Lab6：RISC-V 单周期处理器的设计与实现

1. 实验简介

在本实验室中，将使用 Verilog 编写 32 位的 RISC-V 单周期处理器 singleriscv_fpga，指令集采用 32 位基本集 RV32I 的子集。本实验将设计一个 32 位的 ALU 与处理器其余部分的代码结合起来。然后将加载一个测试程序，并检查指令是否工作。接下来，将实现 3 条新指令，然后编写一个新的测试程序来确认新指令也可以工作。接下来，编写一个 BCD 转二进制模块，并将以往试验中的二进制转 BCD 模块和 LED 显示等主要部件重新组合形成一个简单计算器。接下来，编写汇编程序实现加、减、乘等计算器功能，I/O 操作采用程序查询模式。在本实验结束时，应该彻底了解 RISC-V 单周期处理器和简单 I/O 外设的内部操作，了解汇编语言编程以及完整的硬件系统设计流程。

在开始本实验之前，应该非常熟悉课本和讲义中描述的 RISC-V 处理器的单周期实现。为了方便起见，图 1 中重复了课本中的单周期处理器示意图。该版本的 RISC-V 单周期处理器可以执行以下指令：add, sub, and, or, slt, lw, sw, beq, addi, j。为了实现简单计算器，还添加了 lui，slli，和 srli (shamt = 1)等三条指令。

本实验有三个主要目标：

1）采用布尔公式描述 32 位全加器和超前进位链，实现减法功能，并形成 ALU 设计，最终形成完整的单周期 RISCV 处理器。

2）增加 3 条指令，andi、ori、bne，并通过测试

3）基于单周期处理器完成简单计算器设计，可实现十进制加、减、乘的功能，编写计算器汇编程序并测试通过。

本实验的 RISC-V 单周期处理器有两个模块组成：控制器和数据通路。每个模块都有若干功能模块组成。例如，数据通路包括 ALU、寄存器堆、符号扩展逻辑和用来选择操作数的多组多选器组成。

本实验的地址映射和寄存器地址如下，这些定义和 RISC-V 汇编模拟器 RARS 模拟器保持一致，只在 16 位空间中进行操作。

地址映射：

0x0000-0x0FFF：指令空间

0x1000-0x17FF：数据空间

0x7F00-0x7FFF：I/O 空间

其他：保留

I/O 寄存器地址：

PORTA：0x7F00，只读，寄存器低 4 位保存 4 个按钮 btnL, btnC, btnR, btnU 通过同步器后的输入值。

PORTB：0x7F10，只读，寄存器中保存 16 个开关 sw[15:0]经过 BCD2Bin 转换后的 16 位二进制。

PORTC：0x7F20，读写，寄存器中保存 16 位二进制，后经过 Bin2BCD 的值通过 LED 模块显示。

PORTD：0x7FFC，读写，寄存器中保存值的低 16 位二进制输出 led[15:0]，驱动 16 个发光二极管。

Verilog 模拟虚拟停机寄存器的 I/O 地址

PORTD/HALT：0x7FFC，读写，当数据地址是 0x7FFC，同时写数据为 0x1 时，Verilog 模拟停止。

2. 目标计算器的规范说明

2.1 功能描述：用按钮选择适当的计算并将结果输出到 4 个 7 段数码管上（复用 Lab1）。计算：实现 2 位十进制的加、减、乘的计算功能，要求用 verilog 编写布尔表达式实现加法器，并在此基础上实现减法功能。用 RISC-V 汇编程序实现乘法运算和所有结果的输出。

2.2 按钮输入的功能描述：

BntR：乘法键，将累加器 x5 的数值乘以输入 SW 经 BCD 转换后的 PORTB 数值，结果写回累加器，并将结果输出到 PORTC，以 BCD 形式输出到 7 段数码显示中。

