计算机体系结构实验课程第_1_次实报告

实验名称	多周期 CPU 实现			班级	李雨森老师
学生姓名	郭裕彬	学号	2114052	指导老师	董前琨
实验地点	A314		实验时间	2023.9.18、9.25	

1、 实验目的

1. 在单周期 CPU 实验完成的提前下，理解多周期的概念。
2. 熟悉并掌握多周期 CPU 的原理和设计。
3. 进一步提升运用 verilog 语言进行电路设计的能力。
4. 为后续实现流水线 cpu 的课程设计打下基础。

2、 实验内容说明

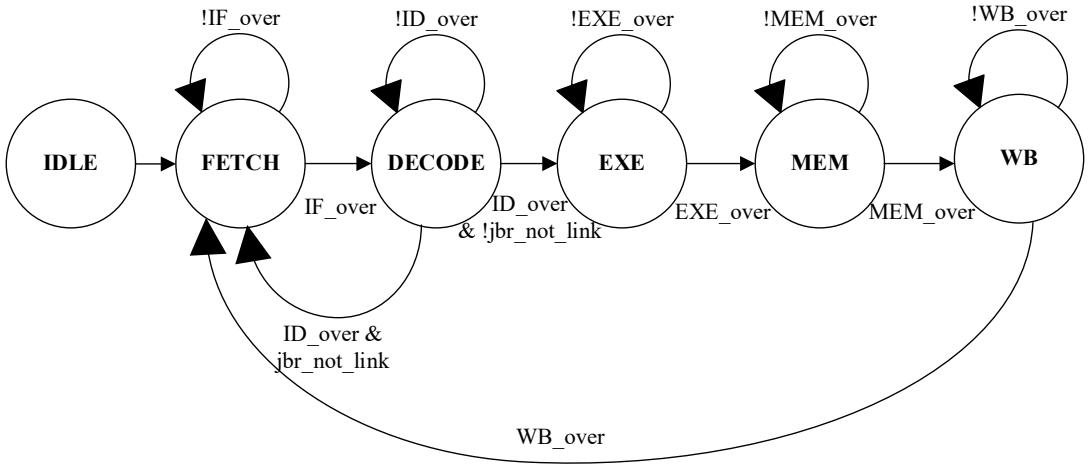
1. 单周期 CPU 实验采用的是异步存储器（data_ram.v 和 inst_rom.v）,多周期实验中因为又时钟控制周期，需要使用异步存储器（新建 IP 核的形式，参考存储器实验）。
2. 实验报告中要有实验箱验证照照片，并针对三种不同类型的 MIPS 指令，分析其对应的指令码、操作数和指令运行结果。

