《计算机组成与设计》

课程设计报告

课程设计题目：基于微指令的简单模型机设计

班级：学堂计机 20

姓名：杨业昶

学号：202022300317

完成日期：2022 年 5 月 20 日

目录

一、 本设计实现的指令集

本设计参考了 MIPS、RISC-V、8086 和课程设计提供的指令集，进行了一定程度的缩减和修改，设计了 16 位的指令系统，并命名为 SDHR16-Primary。包括基本逻辑和算术运算、访存、跳转等，为了进一步缩减指令执行周期数还设计了立即运算指令。

指令功能如下：

指令名称	字节码				意义
	15..9	8..6	5..3	2..0	
取指	'0000000	xxx	xxx	xxx	取指令
SLLV	'0001000	r1	r2	r3	左移
SLLV.i	'0001011	r1	xxx	xxx	左移-立即运算
SRLV	'0001100	r1	r2	r3	右移
SRLV.i	'0001111	r1	xxx	xxx	右移-立即运算
SLTU	'0010000	r1	r2	r3	小于
SLTU.i	'0010011	r1	xxx	xxx	小于-立即运算
ADD	'0010100	r1	r2	r3	算数加
ADD.i	'0010111	r1	xxx	xxx	算数加-立即运算
SUB	'0011000	r1	r2	r3	算数减
SUB.i	'0011011	r1	xxx	xxx	算数减-立即运算
AND	'0011100	r1	r2	r3	逻辑与
AND.i	'0011111	r1	xxx	xxx	逻辑与-立即运算
OR	'0100000	r1	r2	r3	逻辑或
OR.i	'0100011	r1	xxx	xxx	逻辑或-立即运算
XOR	'0100100	r1	r2	r3	异或
XOR.i	'0100111	r1	xxx	xxx	异或-立即运算
NXOR	'0101000	r1	r2	r3	异或非
NXOR.i	'0101011	r1	xxx	xxx	异或非-立即运算
JNZ	'0101100	r1	r2	xxx	条件跳转
JNZR	'0110000	r1	r2	xxx	条件相对跳转
JMP	'0110101	r1	xxx	xxx	无条件跳转
JMPR	'0110110	r1	xxx	xxx	无条件相对跳转
LD	'0111001	r1	r2	xxx	取数
STR	'0111100	r1	r2	xxx	存数
MUL	'0111111	r1	r2	r3	乘法（含高低位）
DIV	'1000011	r1	r2	r3	除法（含除数和商）
MOV	'1000111	r1	r2	xxx	传数
HALT	'1001000	xxx	xxx	xxx	停机
NOP	'1001001	xxx	xxx	xxx	空指令

寄存器功能如下：

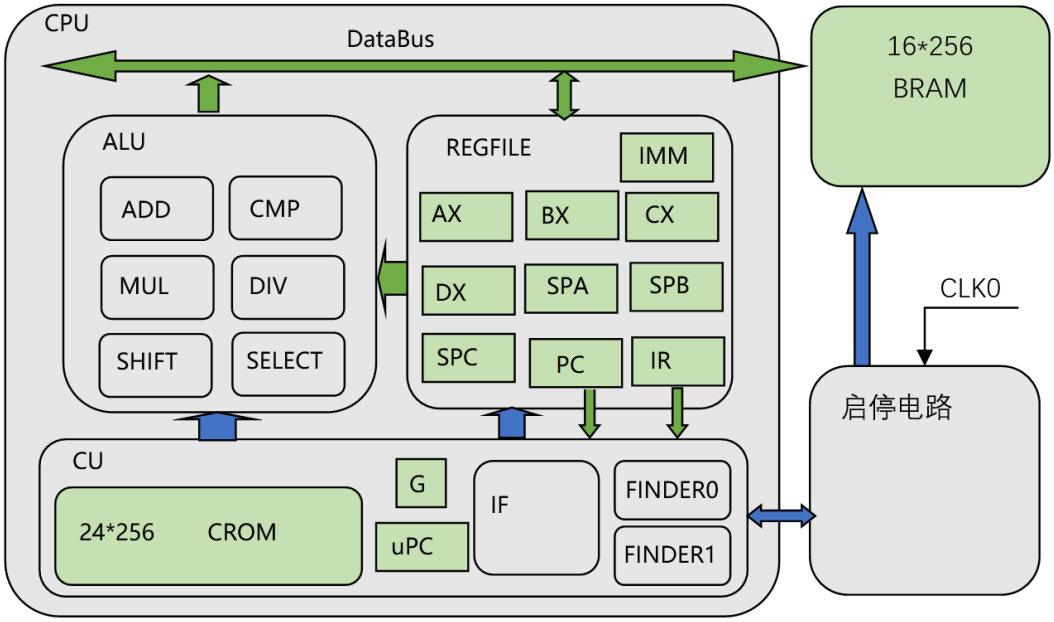
寄存器名	寄存器编号	功能
AX	0	普通寄存器

BX	1	普通寄存器
CX	2	普通寄存器
DX	3	普通寄存器
SPA	4	运算操作数 1
SPB	5	运算操作数 2
SPC	6	运算操作数 3
IMM	7	立即数

