以产生你的基准测试程序。
16. 构造至少包括 40 条以上指令的测试程序，并测试通过。
    a) MIPS-Lite4 定义的每条指令至少出现 1 次以上。
    b) 必须有函数，并至少 1 次函数调用。
17. 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。
18. 详细说明你的测试程序原理及测试结果。【WORD】
    a) 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。
    b) 每条汇编指令都应该有注释。

五、 问答【WORD】

19. 状态机设计通常没有唯一答案。Figure3 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。

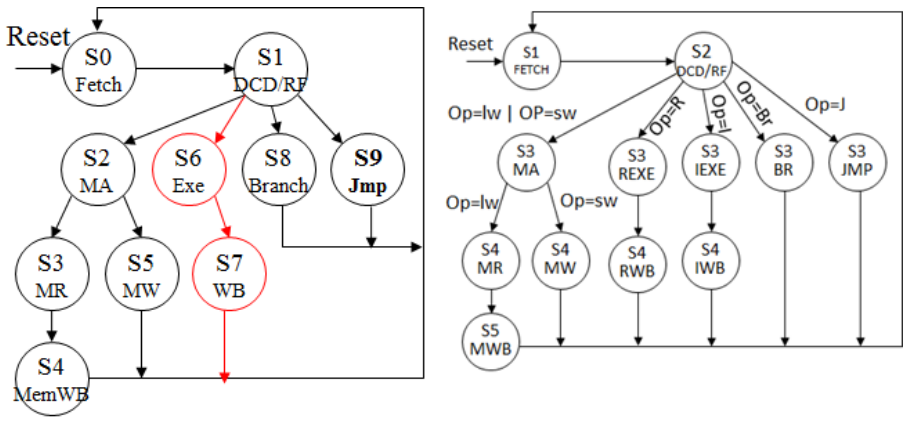

Figure3 多周期控制器状态机参考设计

六、 成绩及实验测试要求

20. 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。
21. 实验测试时，你需要展示你的设计并证明其正确性。
    a) 应简洁的描述你的验证思路，并尽可能予以直观展示。
22. 实验指导教师会临时增加若干条指令，你需要在规定的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。
    a) 考查时，教师将用专用 testbench 和 code.txt 检测代码执行情况。

七、 其他要求

23. 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。
24. 时间要求：各班实验指导教师指定。
25. 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。
26. 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。

八、 开发与调试技巧

27. 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。
    a) 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。
    b) 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

    c) 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

d) 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

e) Figure4 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)。

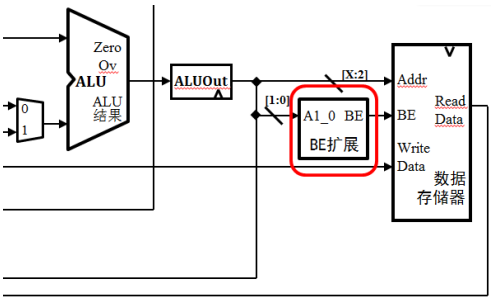

Figure4 BE 扩展

28. 对于 lb、lbu、lh、lhu 来说，Figure1 是不能支持的。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。

a) 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。

29. 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。

a) MIPS-C 指令集描述见《MIPS-C 指令集.pdf》。

b) 《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。

30. 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。
31. 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：
    a) 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。
    b) S2：生成 MIPS 汇编。用 cygwin 下的 gcc 编译测试程序。
    c) 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。
    d) 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。
    e) 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。
32. 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。

a) 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
```

b) 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。

c) 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
```

批注 [gxp4]: 注意：如果写了 2 个 monitor，那么只有最后一个 monitor 会起作用。因此，你需要把关心的变量都放在同一个 monitor 语句中。

批注 [gxp5]: 最后一个 pc 代表 pc.v 中定义的寄存器变量（真正的 PC）