BntC：减法键，将累加器 x5 的数值减去输入 SW 经 BCD 转换后的 PORTB 数值，结果写回累加器，并将结果输出到 PORTC。

BntL：加法键，将累加器 x5 的数值加上输入 SW 经 BCD 转换后的 PORTB 数值，结果写回累加器，并将结果输出到 PORTC。

BntU：等号键，将计算结果输出到 PORTC。

BntD：复位键，单周期处理器从 0x0000 开始取指执行。

2.3 其他输入输出功能描述：

采用 4 位十进制 BCD 输入和 4 位十进制输出。

采用 16 个开关，SW[15:0]，每四个开关分别为十进制 BCD 的千、百、十个位输入，通过 BCD2Bin 转换电路转换成 16 位二进制，进行加法、减法和乘法运算，计算不考虑溢出。乘法运算时，应该输入只输入小于 99 的 BCD 码。

计算结果为 16 位二进制，通过 Bin2BCD 的转换电路转换成 4 位十进制 BCD 输出。

3. RISC-V 汇编模拟器 RARS

RARS 模拟器是在 MARS MIPS 模拟器的基础上开发的支持 RISC-V 指令系统的汇编级模拟器，采用 java 开发。下载网址：

https://github.com/TheThirdOne/rars/releases

3.1 下载并在 Windows 环境中直接运行 rars.jar，需要安装 java8 JRE 或 JDK。

打开 testriscv.asm 汇编程序，或者重编写新的汇编程序。

3.2 RARS 模拟器的 Run->Assemble 进行汇编过程，然后可以运行或逐行调试，可观察寄存器和内存变化。内存窗口下方可选择不同段的内存显示。

3.3 RARS 模拟器的 Settings->Memory Configuration…中选择 Compact, Text at address 0，此时程序段.text 段默认为 0x0000 开始，数据段.data 默认为 0x1000 地址开始，内存映射输入输出 MMIO 段默认为 0x7F00~0x7FFF。

3.4 调试成功后，RARS 模拟器的 File->Dump Memory…可以导出 Verilog 语言$readmemh() 函数可以识别的 16 进制文本指令文件，Dump Format 选择 Hexadecimal text。

4. signleriscv_fpga 单周期处理器

实验给出了 Verilog 单周期 RISC-V 的基本模块。将 Lab6 文件夹复制到你自己的目录中。修改 xx 为指定数字后，在 Vivado 中建立工程文件，顶层模块是 singleriscv_fpga.v。

观察 riscv 模块，它实例化了两个子模块 controller 和 datapath。然后观察控制器模块及其子模块。它包含两个子模块 maindec 和 aludec。除 ALU 外，maindec 模块产生所有控制信号。aludec 模块为 ALU 产生控制信号 alucontrol[2:0]。确保你完全理解控制器模块。

在完全理解控制器模块之后，观察 datapath 的 Verilog 模块。数据通路有相当多的子模块。确保您理解了为什么设计并例化每个子模块，以及每个子模块位于 RISC-V 单周期处理器示意图上的位置。你将观察到 alu 模块在 Xilinx 的 Sources 窗口中有一个问号。您需要重新编写这个 alu 的 Verilog 模块（riscvalu.v）。确保模块名称与实例模块名称(alu)匹配，并确保输入和输出的顺序与数据路径模块中预期的顺序相同。

要求：该 alu 模块可以在课本或者讲义中找到，要求采用布尔表达式的方式实现 32 位加法器，实现超前进位链，并在此基础上实现减法 sub 和比较 slt 指令的功能。

顶层模块 top 包括指令存储器、数据存储器（包含 I/O 部分）和单周期处理器。数据内存是一个 16 字×32 位数组。指令内存需要包含程序的初始值。用初值创建内存最方便的方法是使用 Xilinx 的 Core Generator。