3 原始代码中存在一些隐藏 bug 和不合理的地方，在总结部分讨论这些问题。

3、 实验原理图

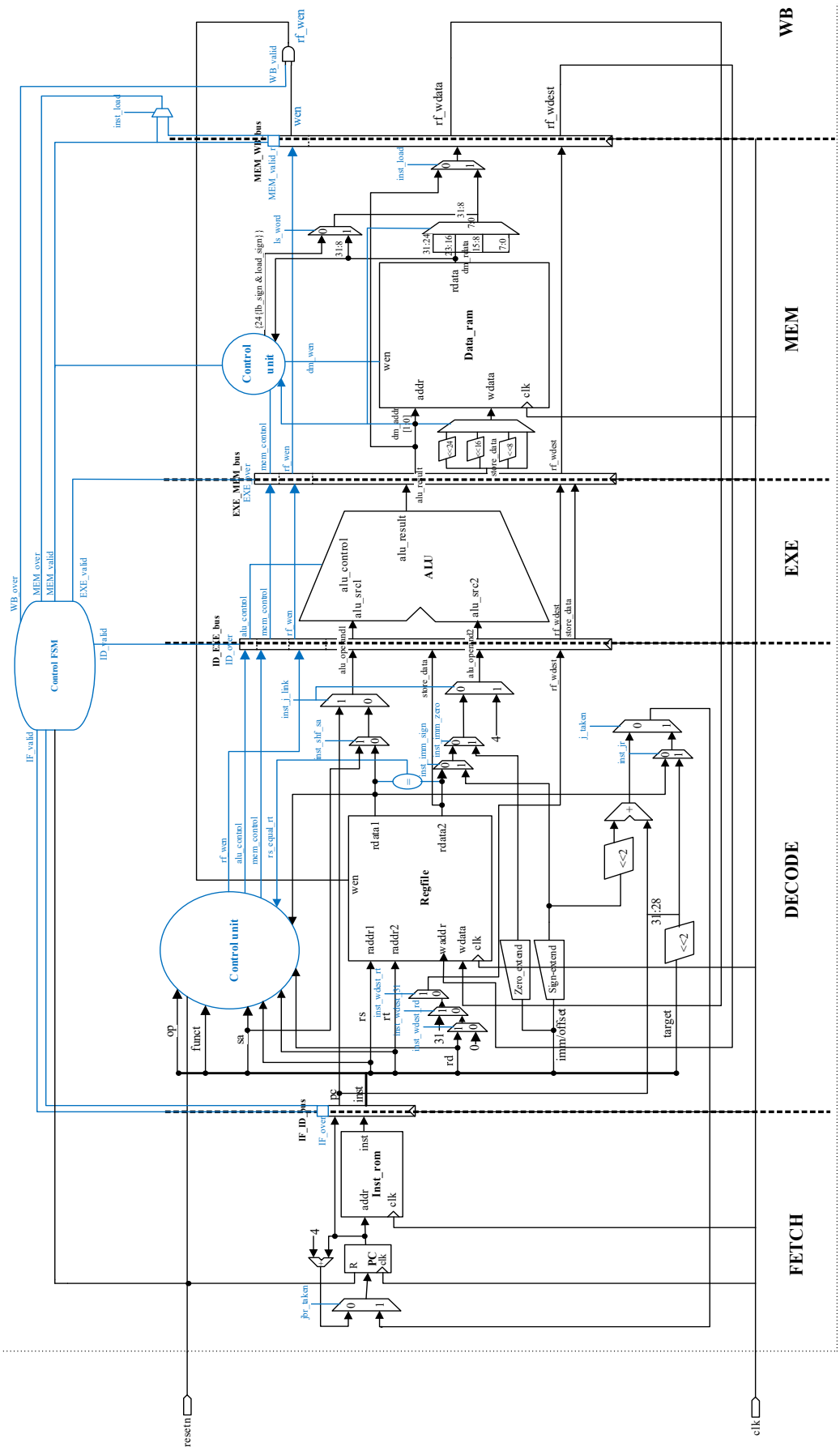
多周期 CPU 设计在单周期 CPU 基础上，主要做两部分改进。第一部分是控制单元，增加控制电路使每一个时钟只有一个阶段的电路产生的结果有效，并锁存上一阶段的结果用于后续阶段的运行；第二部分是数据通路，增加实现新增指令的电路。

第一部分的改进主要是增加状态机控制及增加各阶段之间的用于锁存的寄存器。由于有 5 个阶段，状态机共有 6 个状态：空闲（IDLE）、取指（FETCH）、译码（DECODE）、执行（EXE）、访存（MEM）、写回（WB），如下图：



CPU 复位结束，状态机由 IDLE 进入取指状态，其后在每次上一级结束信号有效的时进入下一状态，写回级结束后返回取指级。当然也有例外，当指令是跳转而非链接跳转指令时，在译码状态后直接返回取下一指令不需要经过执行等后续阶段。

多周期 CPU 的实现框图如下：


WB
MEM
EXE
DECODE
FETCH
rf_wen
WB_valid
wen
rf_wdata
rf_wdest
inst_load
MEM_WB_bus
MEM_valid_r
ls_word
{24{lb_sign & load_sign}}
Control unit
dm_wen
Data_ram
rdata
addr
wdata
clk
dm_addr[1:0]
store_data
mem_control
EXE_MEM_bus
alu_result
ALU
alu_control
alu_src1
alu_src2
ID_EXE_bus
inst_j_link
inst_shf_sa
inst_imm_sign
inst_imm_zero
rs_equal_rt
Regfile
rdata1
rdata2
raddr1
raddr2
waddr
inst_jr
j_taken
Zero_extend
Sign-extend
op
funct
sa
rs
rt
rd
imm/offset
target
inst_wdest_rt
inst_wdest_rd
IF_ID_bus
IF_over
inst
Inst_rom
PC
R
jbr_taken
Control FSM
WB_over
MEM_over
MEM_valid
EXE_valid
ID_valid
IF_valid
resetn
clk