对于需要立即数的指令，指令字长为 32 位，即 16 位原指令+16 位立即数。

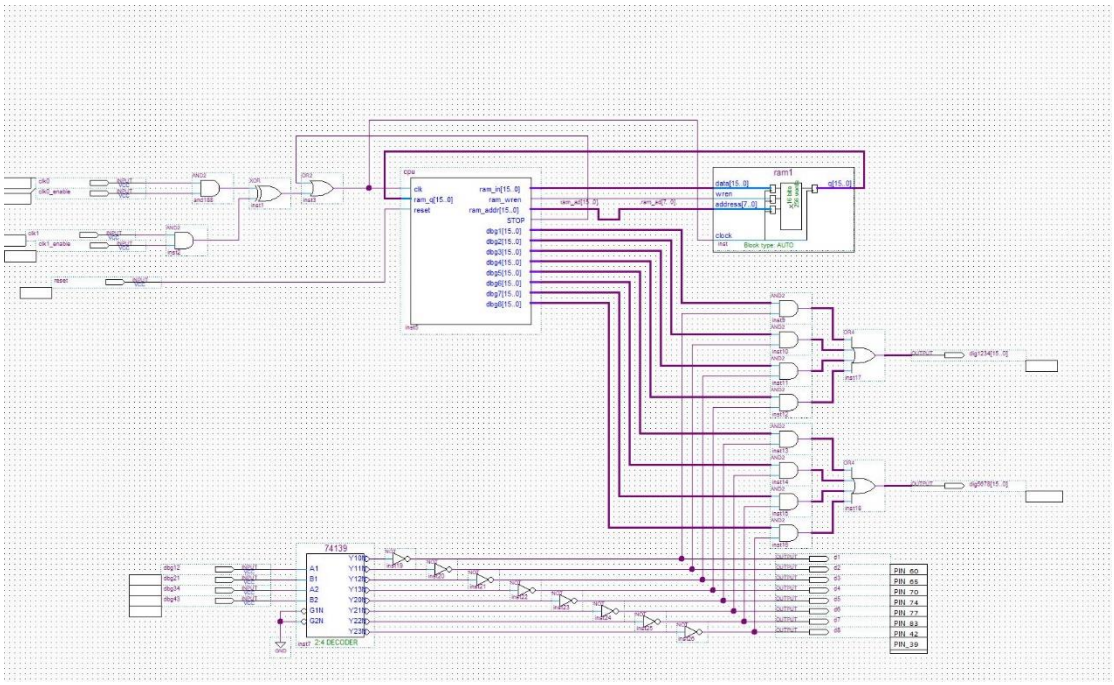
对于上表中“立即运算”的含义，是因为本 CPU 采用了周期数不定的运算指令，对于可以省略的计算减少不必要的传数周期，以减少指令时间。

二、 设计思路与总体结构框图



图表 1 本设计主要部分结构框图

上图中绿色箭头表示数据传输方向，蓝色箭头表示控制信号传输方向。ALU 是运算器、CU 是控制器、REGFILE 是寄存器堆。其中 CU 是最主要的控制部件，本结构有一条片内总线 DataBus，ALU 向寄存器回传、寄存器复写、寄存器与 RAM 之间交换数据都通过 DataBus 进行。

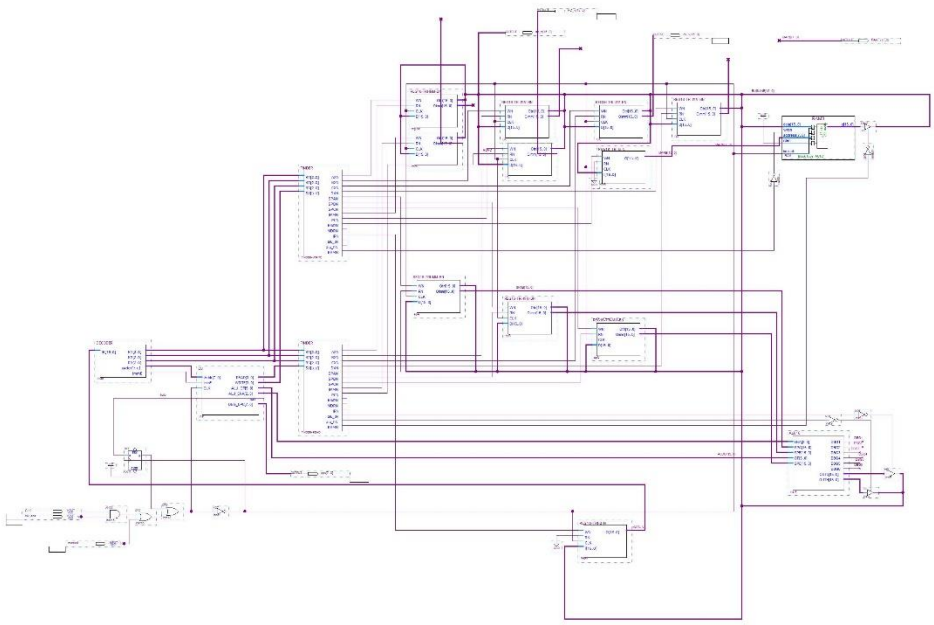


图表 2 soc 顶层设计图

1. soc 顶层设计

顶层设计图封装了 CPU、RAM、启停电路，并从 CPU 额外引出了 8 个 16bit 输出，方便设计人员利用 FPGA 开发板上进行调试，避免了在 CPU 修改调试优化的过程中反复修改引脚配置。