5. 测试程序

本实验将使用下面的简单程序测试基本的功能：

# Instruction		Address	Code	
.text				
main:	addi x2, x0, 5	#0x0000	0x00500113	addi x2,x0,0x00000005
	addi x3, x0, 12	#0x0004	0x00c00193	addi x3,x0,0x0000000c
	addi x7, x3, -9	#0x0008	0xff718393	addi x7,x3,0xfffffff7
	or x4, x7, x2	#0x000c	0x0023e233	or x4,x7,x2
	and x5, x3, x4	#0x0010	0x0041f2b3	and x5,x3,x4

```
        add x5, x5, x4          #0x0014  0x004282b3   add x5,x5,x4
        beq x5, x7, end         #0x0018  0x02728a63   beq x5,x7,0x0000001a
        slt x4, x3, x4          #0x001c  0x0041a233   slt x4,x3,x4
        beq x4, x0, around      #0x0020  0x00020463   beq x4,x0,0x00000004
        addi x5, x0, 0          #0x0024  0x00000293   addi x5,x0,0x00000000
around: slt x4, x7, x2          #0x0028  0x0023a233   slt x4,x7,x2
        add x7, x4, x5          #0x002c  0x005203b3   add x7,x4,x5
        sub x7, x7, x2          #0x0030  0x402383b3   sub x7,x7,x2
        lui x3, 1               #0x0034  0x000011b7   lui x3,0x00000001
        sw x7, 12(x3)           #0x0038  0x0071a623   sw x7,0x0000000c(x3)
        addi x3, x3, 8          #0x003c  0x00818193   addi x3,x3,0x00000008
        lw x2, 4(x3)            #0x0040  0x0041a103   lw x2,0x00000004(x3)
        j end                   #0x0044  0x0080006f   jal x0,0x00000004
        addi x2, x0, 1          #0x0048  0x00100113   addi x2,x0,0x00000001
end:    lui x9, 8               #0x004c  0x000084b7   lui x9,0x00000008
        addi x9, x9, -4         #0x0050  0xffc48493   addi x9,x9,0xfffffffc
        addi x2, x2, -6         #0x0054  0xffa10113   addi x2,x2,0xfffffffa
loop:   sw x2, 0(x9)            #0x0058  0x0024a023   sw x2,0x00000000(x9)
        or x2, x2, x2           #0x005c  0x00216133   or x2,x2,x2
        and x2, x2, x2          #0x0060  0x00217133   and x2,x2,x2
        j loop                  #0x0064  0xff5ff06f   jal x0,0xfffffffa
```

Figure 1. RISC-V 汇编程序 testriscv.asm

这个代码可以在实验材料里面找到，可以用 RARS 汇编模拟器模拟。用 RARS 模拟器导出指令段的 16 进制文本文件 testriscv.dat，复制到 lab6 子目录中，用于 Verilog 模拟。可见于 singleriscv_tb.v 的 imem.v 中的$readmemh()函数。

选择 singleriscv_tb.v 并进行模拟（iverilog 模拟），如果模拟正确，会在 Console 窗口中显示 simulate success 字样，在 0x7FFC 地址上写入了一个 1'b1。

6. 生成指令 ROM 模块 imem_fpga

在 Vivado 中，这其中 singleriscv_fpga 和 imem_fpga 两个新的 module 名字。将 singlersicv_fpga 设为顶层文件，imem_fpga 模块需要采用 Vivado IP 生成器生成。

为了将上面这段代码综合到 FPGA 中，需要将其装载到 singleriscv_fpga 的指令 ROM 中。目前 Xilinx 软件还不能直接将其转换成 ROM。因此需要用 Xilinx Distributed Memory Generator 工具生成特定的可综合的 ROM 模块。

注意：每次更改 ROM 值，都需要重新运行 Distributed Memory Generator，生成新的 ROM 模块。

需要采用手动的方式将 testriscv.dat 改写成如下的 testriscv.coe 文件。建议安装 vim 编辑软件。加入前面 4 行后，利用 vim 软件的块操作功能，将光标移到第一行指令的最后，Ctrl+v 进

