中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：在 FPGAOL 平台上实现 myISA

学生姓名：黄瑞轩

学生学号：PB20111686

完成日期：2021.12.14

计算机实验教学中心制

2020 年 09 月

实验题目

在 FPGAOL 平台上实现 myISA

实验目的

- 学习有限状态自动机的编写
- 学习 FPGAOL 平台上串口的使用

实验环境

- FPGAOL 实验平台：fpgaol.ustc.edu.cn
- Vivado 工具

实验设计

【设计概述】

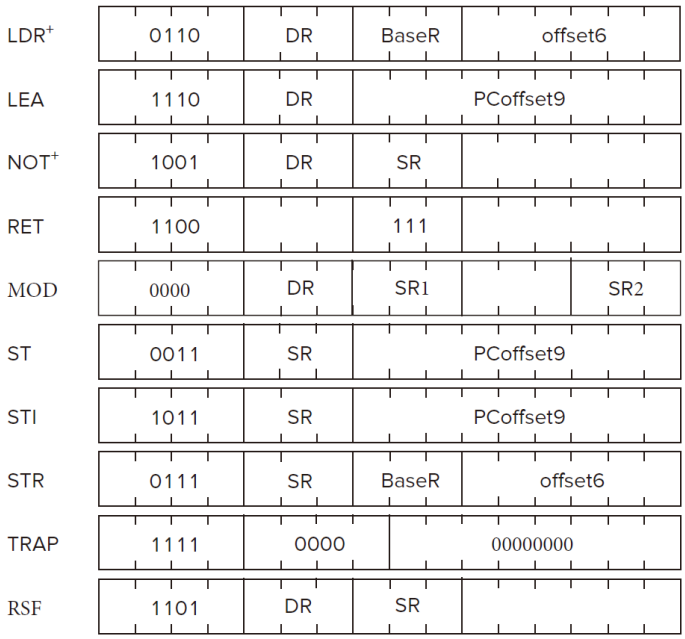
在这个实验中，我将依托 fpgaol 在线平台，以有限状态自动机的方式实现一个支持编程的 myISA 指令集。用户可以通过查阅 myISA 指令表来操作内存和临时变量寄存器的值来进行编程。

myISA 给出了内存组织方式、寄存器组、指令集（包括操作码、数据类型、寻址模式）等信息。myISA 的可寻址空间大小是 64，寻址基本单位是 16 位。myISA 和大多数的机器一样，提供了临时存储空间（R[7:0]）。

myISA 中的一条指令分为两个部分：操作码（做什么）和操作数（对谁操作）。所有指令可以分为三类：运算（operate）、数据搬移（data movement）和控制（control）。运算类指令负责处理信息；数据搬移类指令则负责在内存和寄存器之间以及内存/寄存器和 IO 设备之间转移信息；控制类指令负责改变指令执行的顺序，即它们能让程序随时跳转至另一个地方继续执行（而不是常规的顺序向下执行）。