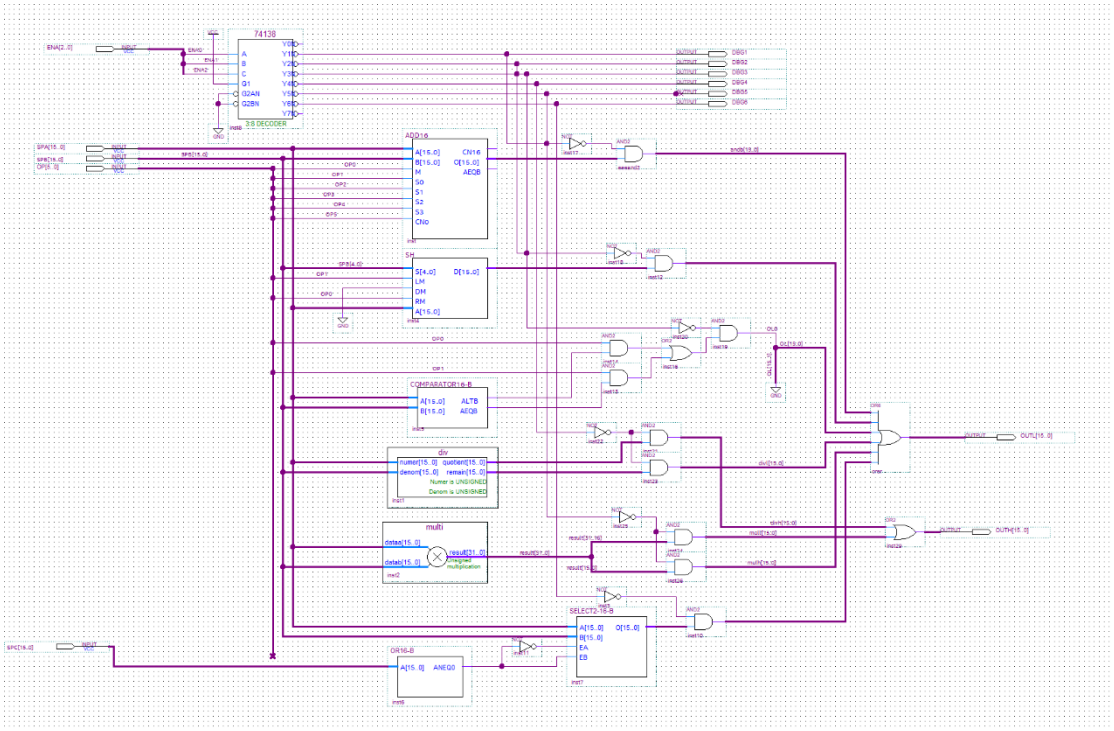
2. CPU 整体设计



图表 3 CPU 整体设计图

因为引脚众多，整体设计图并没有对寄存器堆和 CU 做完整的显式封装，各个部件逻辑上的功能下文将逐一展示。

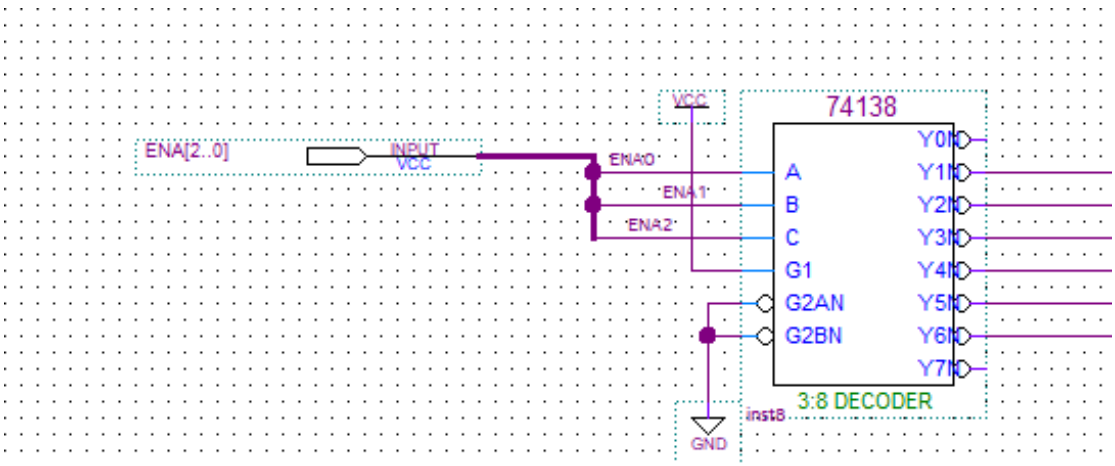
3. 运算器 ALU



图表 4 ALU 整体设计

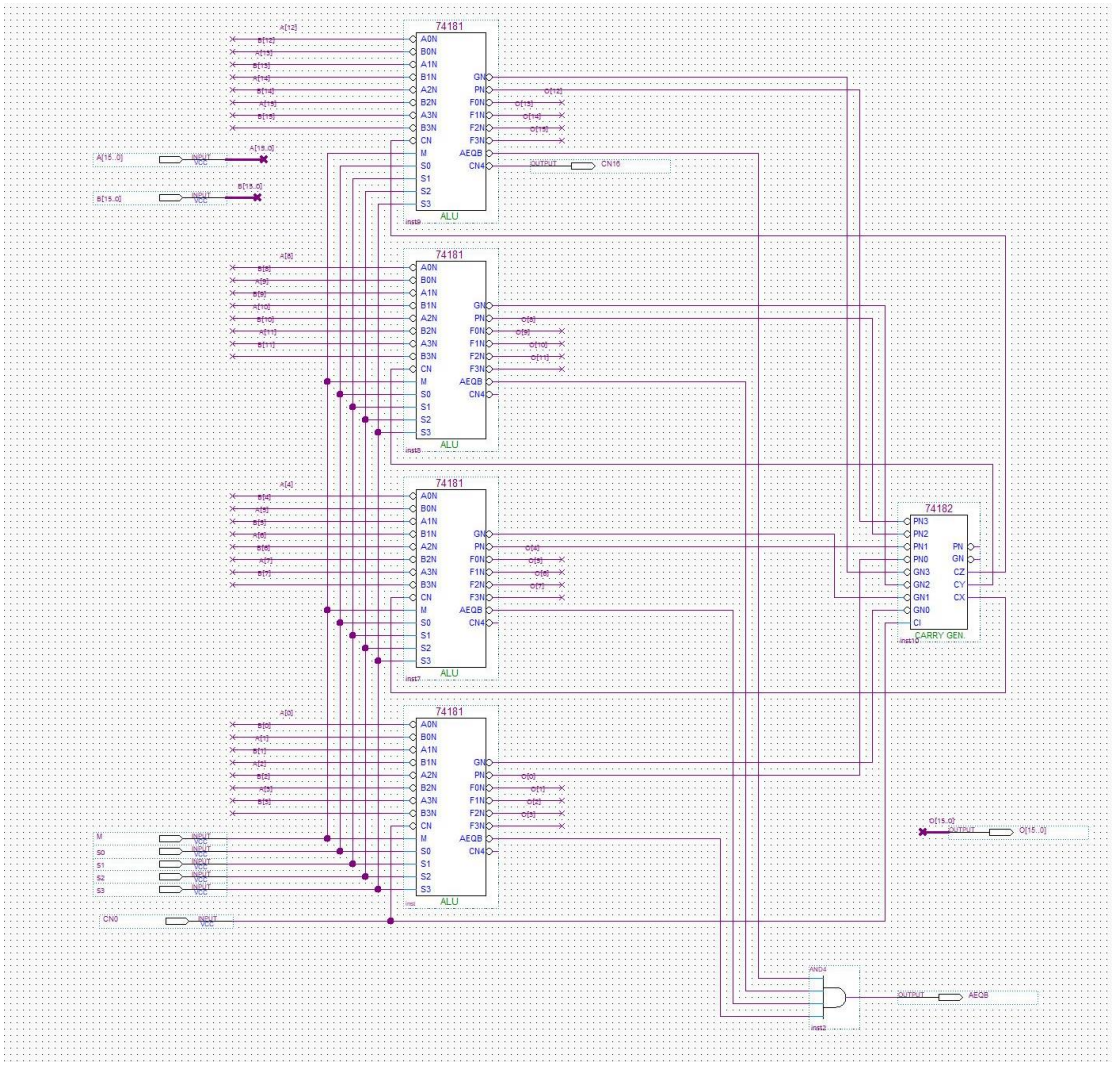
ALU 由 ADD16、SHIFT、 CMP、SELECT、MUL、DIV 六个运算子部件组成，其中 ADD16 承担加减运算和位运算，CMP 单独实现了相等和小于两个比较运算，MUL 和 DIV 是 Quartus 内置的乘除法 IP 核，SELECT 是二选一选择器。ALU 在进行运算时，根据 CU 给出的使能码 ENA 和操作码 OP，选择一个部件并进行相应的计算。

