

# Project1 VerilogHDL 开发多周期处理器-MIPS-Lite1

## 一、 设计说明

1. 处理器应支持 MIPS-Lite1 指令集。
  - a) MIPS-Lite1 = {addu, subu, ori, lw, sw, beq, jal}。
  - b) 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。

## 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)、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。
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) 该图支持 MIPS-Lite1 指令集。

b) 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。

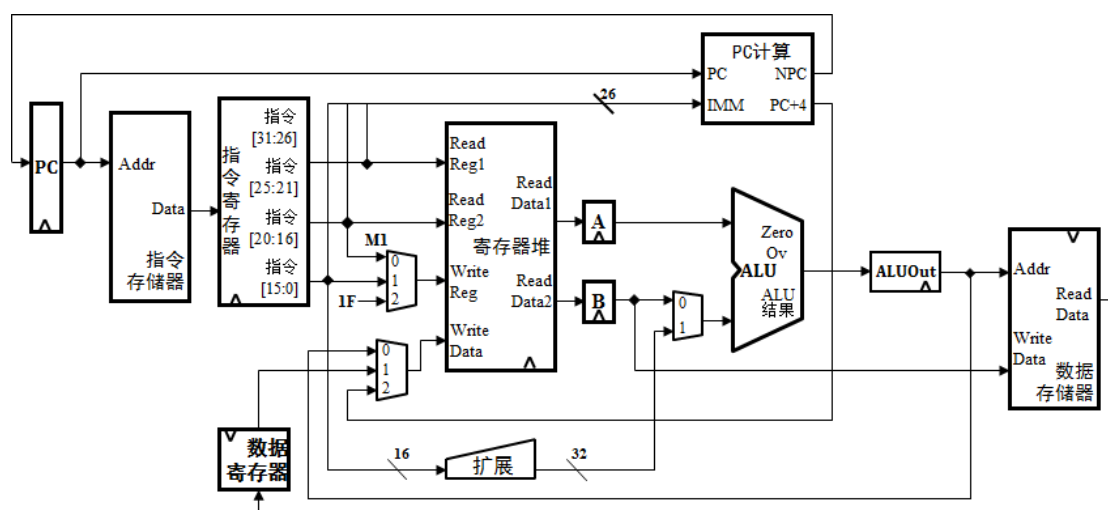


Figure1 多周期数据通路(供参考)