4、 实验步骤

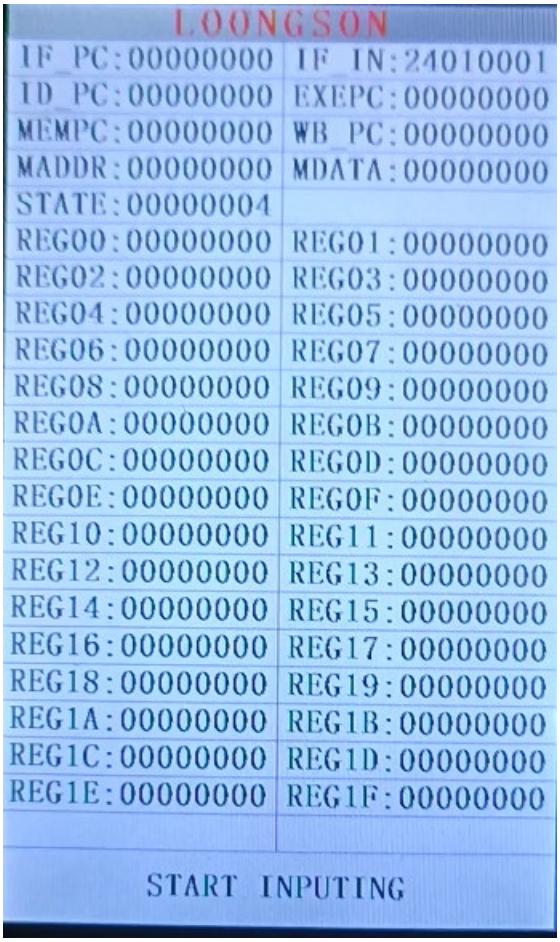
1. 同步 RAM 工程文件中根据实验文档建立双端口 RAM IP 核，同步 ROM 工程文件中根据实验文档建立单端口 ROM IP 核，调用 FPGA 板上的 IO 接口和触摸屏以及建立的 IP 核，进行引脚绑定，导入代码，引入转换成十六进制的测试代码。