myISA 指令表

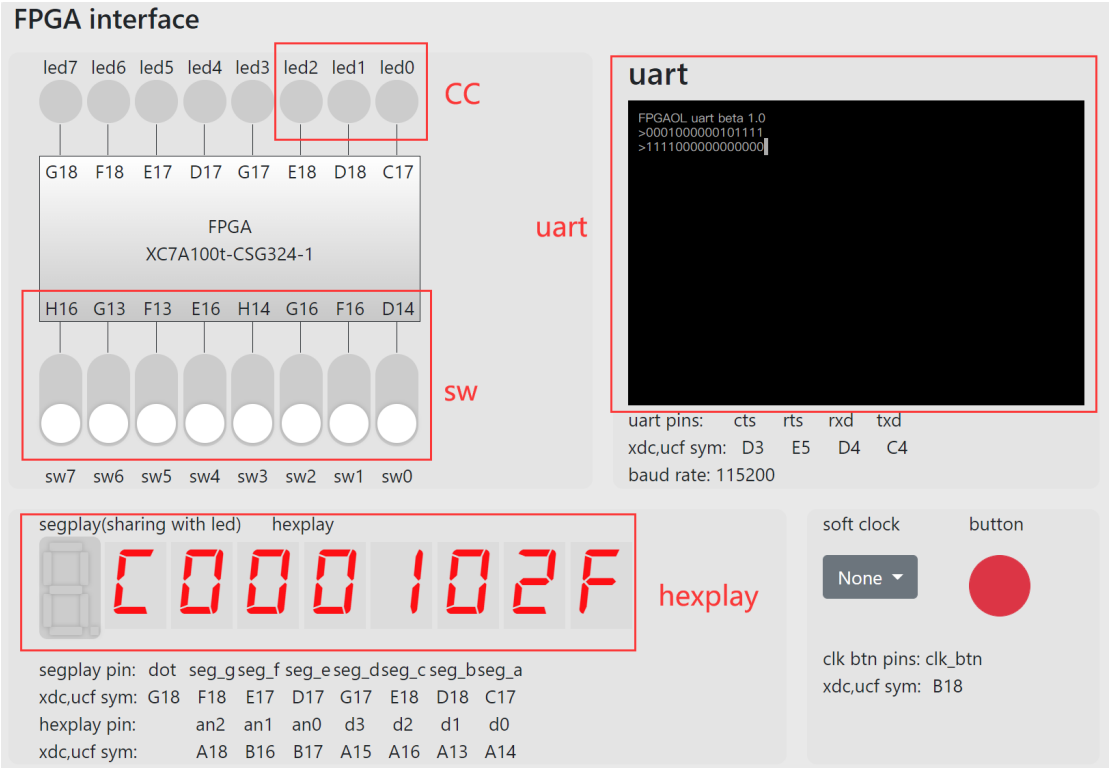
	15 14 13 12	11	10 9	8 7 6	5	4 3	2 1 0
ADD+	0001	DR		SR1	0		SR2
ADD+	0001	DR		SR1	1	imm5	
AND+	0101	DR		SR1	0		SR2
AND+	0101	DR		SR1	1	imm5	
BR	1000	n	z p	PCoffset9			
JMP	1100			BaseR			
JSR	0100	1	PCoffset11				
JSRR	0100	0		BaseR			
LD+	0010	DR		PCoffset9			
LDI+	1010	DR		PCoffset9			



每条指令的用途和示例

INST_ADD	0001 001 001 0 XX 001 0001 001 001 1 11111	将 R1 与 R1 相加，结果放在 R1 中 将 R1 与立即数-1 相加，结果放在 R1 中
INST_AND	与 INST_ADD 类似，功能是相与	
INST_BR	1000 010 000000011	检测条件码是否为 CC = 3'b010，如果是，则 PC <= PC + 3
INST_JMP	1100 XXX 101 XXXXXX	令 PC <= R[5]
INST_JSR	0100 1 00000000001 0100 0 XX 101 XXXXXX	把当前的 PC 存到 R7，同时令 PC <= PC + 1 把当前的 PC 存到 R7，同时令 PC <= R[5]
INST_LD	0010 011 000000011	把 mem[PC + 3]的值存到 R[3]里
INST_LDI	1010 011 000000011	把 mem[mem[PC + 3]]的值存到 R[3]里
INST_LDR	0110 011 010 000011	把 mem[R[2] + 3]的值存到 R[3]里
INST_LEA	1110 111 111111111	把 PC – 1 的值存到 R[7]里
INST_NOT	1001 111 000 XXXXXX	把~R[0]存到 R[7]里
INST_MOD	0000 111 110 XXX 101	把 R[6]模 R[5]的值存到 R[7]里
INST_ST	0011 111 111111111	把 R[7]的值存到 mem[PC - 1]处
INST_STI	1011 111 111111111	把 R[7]的值存到 mem[mem[PC - 1]]处
INST_STR	0111 111 110 111111	把 R[7]的值存到 mem[R[6] - 1]处
INST_HALT	1111 000000000000 1111 111111111111	程序运行到此结束 机器开始从 mem[0]执行所有指令
INST_RSF	1101 111 110 XXXXXX	把 R[6]右移一位的值存到 R[7]

FPGAOL 各功能区使用说明



CC：条件码显示区，显示当前条件码（高电平有效）。

hexplay：当 sw[7:0]全为 0 时，显示上一条输入进内存的指令（以低 4 位 16 进制呈现）；当 sw[7:0]有 1 时，显示为 1 的最低位为编号的寄存器内容。

sw：开关，控制 hexplay 的显示内容。

uart：串口输入，输入指令。

【详细设计】

本实验设计的状态机有如下 4 个状态：

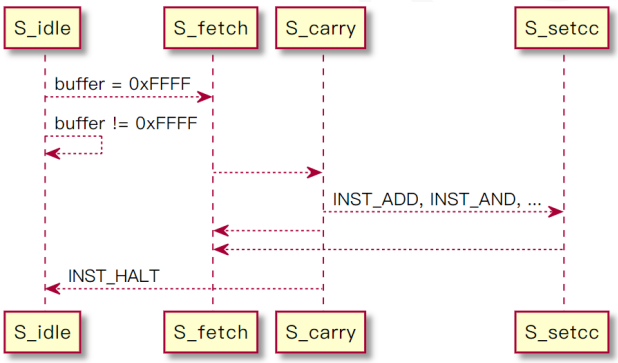
```
S_idle  = 2'd0;      静止状态
S_fetch = 2'd1;      取指令状态
S_carry = 2'd2;      执行指令状态
S_setcc = 2'd3;      设置条件码状态
```

