

实验六：VerilogHDL 开发单周期处理器

目标：

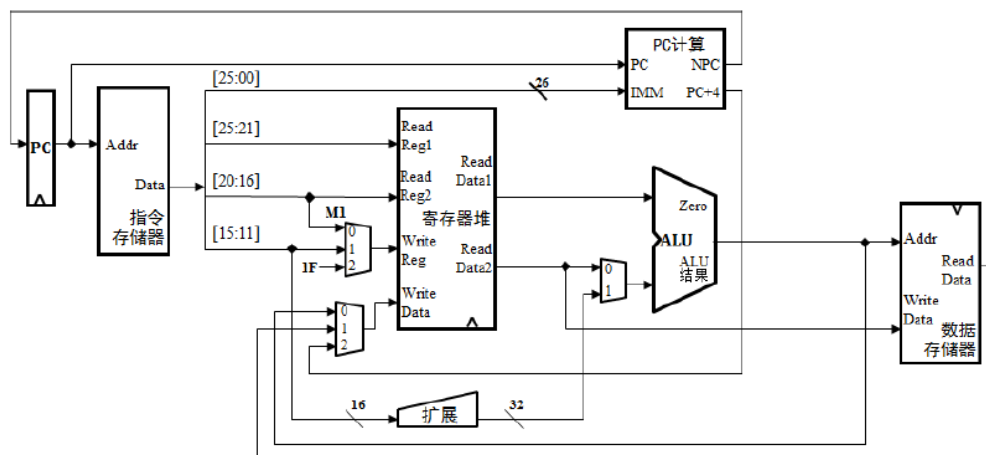
1. 使用Verilog开发基于单周期的处理器。
2. 使用Vivado对开发的处理器进行模拟仿真。
3. 编写测试用汇编，使用Mars工具汇编成为机器代码，并使用它调试测试你开发的处理器。

要求

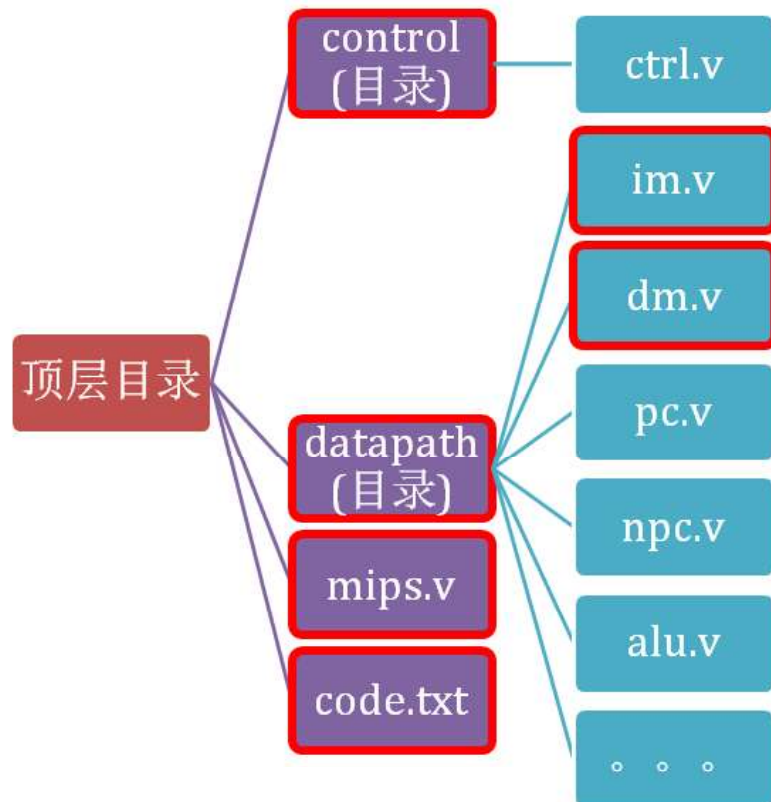
1. 请不要抄袭，可以与同学讨论，但不要直接抄袭同学的代码和实验报告。
2. 请认真完成实验报告，并在你认为关键的位置插入屏幕截图。
3. 请在截止日期（初定为2020.12.2日23：55）前将**Verilog源代码**和**实验报告**提交至Unicourse+上。