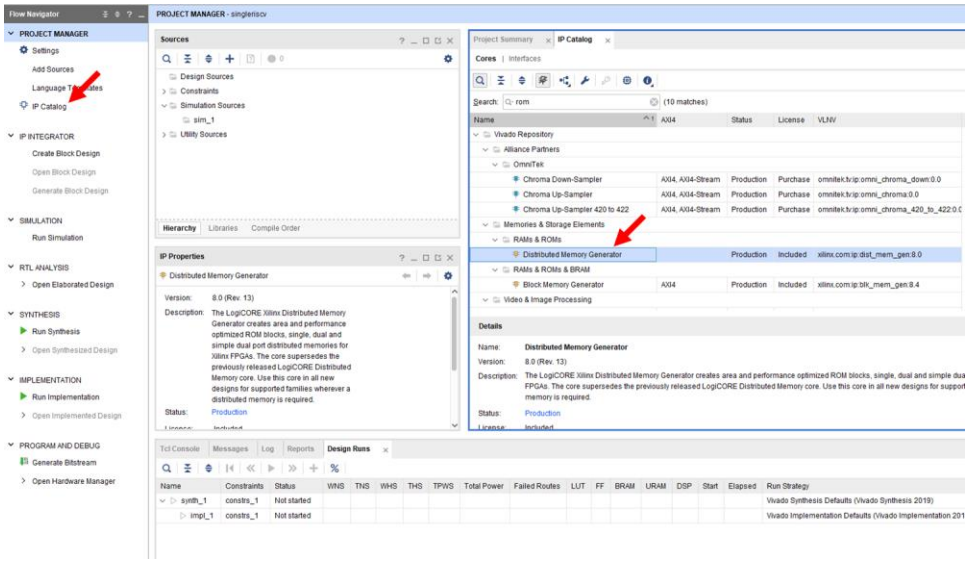
入块编辑模式，利用键盘上下左右键选中最后一列，按 Shift+a，输入“,”逗号，按 Esc 键退出，选中的所有列后面就加入了逗号。然后将最后一行的逗号改成“;”分号，保存退出即可。形成的文件如下：

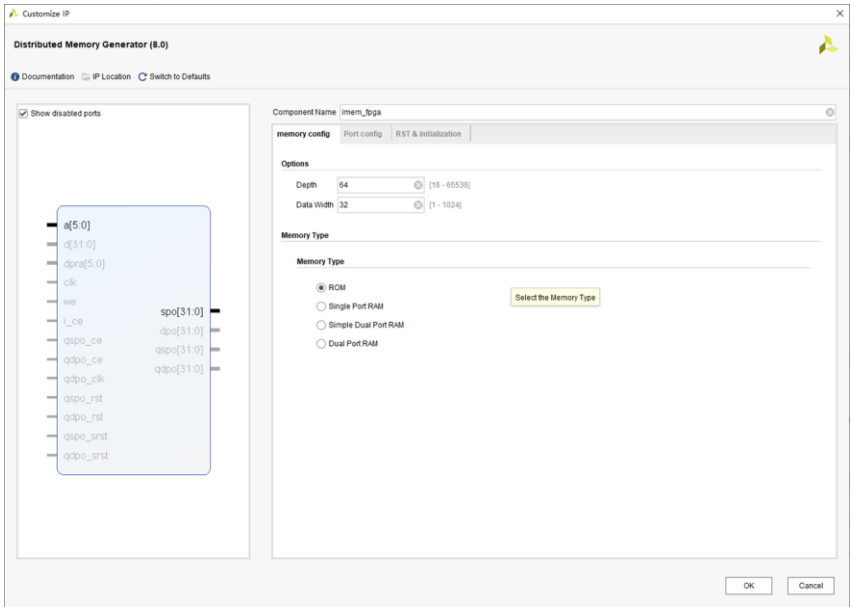
```
; Sample initialization file for a
; 32-bit wide by 256 deep ROM
memory_initialization_radix = 16;
memory_initialization_vector =
00500113,
00c00193,
ff718393,
0023e233,
0041f2b3,
004282b3,
02728a63,
0041a233,
00020463,
00000293,
0023a233,
005203b3,
402383b3,
000011b7,
0071a623,
00818193,
0041a103,
0080006f,
00100113,
000084b7,
ffc48493,
ffa10113,
0024a023,
00216133,
00217133,
ff5ff06f;
```

