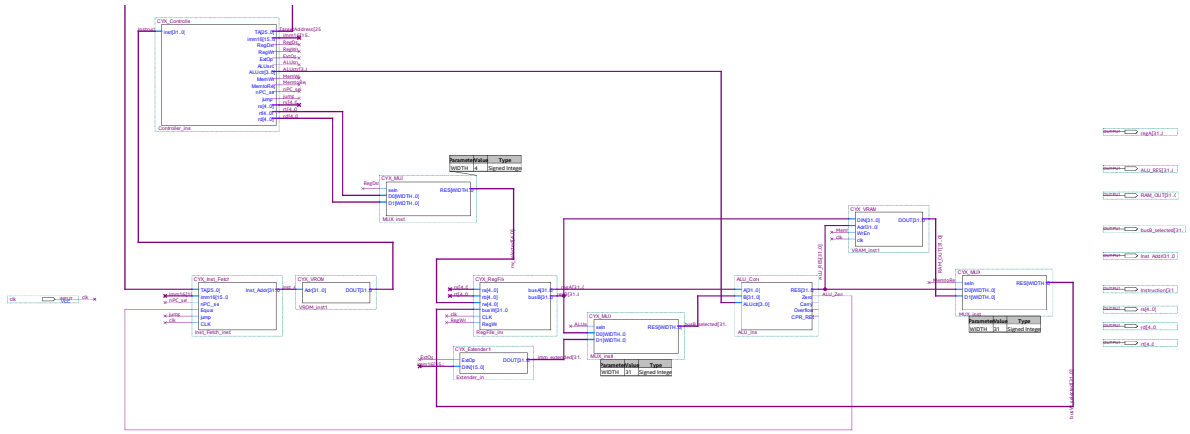
MIPS 单周期 CPU 设计 实验报告

程宇笑 自 84 2018010888

一、数据通路结构设计

根据所学知识设计简单集指令单周期 MIPS 数据通路。为了简化设计流程，在仿真设计工程中暂不加入涉及外设硬件的输入输出模块，同时使用 Verilog 描述的寄存器堆和多路选择器模拟的 RAM 和 ROM。总体数据通路电路如下：



该电路总共使用 9 个通过 Verilog 描述的模块。

1.1 各模块设计

该单周期 CPU 的 9 个模块分别为 ALU 核心、寄存器堆、取指令模块、控制器、多选器、立即数扩展器、虚拟 RAM、虚拟 ROM。

ALU 核心 (ALU_Core)：使用实验二设计的 ALU 模块，但需要将比较结果输出端口与主输出端口合并，即设置为比较功能时如果 A 与 B 相等则输出 32 位的 0。

寄存器堆 (RegFile)：含有 8 个 32 位寄存器，使用 Verilog 描述其初始化和读写功能。

```
assign busA = regs[ra];
assign busB = regs[rb];

always @ (posedge CLK) begin
   regs[0] = REG0_DATA;
   if (RegWr && (rw != 0)) begin
      regs[rw] = busW;
   end
end
```

取指令模块 (Inst_Fetch)：支持顺序执行、PC 相对寻址（beq 指令）和伪直接寻址（j 指令）。除此之外，还包含立即数扩展功能（符号位扩展 14 位，结尾加两个 0）和 PC

寄存器，计算指令地址完成后传给虚拟 ROM。

虚拟 ROM(VROM)：为了方便调试和开发，直接使用 Verilog 描述 ROM 内含数据，通过类似于多选器的描述方式实现 ROM 功能。读取到指令内容后交给控制器。

```
case (Adr)

  0: DOUT = 32'h8c220000; //lw $2, 0($1)
  4: DOUT = 32'h8c230004; //lw $3, 4($1)
  8: DOUT = 32'h8c240008; //main: lw $4, 8($1)
  ...
  default: DOUT = 32'h0;
endcase
```