ALU 有三个输入端、两个输出端，目的是可以同时给出乘法高低位、除法余数和商的运算结果，从而减少需要完成一项作业所需的周期数。三个输入端固定由 SPA、SPB、SPC 读入，对任意两个操做数进行运算时，需先将操作数传入上述三个的寄存器。如果不需要某个 SP 寄存器或操作数已经存放在相应寄存器中，则可以省略传数操作。



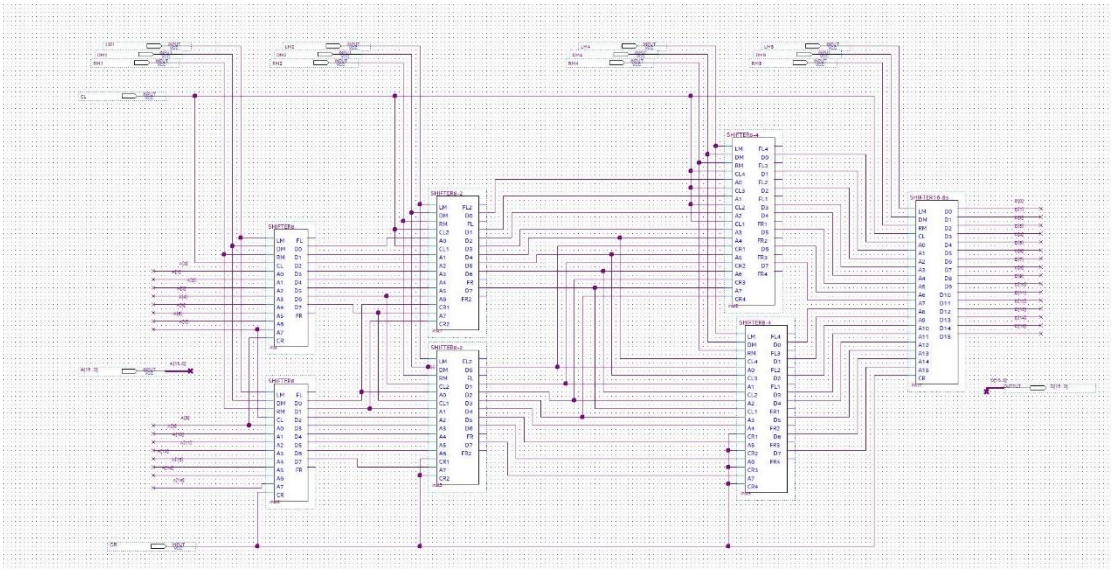
图表 5 选择运算译码器

运算部件的选择是由一片 138 译码器决定的，该译码器对 CU 发出的三位使能信号进行译码，激活某个运算部件后面的与门使其结果传送到片内总线上。



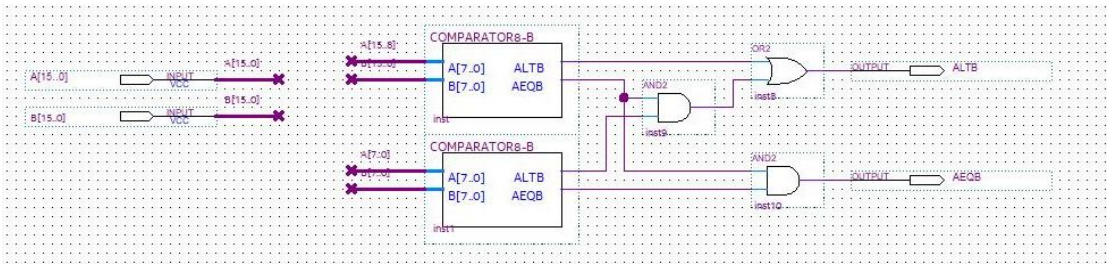
图表 6 ADD16

ADD16 运算部件承担加减运算和位运算功能，由 4 片 74181 运算器和一片 74182 实现超前进位。

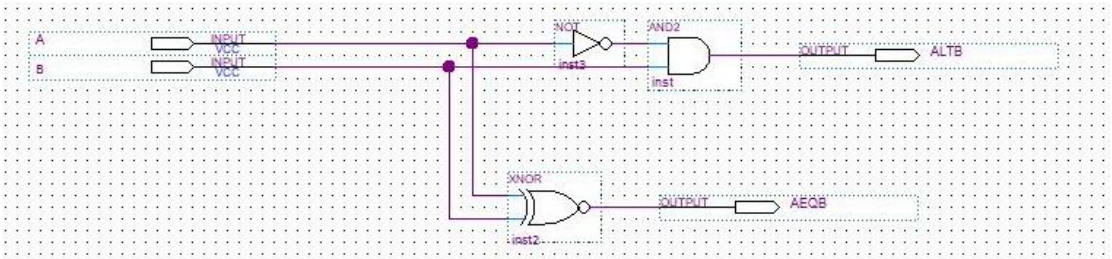


图表 7 SHIFT 的主要部分

SHIFT 部件是自主设计实现的移位单元，承担移位算功能。硬件上支持直传、算数移位、逻辑移位。SH 采用幂次移位算法，将移位数拆解成移动 1、2、4、8 位，以减少设计难度、节省计算资源。由于完整结构十分庞大，上图仅仅展示了 SHIFT 最主要的部分。

