VerilogHDL 开发流水线处理器（1）

V1.0@2014.11.22

高小鹏

一、 设计说明

1. 处理器应 MIPS-Lite2 指令集。
   a) MIPS-Lite2＝{MIPS-Lite1，j，jal，jr}。
   b) MIPS-Lite1＝{addu，subu，ori，lw，sw，beq，lui}。
   c) 所有运算类指令均可以不支持溢出。
2. 处理器为流水线设计。

二、 设计要求

3. 关于你设计的流水线的顶层视图，我们建议采用《数字设计和计算机体系结构》中的图 7-58。

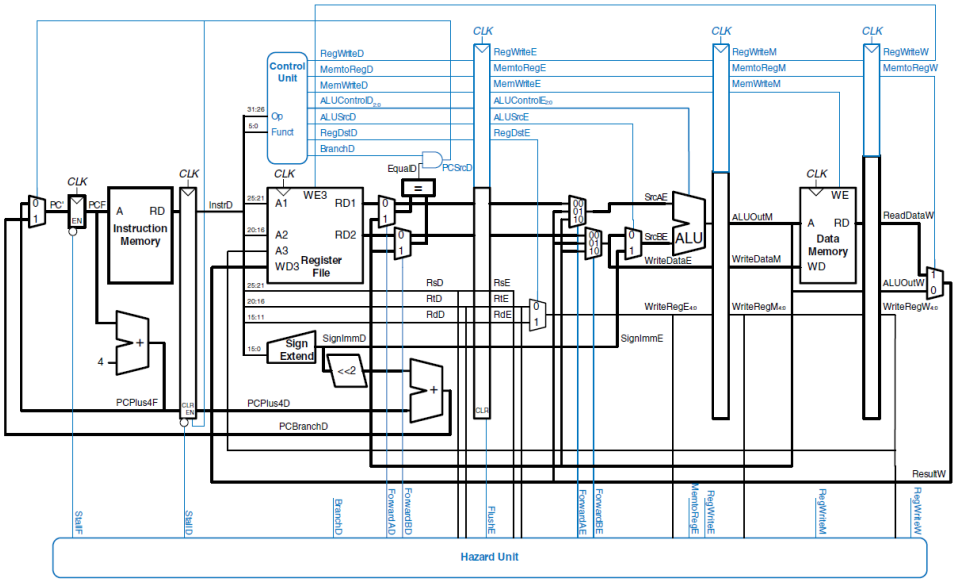

Figure 7.58 Pipelined processor with full hazard handling

   a) 该图仅仅是参考，并不能支持本 project 的全部指令。
   b) 务必将控制器与冲突单元(上图中的 Hazard Unit)分离，这样便于你开发。即定义独立 controller 模块和 hazard 模块。

c) 控制器的设计与单周期没有差别。

4. 流水线的设计以追求性能为第一目标，因此必须尽最大可能支持转发以解决数据冒险。
5. 对于 b 类和 j 类指令，流水线设计必须支持延迟槽，因此设计需要注意使用 PC+8。
   a) 流水线必须支持 NOP 指令。
6. 为了解决数据冒险而设计的转发数据来源必须是某级流水线寄存器，不允许从功能部件的输出开始。
   a) 以上图为例，ALU 的输出不允许直接作为转发输入，只能是 ALUOutM。
7. 指令存储器(IM，instruction memory)和数据存储器(DM，data memory)要求如下：
   a) IM：容量为 1KB(32bit/word×256word)。
   b) DM：容量为 4KB(32bit/word×1024word)。
8. 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 Figure1 所示。