说明

1. 处理器应支持MIPS-Lite2指令集。
 - MIPS-Lite1 = { addu, subu, ori, lw, sw, beq, lui }
 - MIPS-Lite2 = { MIPS-Lite1, jal, jr }
2. 处理器为单周期设计。
3. 单周期处理器由 datapath（数据通路）和 controller（控制器）组成。
 - 数据通路可以由以下模块组成：
 - PC（程序计数器）
 - NPC（NextPC计算单元）
 - GPR（通用寄存器组，包含32个寄存器）
 - ALU（算术逻辑单元）
 - EXT（扩展单元）
 - IM（指令存储器-只读）
 - DM（数据存储器-可读可写）
 - IM规格：容量为4KB（32bit*1024字）。
 - DM规格：容量为4KB（32bit*1024字）。
4. 我为你提供了参考的数据通路架构图：



- 我不确保这个数据通路完全适用于本实验。
 - 鼓励你从数据通路的功能合理划分的角度自行设计更好的数据通路架构。
 - 如果你做了比较大的调整，请务必注意不要与说明5矛盾。
5. 整个project必须采用模块化和层次化的设计。
- 我为你提供了参考的目录结构，注意其中红色边框的目录名称和文件名称不允许调整。



- 顶层设计文件命名为mips.v。
 - 建议datapath中每个module由一个独立的Verilog文件组成。模块化的要领不是从模块的大小出发，而是从模块的功能独立性出发。
 - 一些功能相近的模块，比如mux（不同位数，不同端口数的所有多路选择器）都建模在一个mux.v文件中。
6. code.txt中存储的是指令码。
- 测试时，code.txt的格式为每条指令占用1行，指令以十六进制文本码形式存储。
 - 请使用verilog建模指令存储器IM时，以读取code.txt的方式载入指令，可以自行学习\$readmemh指令。

7. 模块约定

- 实验四的PC模块可以直接应用于本实验，如果你要进行修改，请注意，收到复位指令后，第一条指令的地址为0x0000_3000。
- 顶层实体必须按如下方式定义：

```

1 module mips(clk,rst);
2     input  clk; //时钟信号
3     input  reset; //复位信号

```

- 数据存储器必须按如下方式定义：

```

1 module dm_4k(addr,din,we,clk,dout);
2     input [11:2] addr //地址总线
3     input [31:0] din //写入数据
4     input we; //写入使能信号
5     input clk; //clock
6     output [31:0] dout; //输出数据
7
8     reg [31:0] dm [1023:0];

```

- 指令存储器必须按如下方式定义：

```

1 module im_4k(addr,dout);
2     input [11:2] addr; //地址总线
3     output [31:0] dout; //输出指令
4
5     reg [31:0] im [1023:0];