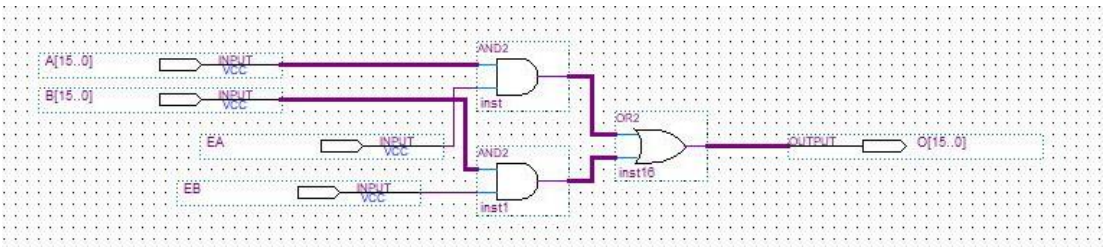


图表 8 CMP 顶层设计



图表 9 CMP 底层设计

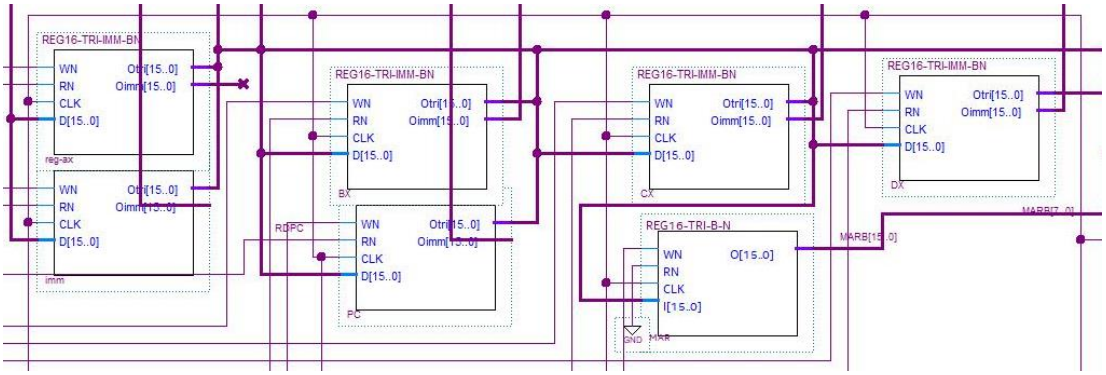
CMP 部件是自主设计实现的比较部件，承担比较运算功能，支持小于和等于运算，独立计算并输出，可扩展。CMP 采用二分法进行比较，以缩减运算回路长度和提高运算速度。

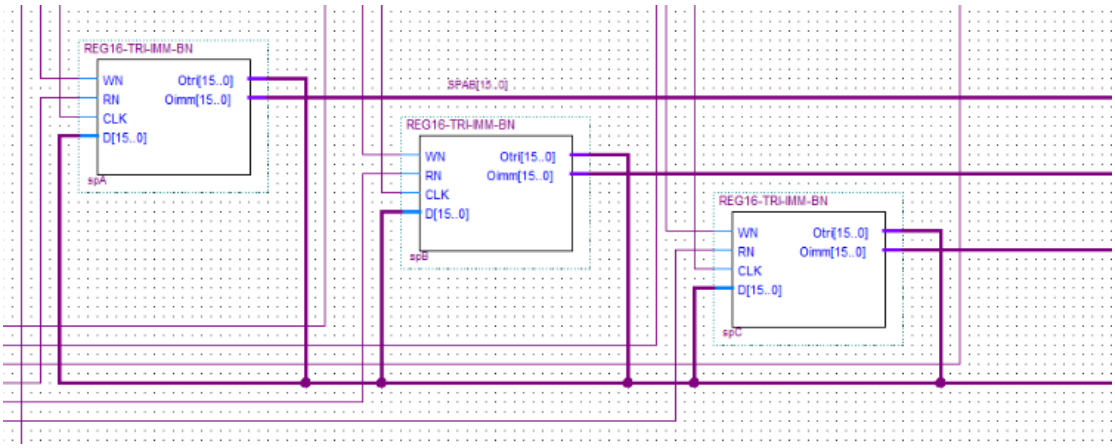


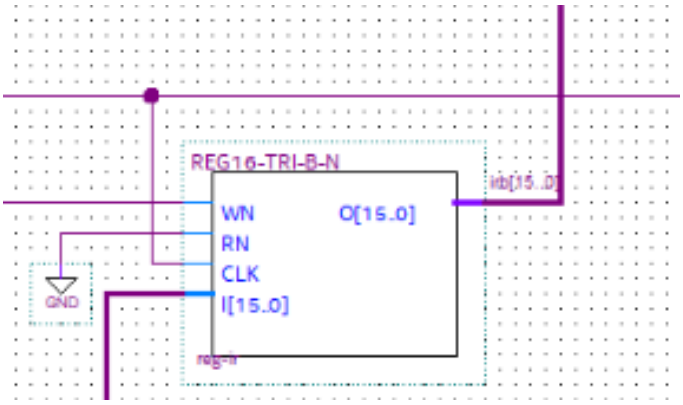
图表 10 SELECT

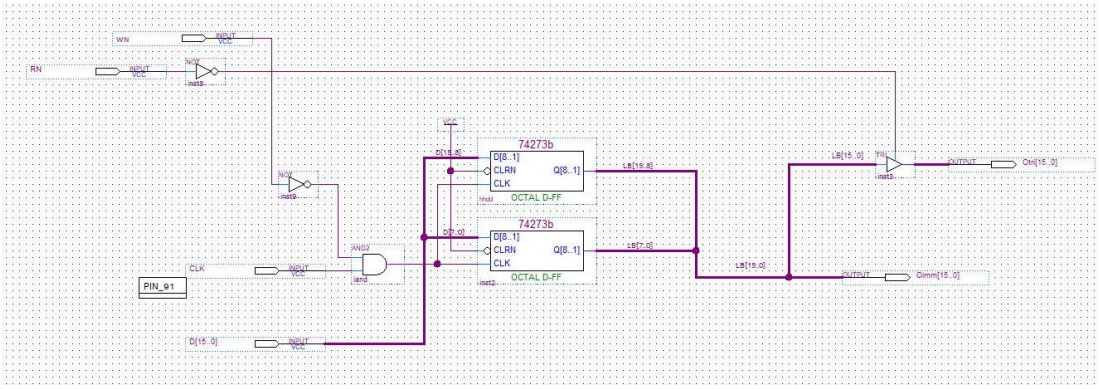
SELECT 部件是二选一选择器，这也是 ALU 中唯一需要指定三个操作源的运算部件。

4. REGFILE 寄存堆









图表 11 三态输出 16 位寄存器

寄存器堆中有四个普通寄存器 AX、BX、CX、DX，三个操作数寄存器 SPA、SPB、SPC，立即数寄存器 IMM，程序计数器 PC，内存地址寄存器 MAR，指令寄存器 IR。