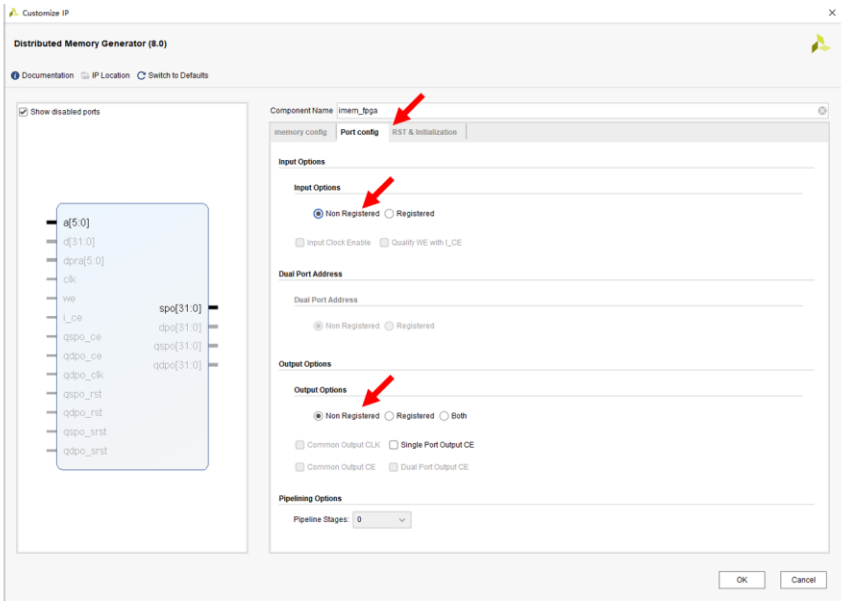
Figure 2. RISC-V 汇编程序 testriscv.asm 的 testriscv.coe 文件

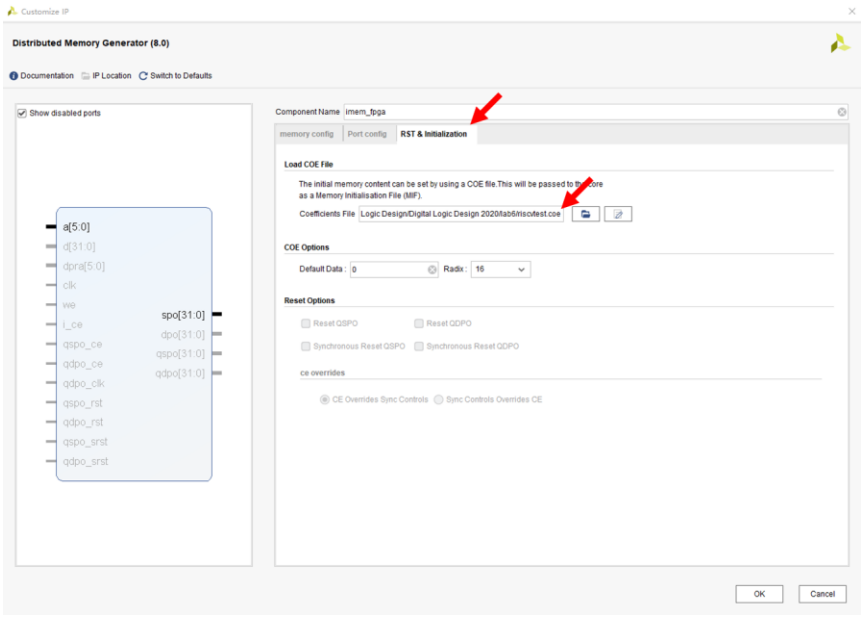
指令存储器 imem_fpga, 用 Xilinx 的 Distributed Memory Generator 构造 ROM，它将包含需要执行的指令。生成 ROM，模块名“imem_fpga”。生成流程如下图所示。需要你输入模组名字(“imem_fpga”)、深度 depth (256)、宽度 width (32) 等等。 选择 ROM 类型并确保输入输出都是非寄存器(unregistered)。 在左侧会显示符号图 8-bit address 和 SPO 32-bit data 输出. 注意只有 PC 的部分比特位 ($PC_{9:2}$)会用来作为指令地址（要理解为什么只用了这些比特位）。在 ROM