控制器 (Controller)：读取指令内容，解析出各字段含义及控制信号。该模块本质上是组合逻辑电路模块。

多选器 (MUX)：用于控制信号对数据的选择。

立即数扩展器 (Extender16)：用于立即数的输入，分为 0 扩展和符号扩展两种模式。

虚拟 RAM(VRAM)：使用 Verilog 描述，其行为与寄存器堆类似。可以在任意时刻读取数据，在时钟上升沿写入数据。

二、仿真分析

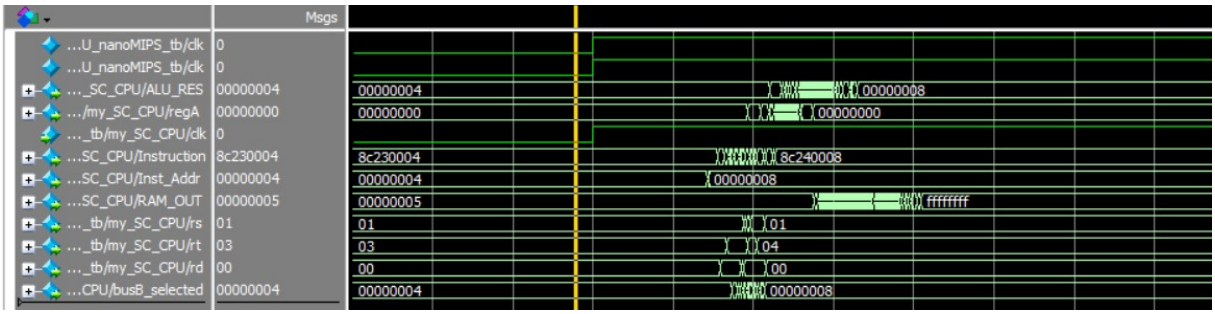
使用如下的 MIPS 汇编程序进行测试，其本身是一个循环。在内存的 0，4，8 地址中预先填充数据 1，2，-1。设置输出信号，然后进行时序仿真。

```
   lw $2, 0($1)
   lw $3, 4($1)
main:
   lw $4, 8($1)
   add $6, $2, $6
   sub $5, $3, $4
   sw $5, 8($1)
   or $5, $4, $3
   and $5, $4, $3
   slt $5, $6, $3
   beq $6, $3, exit
   j main
exit: lw $2, 0($3)
   j main
```

2.1 数据通路时序分析

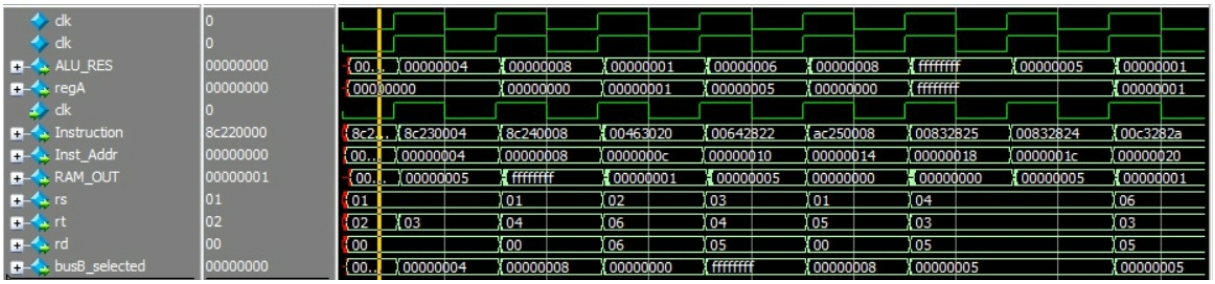
截取单个时钟上升沿之后的波形图进行观察，可以发现信号发生改变的时间顺序可以列举如下。该时序符合理论设计。

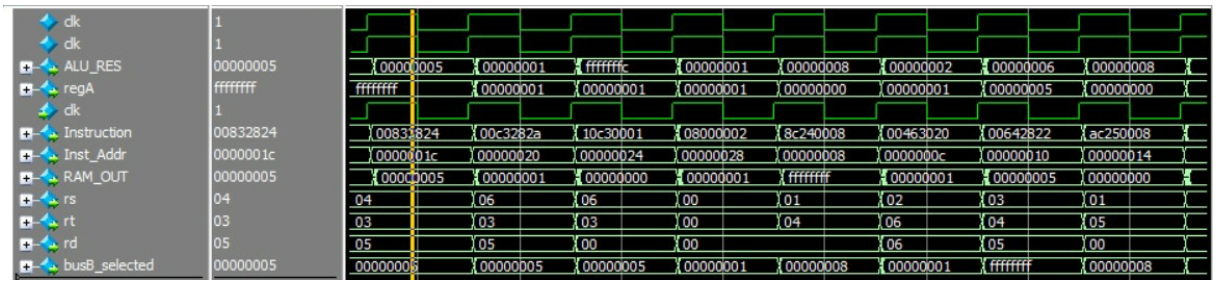
- 取指令模块：PC 寄存器的输出（Inst_Addr）
- 虚拟 ROM：指令内容（Instruction）
- 控制器：rs、rt、rd 等信号
- 寄存器堆：regA 等读取出的数据
- ALU 核心：运算结果（ALU_RES）
- 虚拟 RAM：内存读取结果（RAM_OUT）



2.2 程序执行逻辑分析

程序先从内存中读取数据 1、5、-1 到寄存器 2、3、4，然后开始循环。每次循环给寄存器 6 累加 1，直到寄存器 6 的值等于寄存器 3 的值（即 5），跳转到 exit 标签。程序执行的波形图符合预期：





三、硬件实验和 signalTap II 分析

为了便于观察，设置时钟周期为 1Hz。加入数码管扫描模块并设置输入输出端口后烧录进 FPGA，硬件测试电路如图（可以通过发光二极管观察时钟信号、ALUsrc、RegWr、RegDst 信号，通过数码管可以观察 ALU 计算结果的最后四位）。