2. 上箱验证，将工程文件烧录至开发板中，使用脉冲开关模拟时钟信号，观察各类型指令的执行情况。

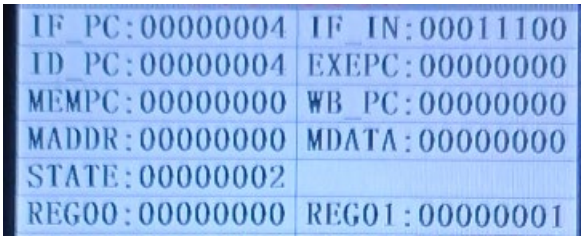
5、 实验结果分析

上箱验证，分别针对 R 型、I 型、J 型的 MIPS 指令，分析其对应的指令码、操作数和指令运行结果：

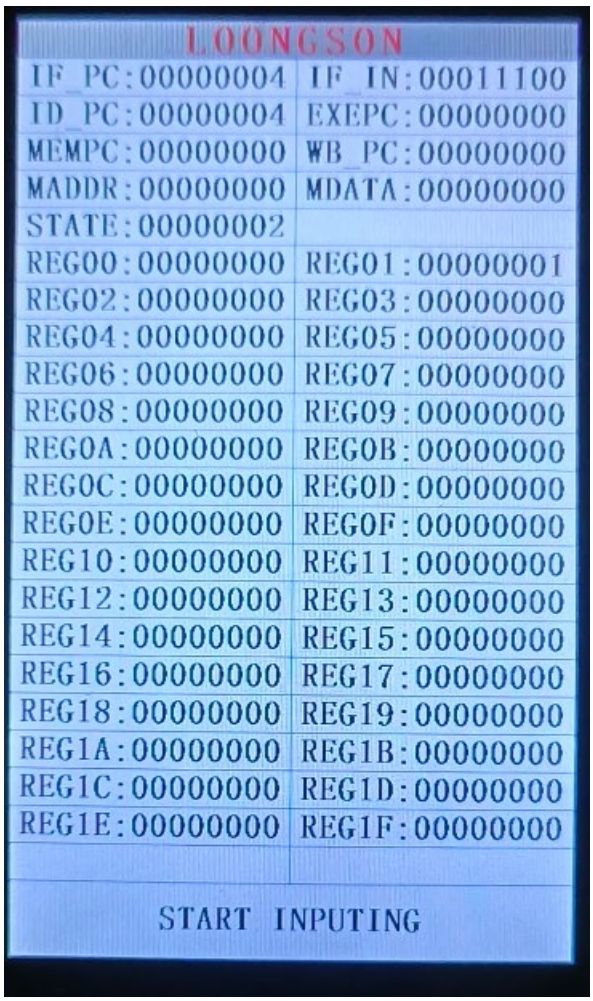
1. I 型指令



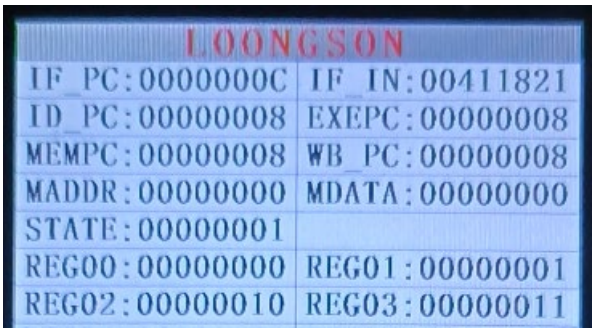
测试程序中，地址 0x0000_0000 存储的指令为 addiu $1, $0,#1，机器码为 24010001。Op 指令码字段为 6’b001001，表示 addiu 指令；rs 字段为 5’b00000，表示源寄存器为 0 号寄存器；rt 字段为 5’b00001，表示目的寄存器为 1 号寄存器；imm 立即数字段为 16’b0000_0000_0000_0000，指令的运行结果是，1 号寄存器的值从 0x0000_0000 被赋值为 0x0000_0001。正确结果如下：