寄存器支持直接输出和三态输出，寄存器内部采用两片 74273 实现。

普通寄存器有四个的原因是，不进行间接寻址访问时，至少需要四个通用寄存器才能编写出流畅的程序。

立即数寄存器，对于需要立即数的指令，CU 会将立即数存放到该寄存器内，访问方式与普通寄存器相同。值得注意的是，IMM 不可以作为任何指令的目的寄存器。

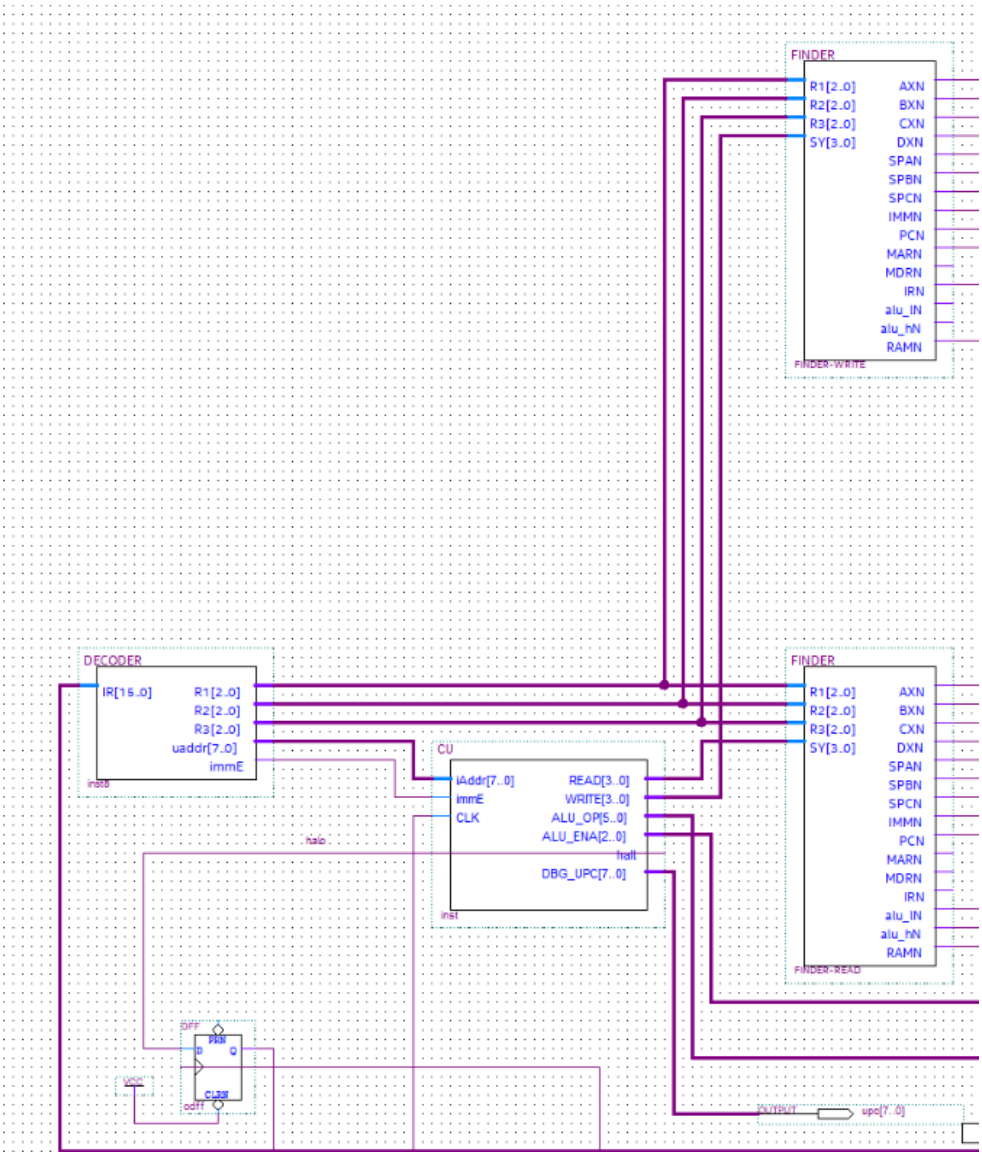
程序计数器 PC 的+1 操作由 ALU 配合微指令实现，硬件上没有实现。

每个操作数寄存器 SPx 有两个输出端口，直接输出固定连接到 ALU 的数据端口上，三态输出连接到 DataBus 上。

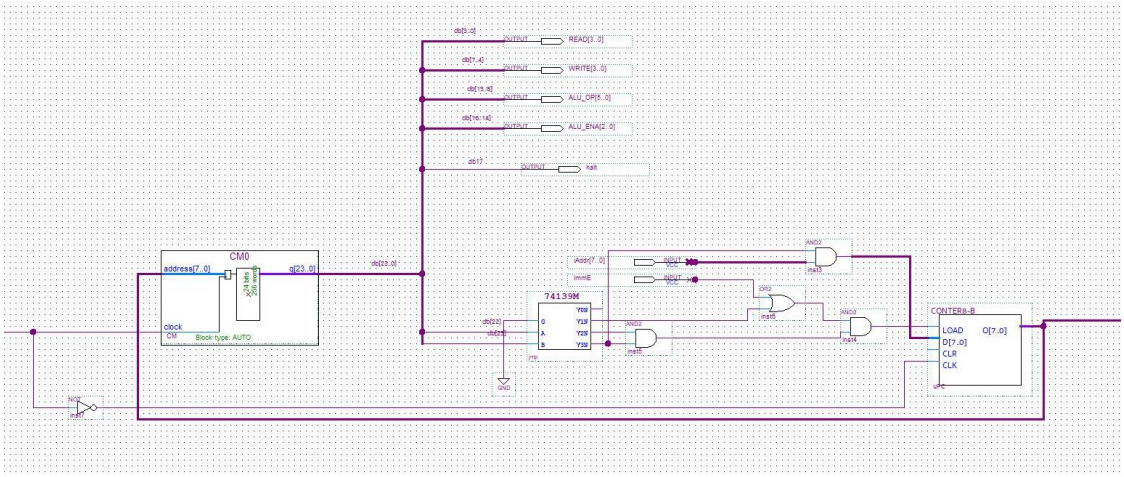
三、 基于微指令的 CU 实现

CU 利用时钟周期 CLK 的上升和下降两个沿，下降沿将 ROM 中的微指令读出，上升沿进行其他操作。