用户通过串口在静止状态输入指令，指令格式如下：

```
01010...01010101      输入一个 16bit 指令，最后一行指令必须是 HALT
1111111111111111      表示包括 HALT 在内的所有指令都输入完毕，开始执行
```

设置一个串口缓冲区寄存器 buffer[16:0]和指针 buffer_ptr 指向下一个待写入的 buffer 位置，buffer_ptr 从 buffer 高位写到低位，即初始状态是 buffer_ptr = 15。每当 rx_data 不是换行符，就向 buffer 中写，否则初始化 buffer_ptr = 15。当读到换行符时，说明这一行指令已经被完全读到 buffer 中，然后需要判断：如果 buffer 的内容是 0xFFFF，表示用户输入结束了，不应该写入内存中；否则需要将 buffer 内容存入 mem[PC]，然后使得 PC 自增。

状态机的第一部分是现态（cs）和次态（ns）的转换部分，遵从如下状态图：



简单解释一下：S_idle 状态下，需要对 buffer 进行判断以切换到 S_fetch 状态，否则保持 S_idle 状态；S_fetch 状态下，无条件切换到 S_carry 状态；S_carry 状态需要判断 IR 中操作码以确定是否跳转到 S_setcc 状态和 S_idle 状态（操作码为 INST_HALT 时），否则跳转到 S_fetch 状态；S_setcc 状态无条件跳转到 S_fetch 状态。

状态机的第二部分是时序状态转换部分。在每个 posedge clk，此部分使 cs 转化为 ns 状态。如果 cs 是 S_idle 而 ns 是 S_fetch，保险起见还要将 buffer 清零。

状态机的第三部分是逻辑处理部分。这一部分指示了在各个状态我们的状态机还要执行的其他的操作。

- **S_idle**

首先是展示寄存器状态。通过开关 sw[7:0]以独热码形式来控制七段数码管显示内容，如 sw[7:0] = 8'b10000000 表示七段数码管应当展示寄存器 R7 的内容，格式为：R700XXXX，XXXX 为 R7 内容的十六进制编码。

其次是完成对 buffer 缓冲区的写入和对内存指令的写入。

- **S_fetch**

将 mem[PC]读入 IR，并使 PC 自增。

- **S_carry**

根据 IR 内容完成相应操作。这里的执行操作以 LC-3（以 *Introduction to Computing Systems: from Bits & Gates to C/C++ & Beyond, 3rd edition* 书中所述为准）为基础，将未定义的 unused 指令定义为 INST_RSF，即右移一位指令（高位补逻辑 0），并修改条件码；将在此不会用到的 INST_RTI 重定义为 INST_MOD，可以方便地取余数。各指令具体的实现操作如下所示。

```
INST_ADD:begin
    if(IR[5] == 1'b1)begin
        R[IR[11:9]] <= R[IR[8:6]] + {{11{IR[4]}}, IR[4:0]};
    end
    else begin
        R[IR[11:9]] <= R[IR[8:6]] + R[IR[2:0]];
    end
end
INST_AND:begin
    if(IR[5] == 1'b1)begin
        R[IR[11:9]] <= R[IR[8:6]] & {{11{IR[4]}}, IR[4:0]};
    end
    else begin
        R[IR[11:9]] <= R[IR[8:6]] & R[IR[2:0]];
    end
end
INST_BR:begin
    if((IR[11] & CC[2]) | (IR[10] & CC[1]) | (IR[9] & CC[0])) begin
        PC <= PC + IR[5:0];
        R[3'b110] <= 16'hFFFF;
    end
end
INST_JMP:begin
    PC <= R[IR[8:6]][5:0];
end
INST_JSR:begin
    R[7] <= PC;
    if(IR[11]) PC <= PC + IR[5:0];
    else PC <= R[IR[8:6]][5:0];
end
INST_LD:begin
    R[IR[11:9]] <= mem[PC + IR[5:0]];
end
INST_LDI:begin
    R[IR[11:9]] <= mem[mem[PC + IR[5:0]][5:0]];
end
INST_LDR:begin
    R[IR[11:9]] <= mem[R[IR[8:6]][5:0]+IR[5:0]];
end
```

```
INST_LEA:begin
    R[IR[11:9]] <= PC + IR[5:0];
end
INST_NOT:begin
    R[IR[11:9]] <= ~R[IR[8:6]];
end
INST_ST:begin
    mem[PC + IR[5:0]] <= R[IR[11:9]];
end
INST_STI:begin
    mem[mem[PC + IR[5:0]][5:0]] <= R[IR[11:9]];
end
INST_STR:begin
    mem[R[IR[8:6]][5:0] + IR[5:0]] <= R[IR[11:9]];
end
INST_RSF:begin
    R[IR[11:9]] <= {1'b0, R[IR[8:6]][7:1]};
end
INST_TRAP:begin
    //PC <= ENDL;
end
INST_MOD:begin
    R[IR[11:9]] <= R[IR[8:6]] % R[IR[2:0]];
end
```