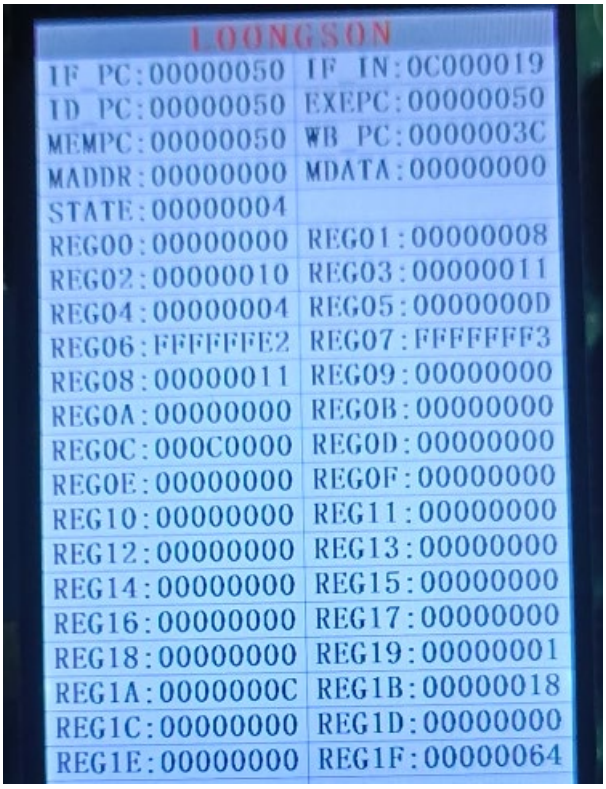
2. R 型指令



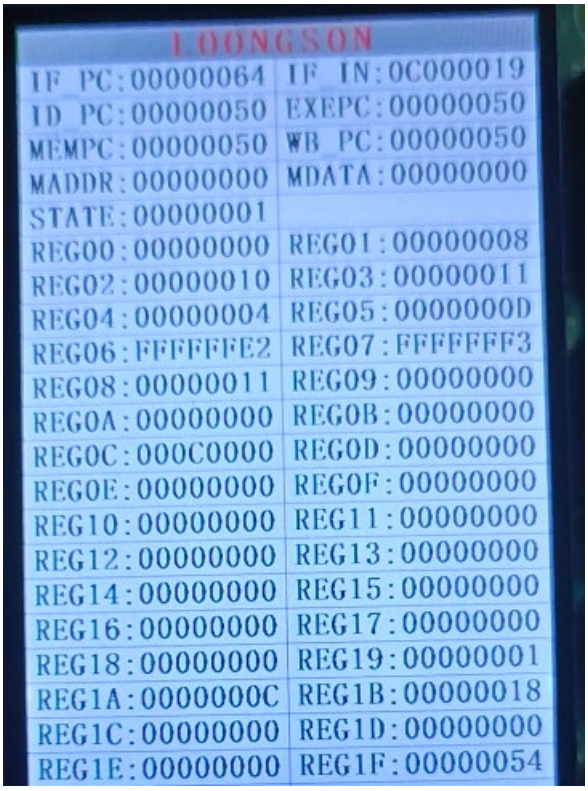
测试程序中，地址 0x0000_0004 存储的指令为 sll $2,$1,#4，机器码为 0x00011100。Op 指令码字段为 6’b000000，说明是 SPECIAL 类的 R 型指令，需要根据指令中 0~5bit 的值进一步判断；rs 字段为 5’b00000，无作用；rt 字段为 5’b00001，表示源寄存器为 1 号寄存器；rd 字段为 5’b00010，表示目的寄存器为 2 号寄存器；sa 字段为 5’b00100，表示相对偏移量是 4，0~5bit 功能码的值是 6’b000000，表明这条指令是 sll 指令。指令运行结果是，将 1 号寄存器的值 0x0000_0001 左移 4 位，空出来的位置用 0 填充，结果保存到 2 号寄存器中，值为 0x0000_0010。正确结果如下：



3. J 型指令



测试程序中，地址 0x0000_0050 存储的指令为 jal #25，机器码为 0x0C000019。Op 指令码字段为 6’b000011，表示 jal 指令，target 字段为 26’b00_0000_0000_0000_0000_0001_1001。指令运行结果是这条跳转指令后面的指令的地址 0x0000_0054 作为返回地址保存到寄存器$31 中，再转移到新的地址，其中低 28 位是 target 字段左移两位的值，高 4 位是当前 PC 的高 4 位，组成的跳转地址是 0x0000_0064，正确结果如下：



6、 总结感想

观察到指令在运行到位于 0x14 的指令 bgez $25,16 时，本应在 ID 译码阶段结束后即回到 IF 阶段取跳转后的指令，但实际情况为继续依次经过 EX、MEM、WB 阶段后，连续两次偏移 16*4=40H 跳转到 0X54、0X94。

查看代码，发现在 multi_cycle_cpu.v 中控制 ID 跳转 IF 阶段的信号 jbr_not_link 在 decode.v 中需要两次的赋值，译码跳转指令在一个时钟周期后并不能检测到正确的 jbr_not_link 信号从而跳转。

已经实现的改进方式是，在 decode 译码阶段手动添加一个时钟周期的延时，从而保证在下一个周期就要使用的 jbr_not_link 信号的正确输出：

```
//assign ID_over = ID_valid 改为:
   reg ID_valid_r;
  always @(posedge clk)
   begin
      ID_valid_r <= ID_valid;
   end
   assign ID_over = ID_valid_r;
```

经验证，测试程序的所有指令都能够正常执行。

还有两种尚未验证可行性的思路，一是对译码阶段分支和跳转语句要用到的、被多重 assign 赋值语句关联的信号进行语句的压缩，使其能在一个周期内达到准确值；二是将判断是否重新回到 IF 阶段的过程移动到 MEM 阶段，也就是一个周期后，需要将一些信号作为新的部分添加进总线中，但仅修改这些会导致跳转链接的语句出现异常，受限于时间没有实现进一步的完善。