1. CU 总体结构



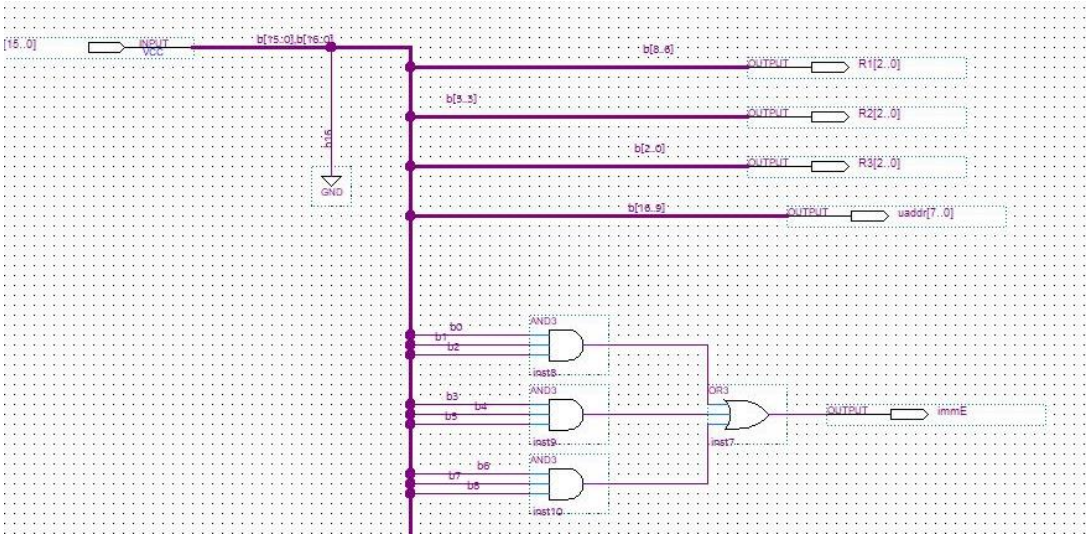
图表 12 CU 顶层结构



图表 13 CU **内部**

整个 CU 分为微指令存储器 CM、微计数器 uPC、译码器 IF、寄存器读写选择器 FINDERI 和 FINDERO。

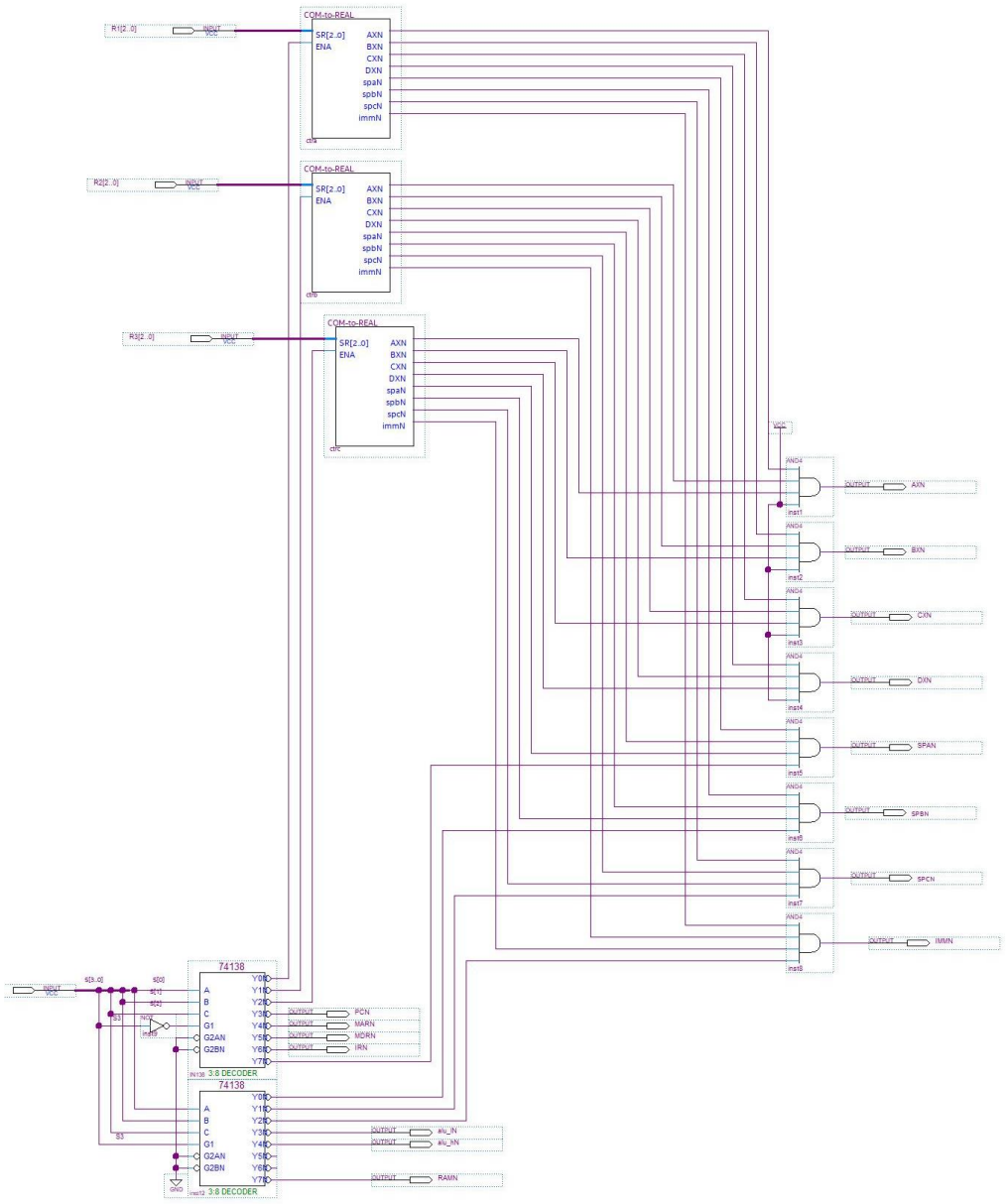
2. 译码器



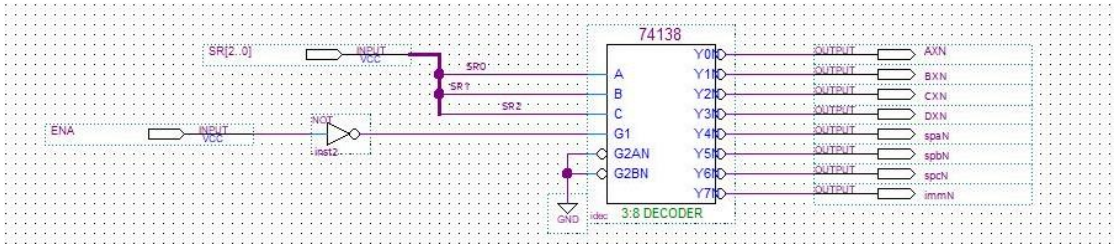
图表 14 **译码器** IF

要执行下一条指令，首先根据 PC 的值从 RAM 中将指令读出、送入到 IR，然后由 IF 将指令拆分为操作码 op、实际寄存器号，op 送入 uPC 下地址选择器实现微指令的跳转，实际寄存器号与微指令提供的形式寄存器号翻译形成寄存器控制信号。