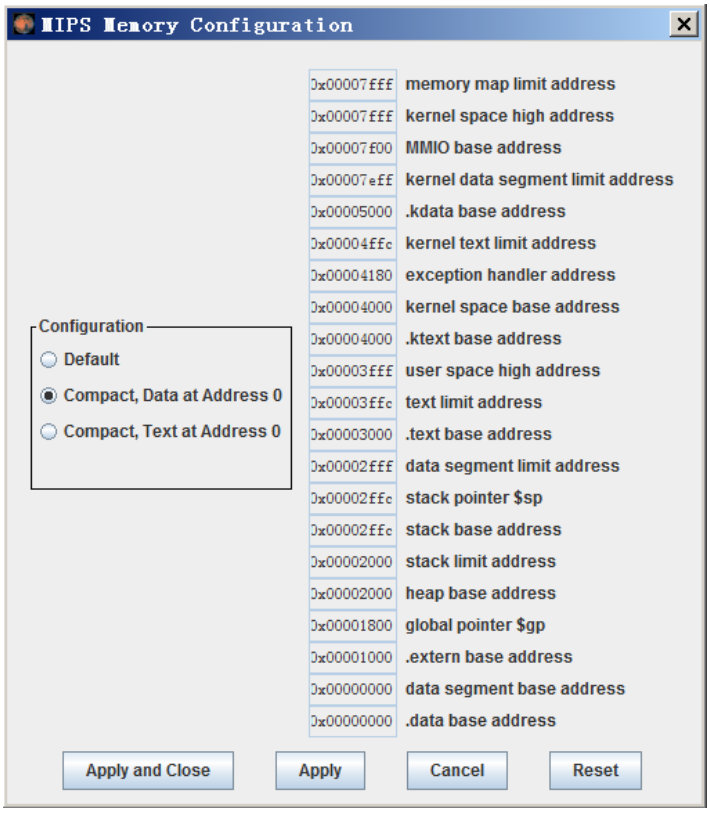

Figure1 MIPS 存储配置模式(MARS memory configuration)

三、 命名规范化

9. 规范化是 IT 产业高速发展的重要原因之一。你应该从现在开始就培养自己的工程规范性，其中设计中的命名规范就是你应该迅速建立的良好习惯。良好的命名规范有助于你高效而且正确的完成开发工作。
10. 无论是模块名、还是变量名，都应该有意义。望文生义非常重要！
    a) 决不能出现 aaa、bbb 之类完全不知所云的命名！
11. 规律性是规范化的关键要素之一。有规律的命名可以让你迅速清楚变量的含义、有效范围等信息。
    a) 你应该考虑合理使用大小写。如 ALUOp 可能比 aluop 更好理解些。
    b) 你应该考虑规律性的使用后缀。例如所有的寄存器写使能都用 Wr 作为后缀，如 DMWr、RFWr。所有的控制码都用 Op 作为后缀，如 ALUOp，RegOp。
    c) 区分 module 定义和 module 实例化的一个重要方式是在实例化用有规律

和有意义的名字来，例如

3 输入/32 位的 MUX 被命名为：

```
module MUX_3_32b(。。。)
```

该 MUX 在转发阶段被实例化时命名为：

```
MUX_3_32b  U_MUX_ALU1_Forword(。。。) ;
```

d) 建议你用下划线加流水线级的方式来命名变量，例如 RegWr_D 和 RegWr_E 分别代表处于 Decode 级和 Exe 级的 RegWr 信号。

12. 你可以根据自己的喜好去定义自己的命名规律。但务必注意的是确保命名上的一致性！
13. 使用任何有意义的命名时，都会让你在键盘输入时会花费更多时间。但从我们积累的经验来看，这点代价是值得的！请谨记这点！

四、 测设要求

14. 你编写的汇编测试程序必须确保所有指令都应被测试充分。
15. 相关性是你的测试重点。你的测试程序必须去充分的测试数据相关和分支相关。构造相关性测试的基本思路是对各类指令组合{读操作数的位置，产生有效结果的位置}。
16. 下面是测试流水线 addu 指令相关性正确性的测试例子。(这个例子不能满足你全部的测试需求！)

a) 假定要测试运算类 addu 的相关性的正确性。

b) 由于 addu 最后必须读取操作数的位置在 EX 阶段，那么与之相关的指令就只能是 N-1(MEM 阶段)，N-2(WB 阶段)。即必须让 N-1 条指令或 N-2 条指令的回写寄存器与 addu 的读取寄存器必须相同。

c) 能够产生有效结果的指令只能是运算类指令和 lw 指令，因此对于 addu 来说至少(事实上还要更多的测试组合)包括如下测试用例。测试类型 X-Y-Z 含义如下：

i. X：产生冲突的前序指令类型。其中 R 代表 R 型运算类产生计算结果。

ii. Y：前序指令在哪个阶段与 addu 产生冲突。其中 M 代表 MEM 阶段，W 代表 WB 阶段。

iii. Z：代表冲突的寄存器，其中 RS 代表 rs 寄存器，RT 代表 rt 寄存器)

