Project3 VerilogHDL 开发多周期处理器-MIPS-Lite2

一、 设计说明

1.  处理器应支持 MIPS-Lite2 指令集。

a) MIPS-Lite2＝{MIPS-Lite1，lb，sb}。

b) MIPS-Lite1＝{addu，subu，ori，lw，sw，beq，lui，addi，addiu，slt，jal，jr}。

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)、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](#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。

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。

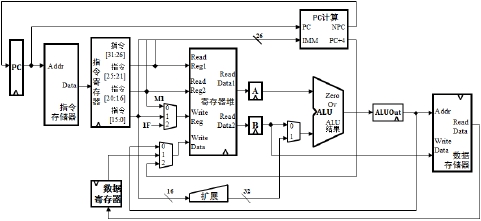


Figure1 多周期数据通路(供参考)

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。

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| NPC[31:2] | I | 下条指令的地址 |

(2)  模块接口

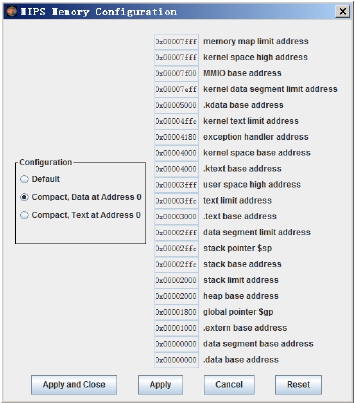


Figure2 MIPS 存储配置模式(MARS memory configuration)

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：

a) 按类构造状态分支，而不是按指令。

b) 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 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。

|  |  |
| --- | --- |
| 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, we, clk, dout ) ;  input       [11:2]     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 |

|  |  |  |
| --- | --- | --- |
| 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) |

|  |  |  |
| --- | --- | --- |
| 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。 |

批注**:** 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。

a) 你必须在 VerilogHDL 设计中建模这 3 个模块。

(3)  功能定义

13.  下列模块必须严格满足如下的接口定义：

四、 测设要求

b)   不允许修改模块名称、端口各信号的名称/类型/位宽。

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。

16. 构造至少包括 40 条以上指令的测试程序，并测试通过。

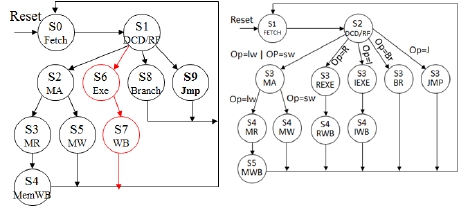
a) 定义的每条指令至少出现 1 次以上。

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、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。

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. 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