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。

同一个.v 文件中可以有多个 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。

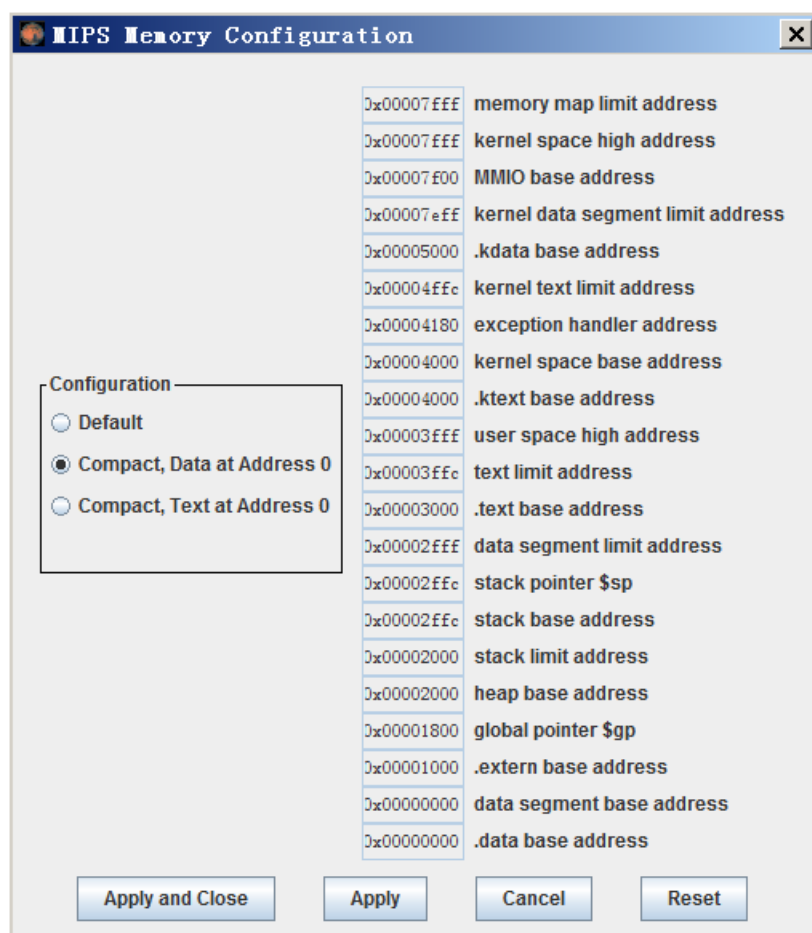


Figure2 MARS 存储配置模式(MARS memory configuration)

10. 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：
- 按类构造状态分支，而不是按指令。
  - 状态分支不宜过多。

### 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 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。

#### (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
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)

### (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. IM 模块定义

### (1) 基本描述

### (2) 模块接口

信号名	方向	描述
AIn[31:0]	I	
DO[31: 0]	O	

### (3) 功能定义

序号	功能名称	功能描述
1		
2		

## 14. 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	<pre>module mips(,clk, rst) ;     input          clk ;    // clock     input          rst ;// reset</pre>
im.v	<pre>module im_4k( addr, dout ) ;     input  [11:2]  addr ;  // address bus</pre>

	output [31:0] dout ; // 32-bit memory output
dm.v	<pre> 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 </pre>

#### 四、 测设要求

15. 所有指令都应被测试充分。
16. 本 project 提供 Test\_7\_Instr.asm 的测试程序，可在 MARS 中运行观察结果。
17. 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。
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。

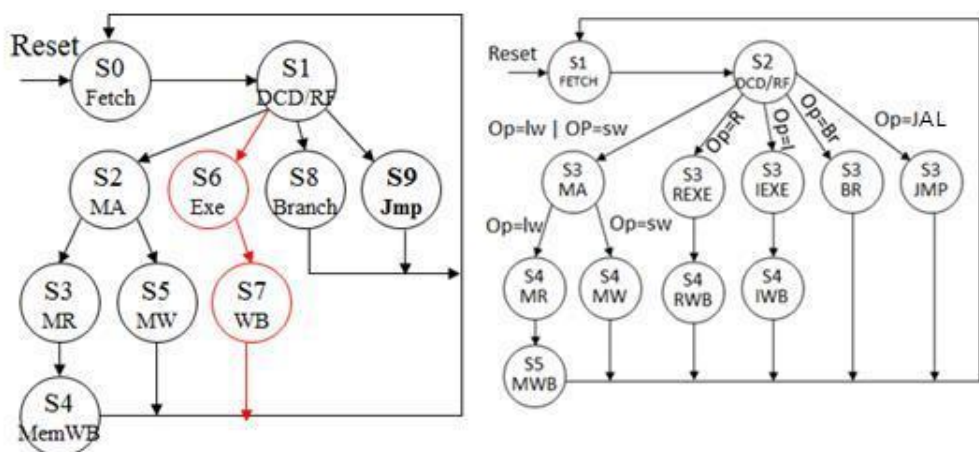


Figure3 多周期控制器状态机参考设计

## 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、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。

## 七、 其他要求

- 22. 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。
- 23. 时间要求：实验指导教师指定。
- 24. 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。
- 25. 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

- 26. 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。
- 27. 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。
- 28. 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。

29. 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 begin
    $monitor("PC = %8X, IR = %8X", U_MIPS.U_pc.pc, U_MIPS.
instr );

    clk = 0 ;

    rst = 0 ;

    其他语句

end

```