用例编号	测试类型	前序指令	冲突位置	冲突寄存器	测试序列
1	R-M-RS	subu	MEM	rs	`subu $1, $2, $3` `addu $4, $1, $2`
2	R-M-RT	subu	MEM	rt	`subu $1, $2, $3` `addu $4, $2, $1`

3	R-W-RS	subu	MEM	rs	`subu $1, $2, $3`
					`instru 无关`
					`addu $4, $1, $2`

4	R-M-RT	subu	MEM	rt	`subu $1, $2, $3`
					`instru 无关`
					`addu $4, $2, $1`

5	I-M-RS	ori	MEM	rs	`ori $1, $2, 1000` `addu $4, $1, $2`
6	I-M-RT	ori	MEM	rt	`ori $1, $2, 1000` `addu $4, $2, $1`

7	I-W-RS	ori	MEM	rs	`ori $1, $2, 1000`
					`instru 无关`
					`addu $4, $1, $2`

8	I-W-RT	ori	MEM	rt	`ori $1, $2, 1000`
					`instru 无关`
					`addu $4, $2, $1`

9	LD-M-RS				
10	LD-M-RT				
11	LD-W-RS				
12	LD-W-RT				

17. 当你按照上述方法构造测试用例时，其工作本质是进行覆盖性分析。做覆盖性分析时未必一定要逐条指令来分析，可以按照类别来。
    a) 建议你参考《MIPS-C 指令集》中的指令分类，而不是简单的按照 R-I-J 三类。原因是 R-I-J 三类是从指令格式出发的，而你现在则需要从指令功能出发。指令格式与指令功能很难完全对应，例如你会发现 JALR(本次不要求)就是 R 类格式，但其功能则与其他 R 类格式的指令大相径庭。
18. 如果你仔细思考，就会发现，上述冲突的覆盖性分析与流水线的设计其实是相互促进的。
    a) 你的测试用例集的构造是有理论指导的，而不是凭空瞎想的。
    b) 什么样的流水线设计应该怎么测试，做哪些测试就基本合格了，你也会心中有数了。

19. 本 project 不提供基准测试程序了。你应该仿照上述思路自行构造测试 asm。
    a) 强烈建议你先进行各类冲突的组合分析。
    b) 在组合分析的指导下构造测试用例表。
    c) 最后才编写测试用例 asm。
20. 请思考并回答：lw-sw 指令也存在数据相关，并且理论上也可以通过转发来化解。但遗憾的是教科书没有给出这个例子，并且流水线也不支持相应的转发机制。按照上述方法，你应该能分析出 lw-sw 的相关性。请给出案例，并给出如何通过转发消除这个数据相关。【WORD】
21. 请思考并回答：对于本 project 定义的指令集来说，如果进行较为全面的相关性测试，最少需要构造多少个测试用例。给出你的计算依据及过程。【WORD】
22. 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、 成绩及实验测试要求

23. 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。
24. 实验测试时教师可能要求你在你的设计中增加新的指令。
25. 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。

六、 Project 提交

26. 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。
27. 时间要求：实验指导教师指定。
28. 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。
29. 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。

七、 开发与调试技巧

30. 在没有想清楚流水线的时候就在 VerilogHDL 层次进行编码是不明智的。前

人大量经验与教训告诉我们，对于大工程而言，在代码层次上做设计是短视的行为，表面看起来可以在早期迅速看到初步结果，但从整个项目的过程来看，开发人员将付出巨大的频繁试错的代价。任何一个项目的开发都可以至少划分为 2 个层次：设计与实现。我们强烈建议你在没有很好的构思出你的设计前，不要贸然的去实现。

a) 设计：在 EXCEL 中进行详尽的设计并进行逻辑推演，是高效率的设计方法。

b) 实现：当你认为自己的设计没有问题或基本没有问题时，再用 VerilogHDL 将你的设计描述出来。

c) 这样的物理上的 2 个分割的开发层次有助于提高效率和开发正确率。

31. 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 [gxp1]: 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 [gxp2]: 最后一个 pc 代表 pc.v 中定义的寄存器变量（真正的 PC）