文件名窗口中，输入或选择指定的.coe 文件名。当完成时，点击 “Generate” 生成 ROM 模块，相应的符号名会出现在工程窗口中。









7. singleriscv_fpga 的综合

综合最上层模块 singleriscv_fpga。注意这个模块和用于模拟的 signleriscv_tb.v 模块的区别，去掉了一些不必要的用于模拟的输出输出信号。

查看综合报告，确保没有错误发生，然后下载到 fpga 板上模拟。当按下 bntU 后，发光二极管的最低位 ld[0]亮，其他不亮，模拟正确。

8. 修改 singleriscv_fpga 单周期处理器

下一步，在 singleriscv_fpga 单周期处理器中增加 3 条 RISC-V 指令 andi、ori、bne，修改主译码器 main decoder 和 ALU decoder 以及其他模块。

9. 设计简单计算器功能

在 singleriscv_fpga 中增加 BCD2Bin、Bin2BCD 和 Display7seg 模块。

10. singleriscv_fpga 单周期处理器的指令添加和计算器汇编的编写调试

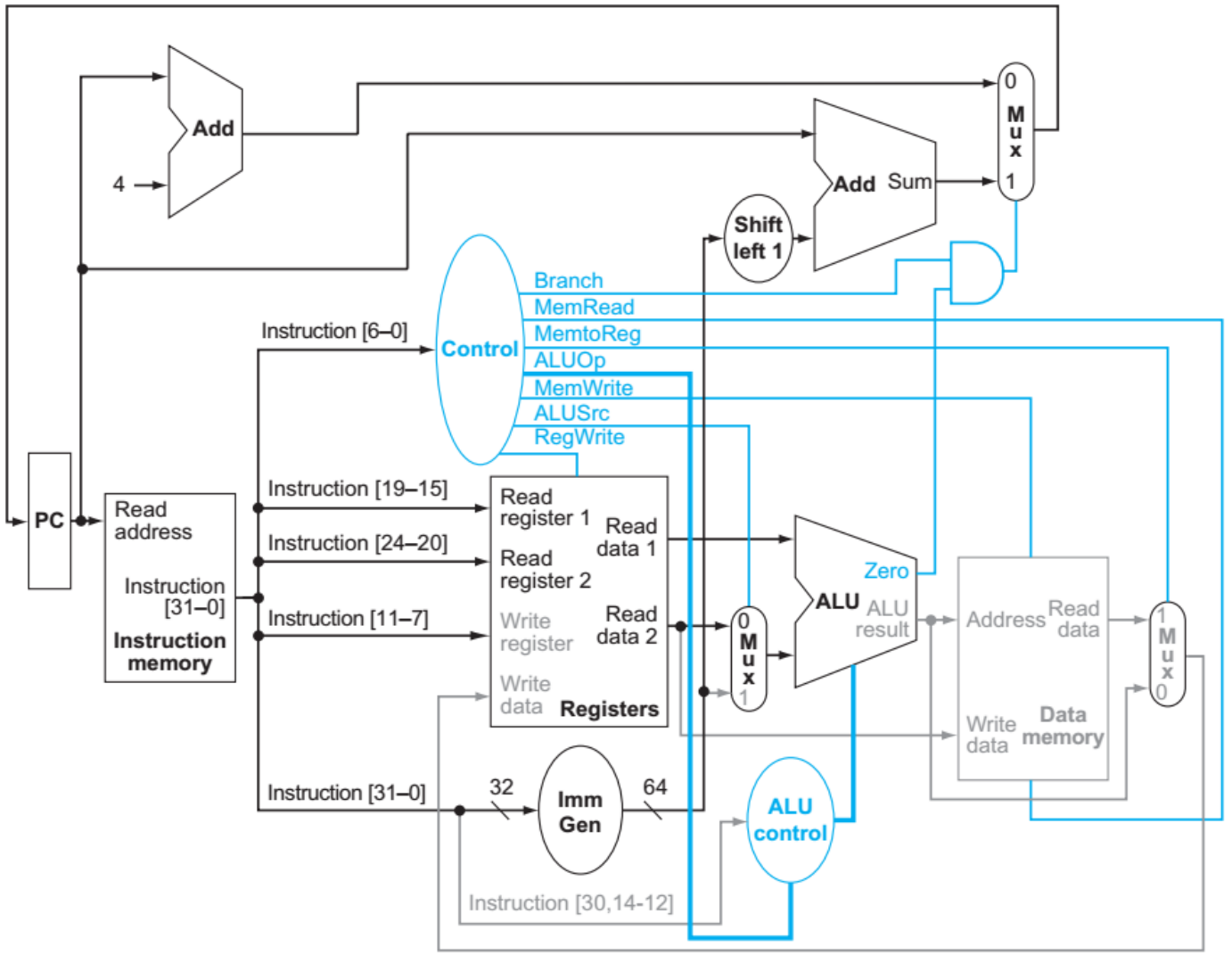
在以上实验完成的基础上，增加三条指令 andi，ori，bne 指令，并测试。

并编写计算器的汇编程序，采用查询方法检测输入的变化，参考过程如下：

- 复位从 0x0000 开始执行指令
- 将作为累加器的寄存器 x5 和作为命令记录的寄存器 x6 写成 0
- 循环检测 porta 和 portb 的值是否改变，不改变循环检测。
- 如果 portb 的值改变了，将该值写入临时寄存器 x7 中，并写入输出寄存器 portc 中显示。
- 如果 porta 的值改变了，将该输入值写入临时寄存器 x8 中，并等待 porta 的值变为 0，否则循环等待。
- 如果 porta 的值变为 0，说明按钮已经抬起，判断 x6 寄存器是否为 0，为 0 将 x8 中的值写入 x6，将 x7 写入 x5，进入等待另一个操作数的阶段。