```

8. 你所完成的测试文件的要求：

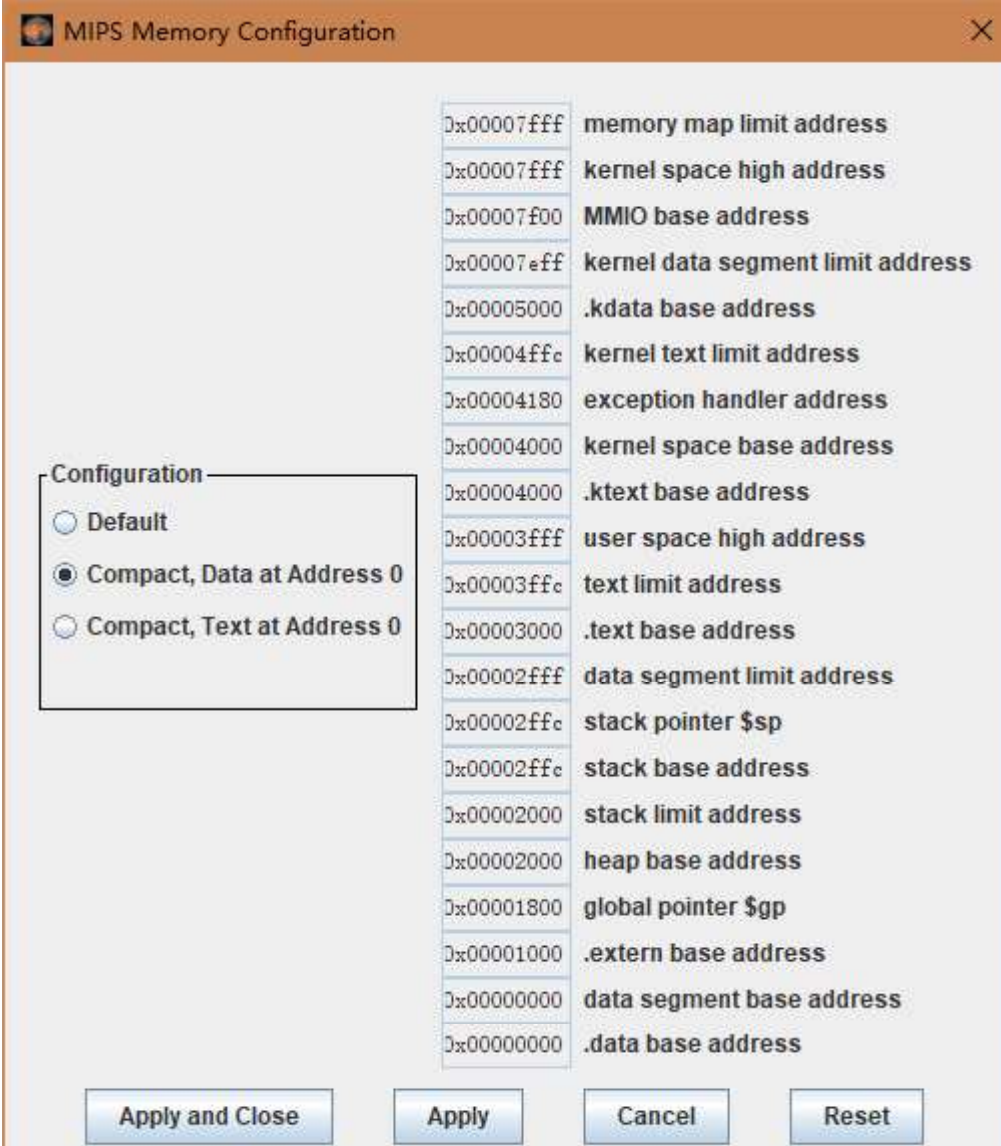
- 所有指令都应被充分测试，每条指令至少出现1次以上。
- 函数相关指令（jal和jr）是较为复杂的指令，所以为了充分的测试，你必须在测试程序中组织一个循环，并在循环中多次函数调用，以确保正确实现了这2条指令。

9. 你的实验报告必须至少包含以下内容：

- 每个模块的功能设计
- 控制器的设计过程
- 验证程序的汇编程序，思路以及预期结果

10. 我们为你提供了MIPS32指令集的格式及功能详解。

11. 调试时你可以使用Mars工具将汇编程序编译成机器码导入code.txt中。请注意Mars工具的内存设置请依据如下：



MIPS Memory Configuration

Configuration

☐ Default
☒ Compact, Data at Address 0
☐ Compact, Text at Address 0

0x00007fff	memory map limit address
0x00007fff	kernel space high address
0x00007f00	MMIO base address
0x00007eff	kernel data segment limit address
0x00005000	.kdata base address
0x00004ffe	kernel text limit address
0x00004180	exception handler address
0x00004000	kernel space base address
0x00004000	.ktext base address
0x00003fff	user space high address
0x00003ffe	text limit address
0x00003000	.text base address
0x00002fff	data segment limit address
0x00002ffe	stack pointer \$sp
0x00002ffe	stack base address
0x00002000	stack limit address
0x00002000	heap base address
0x00001800	global pointer \$gp
0x00001000	.extern base address
0x00000000	data segment base address
0x00000000	.data base address

你可以学习使用`$display`指令使你的调试更加简便。

- 本次实验你需要自己编写`test_bench`文件以生成时钟信号，在测试时，我将使用特定的`test_bench`文件及`code.txt`文件对你的处理器进行正确性验证。为了方便测试，我们约定每条指令都在一个时钟周期内执行完毕。