- **S_setcc**

根据 IR 中 DR 寄存器内容设置条件码。

功能测试

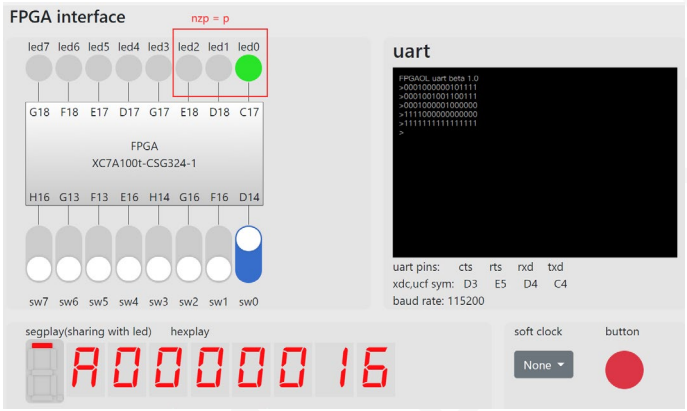
因为这是片上系统的实现，自然用一些示例程序来做测试较为合理，下面列出几种示例程序来测试程序功能。（上电时，所有临时寄存器 R[7:0]的值被初始化为 0）

（1）简单加法

串口输入：0001000000101111 // R0 = R0 + 0xF
0001001001100111 // R1 = R1 + 0x7
0001000001000000 // R0 = R1 + R0
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R0 == 0x16，CC == 3’b001

实际结果：

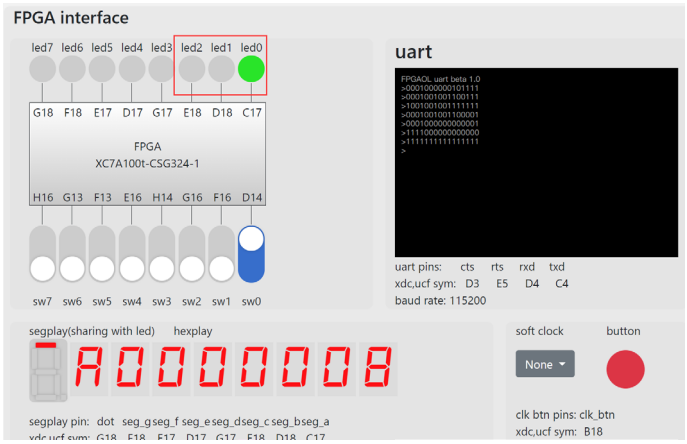


（2）简单减法

串口输入：0001000000101111 // R0 = R0 + 0xF
0001001001100111 // R1 = R1 + 0x7
1001001001111111 // R1 = ~R1
0001001001100001 // R1 = R1 + 1
0001000000000001 // R0 = R0 + R1
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R0 == 0x8，CC == 3’b001

实际结果：

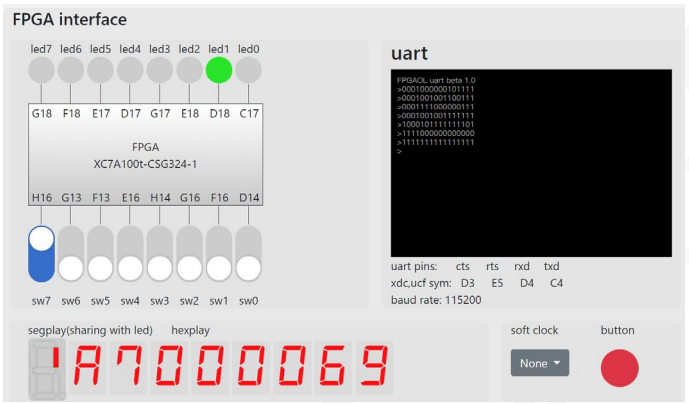


（3）乘法

串口输入：0001000000101111 // R0 = R0 + 0xF
0001001001100111 // R1 = R1 + 0x7
0001111000000111 // R7 = R0 + R7(*)
0001001001111111 // R1 = R1 – 1
1000101111111101 // if(R1 != 0) goto (*)
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R7 == 0x69，CC == 3’b010