- 循环计算开始地址：循环判断 portb 是否改变，如果 portb 的值改变了，将该值写入临时寄存器 x7 中，并写入输出寄存器 portc 中显示
- 当检测再次 porta 的值再次变化，将该值写入 x8 中，循环等待变回 0 后，判断 x6 中的值进行转移到相应计算，将累加器 x5 的值和 x7 中的值进行操作，结果写入累加器 x5 中，并将结果写入 portc 显示。
- 回到循环计算开始地址。

11. 简单计算器模拟

无论采用 iverilog 模拟还是 vivado 模拟，自行在 singleriscv_tb.v 中增加 BCD 转换模块和 display7seg 模拟，自行编写输入向量模拟，验证功能的正确性。



RISCVfpga 单周期处理器数据通路示意图

主译码器扩展功能

Instruction	**$Op_{5:0}$**	**RegWrite**	**RegDst**	**$AluSrc_{1:0}$**	**Branch**	**MemWrite**	**MemtoReg**	**$ALUOp_{1:0}$**	**Jump**	**LuitoReg**		
R-type		1	1	00	0	0	0	10	0			
lw		1	0	01	0	0	1	00	0			
sw		0	X	01	0	1	X	00	0			
beq		0	X	00	1	0	X	01	0			
addi		1	0	01	0	0	0	00	0			
j		0	X	XX	X	0	X	XX	1			
ori												
bne												
andi												
liu												
slli												
srli												

ALU 译码器扩展功能

$ALUOp_{1:0}$	**Meaning**
00	Add
01	Sub
10	R-TYP 看 funct3 域
11	I-TYP 看 funct3 域