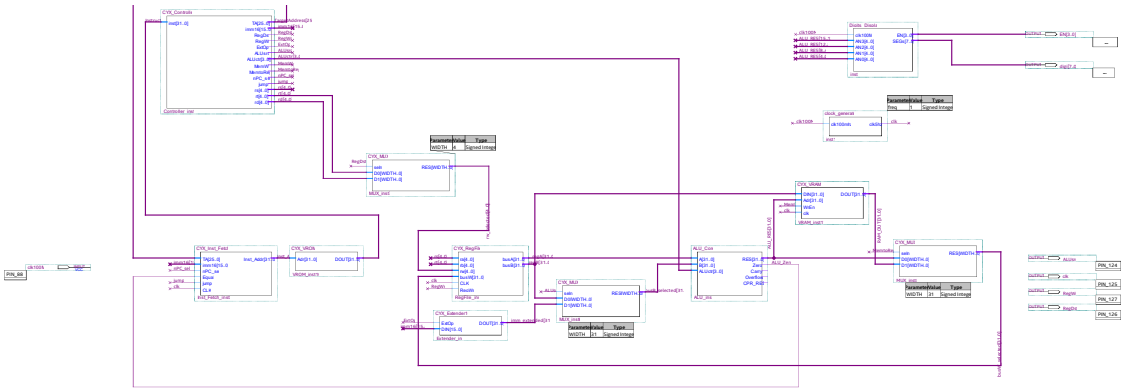


图 **1**　加入输入输出模块的硬件实验电路

经过观察发现，通过 LED 和数码管现实的各信号值的时间顺序符合预期。

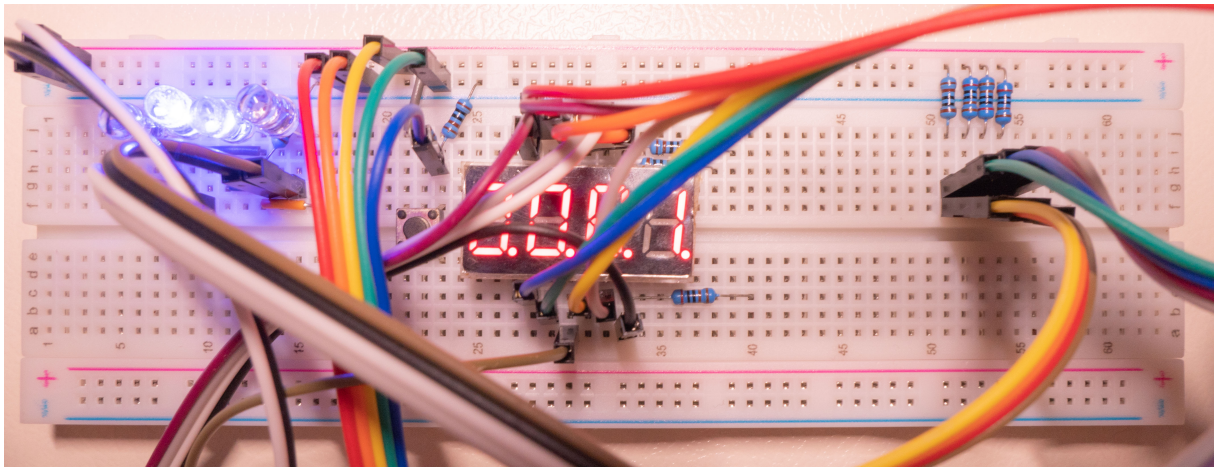
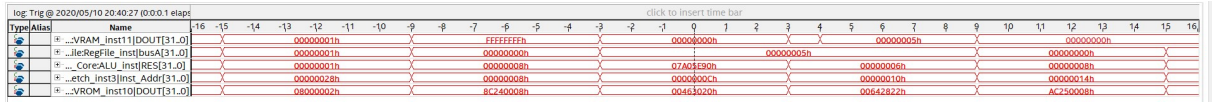
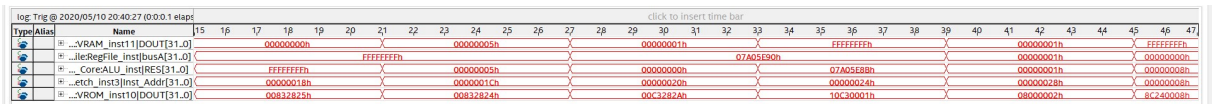


图 **2**　面包板上搭建的外围电路

设置时钟频率为 10MHz 后，通过 signalTap II 观察内部波形，结果如下，符合预期：





四、总结与思考

单周期 CPU 的设计过程中，存储器的时钟设计最让人困扰。之所以寄存器堆和主存需要在时钟沿存储数据，却不造成类似于流水线的冒险现象，是因为寄存器堆和主存（RAM）的写入操作总是在指令执行流程的最后。即便需要等待下一个时钟上升沿的到达才能写入数据，也并不影响下一个时钟周期的取指令等操作。

然而如果使用 Quartus 软件提供的 ROM 宏器件（该器件具有无法去掉的读取时钟）用于存储程序，情况便有所不同。如果指令地址准备好后需要等待下一个时钟周期才能读取指令内容，那么便难以控制单周期 CPU 数据通路的良好运行。此时可以给 ROM 使用更高频率的时钟、使用相同频率但上升沿延后的时钟或者使用时钟信号的反相，从而解决该问题。

在我的解决方案中，直接通过 Verilog 描述虚拟的 RAM 和 ROM，规避了上述时钟混乱的问题。