实际结果：

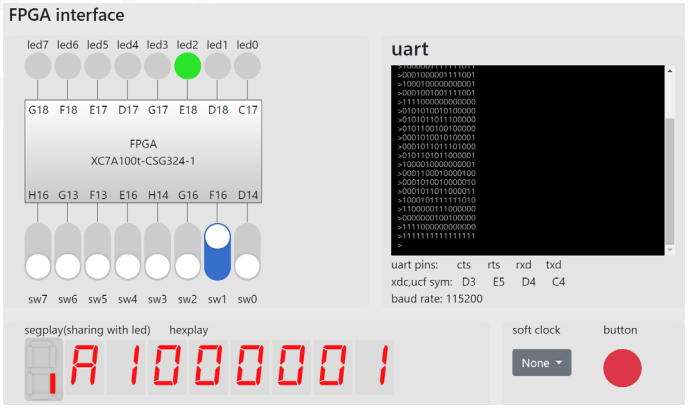


（4）求除以 7 的余数

串口输入：0010001000010101 // 用 LD 指令读入#288
0100100000001000 // 用 JSR 指令表示进入函数
0101010001100111
0001001010000100
0001000001111001
1000001111111011
0001000001111001
1000100000000001
0001001001111001
1111000000000000
0101010010100000
0101011011100000
0101100100100000
0001010010100001
0001011011101000
0101101011000001
1000010000000001
0001100010000100
0001010010000010
0001011011000011
1000101111111010
1100000111000000
0000000100100000 // #288
1111000000000000 // HALT
1111111111111111 // RUN

期望结果：R1 == 0x1 [288≡1 (mod 7)]

实际结果：



（5）用 INST_RSF 实现逻辑右移

串口输入：0001000000101111 // R0 = R0 + 0xF

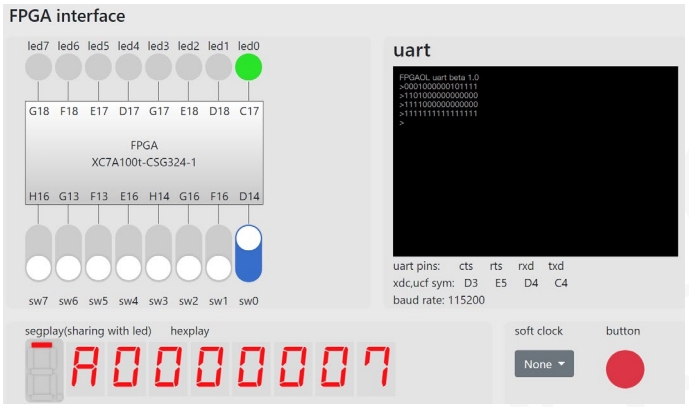
1101000000000000 // R0 = R0 >> 1

1111000000000000

1111111111111111

期望结果：R0 == 0x7

实际结果：



经过以上测试，已经可以大致确认本实验成果具有相当的稳定性和正确性。

总结与思考

- 本次实验中我学会了除了 sw[7:0]、button 之外另一种和 FPGA 交互的方式：串口。并且这种交互方式自由度极大，几乎可以利用其实现任何交互功能。
- 本次实验是在本学期课程《计算系统概论 A》所学内容上的扩充，由于工作量较大、逻辑设计容量较大，导致本次实验的难度较大。
- 本次实验的任务量是视所选题材的具体内容而定的。衡量本实验，需要从创新性和复杂性中找到一种平衡。有的题材虽然任务量很重，但是大多是重复的操作，用任务量来衡量就毫无意义。而本题材灵感取自课本所学，在这之上加以扩充，实现在这门课的编程中较难实现的一些指令，如右移运算。
- 改进建议：详细介绍一下 fpgaol 平台上的串口使用，原使用文档中的串口还是实体外设，和虚拟平台略有出入。

附件

myISA.v //设计文件

myISA-xdc.xdc //约束文件