3. 寄存器选择器



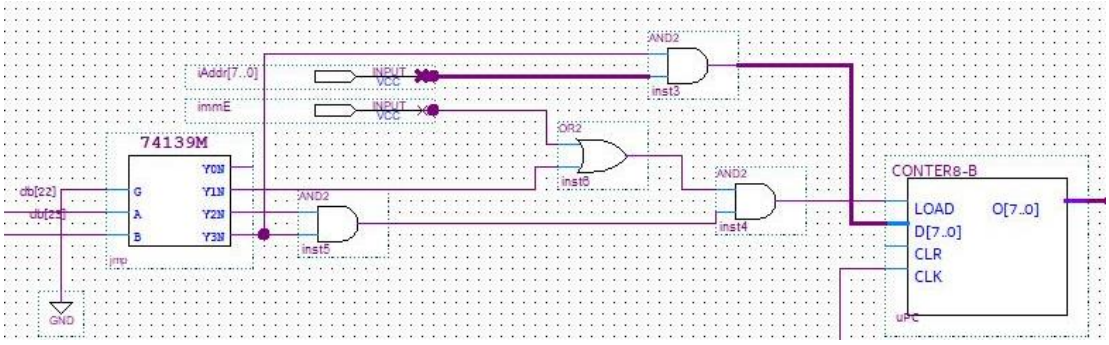
图表 15 寄存器选择器结构



图表 16 实际寄存器译码

寄存器选择器根据微指令给出的形式寄存器 R1R2R3 和用户指令的内容翻译出实际寄存器的读写信号。选择功能通过 38 译码器和若干个与门实现。

4. 下地址选择器



图表 17 下地址选择部件和 uPC

下地址选择器在硬件上支持指定、自增、清零三种方式。需要取指时将通过下地址选择器将 uPC 清零，以执行取指部分的微程序；执行指令时给定地址微指令地址和指定信号，以跳转到微指令首地址；多周期指令执行过程中，自增信号一直有效。

5. 微指令表

地址	指令	周期	字节码							解释
			jmp	blank	G	alu_ena	alu_op	write	read	
			23..22	N/A	17	16..14	13..8	7..4	3..0	
00	取指	1	'00	'0000	'0	'000	'000000	'0100	'0011	pc->mar
01	取指	2	'00	'0000	'0	'000	'000000	'0111	'0011	pc->spa
02	取指	3	'00	'0000	'0	'000	'000000	'0110	'1111	ram->ir
03	取指	4	'01	'0000	'0	'001	'000000	'0011	'1011	pc++,jmp1
04	取指	5	'00	'0000	'0	'000	'000000	'0100	'0011	pc->mar
05	取指	6	'00	'0000	'0	'000	'000000	'0111	'0011	pc->spa
06	取指	7	'00	'0000	'0	'000	'000000	'1010	'1111	ram->imm
07	取指	8	'10	'0000	'0	'001	'000000	'0011	'1011	pc++,jmp2
08	SLLV	1	'00	'0000	'0	'000	'000000	'0111	'0001	r2->spa
09	SLLV	2	'00	'0000	'0	'000	'000000	'1000	'0010	r3->spb
0A	SLLV	3	'11	'0000	'0	'010	'000001	'0000	'1011	ans->r1,jmp0
0B	SLLV.i	1	'11	'0000	'0	'010	'000001	'0000	'1011	ans->r1,jmp0
0C	SRLV	1	'00	'0000	'0	'000	'000000	'0111	'0001	r2->spa
0D	SRLV	2	'00	'0000	'0	'000	'000000	'1000	'0010	r3->spb
0E	SRLV	3	'11	'0000	'0	'010	'000010	'0000	'1011	ans->r1,jmp0
0F	SRLV.i	1	'11	'0000	'0	'010	'000010	'0000	'1011	ans->r1,jmp0
10	SLTU	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
11	SLTU	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
12	SLTU	3	'11	'0000	'0	'011	'000001	'0000	'1011	ans->r1,jmp0
13	SLTU.i	1	'11	'0000	'0	'011	'000001	'0000	'1011	ans->r1,jmp0
14	ADD	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
15	ADD	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb

16	ADD	3	'11	'0000	'0	'001	'110010	'0000	'1011	ans->r1,jmp0
17	ADD.i	1	'11	'0000	'0	'001	'110010	'0000	'1011	ans->r1,jmp0
18	SUB	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
19	SUB	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
1A	SUB	3	'11	'0000	'0	'001	'001100	'0000	'1011	ans->r1,jmp0
1B	SUB.i	1	'11	'0000	'0	'001	'001100	'0000	'1011	ans->r1,jmp0
1C	AND	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
1D	AND	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
1E	AND	3	'11	'0000	'0	'001	'010111	'0000	'1011	ans->r1,jmp0
1F	AND.i	1	'11	'0000	'0	'001	'010111	'0000	'1011	ans->r1,jmp0
20	OR	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
21	OR	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
22	OR	3	'11	'0000	'0	'001	'011101	'0000	'1011	ans->r1,jmp0
23	OR.i	1	'11	'0000	'0	'001	'011101	'0000	'1011	ans->r1,jmp0
24	XOR	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
25	XOR	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
26	XOR	3	'11	'0000	'0	'001	'001101	'0000	'1011	ans->r1,jmp0
27	XOR.i	1	'11	'0000	'0	'001	'001101	'0000	'1011	ans->r1,jmp0
28	NXOR	1	'00	'0000	'0	'000	'000000	'1000	'0010	r2->spa
29	NXOR	2	'00	'0000	'0	'000	'000000	'0111	'0001	r3->spb
2A	NXOR	3	'11	'0000	'0	'001	'010011	'0000	'1011	ans->r1,jmp0
2B	NXOR.i	1	'11	'0000	'0	'001	'010011	'0000	'1011	ans->r1,jmp0
2C	JNZ	1	'00	'0000	'0	'000	'000000	'1001	'0000	r1->spc
2D	JNZ	2	'00	'0000	'0	'000	'000000	'0111	'0011	pc->spa
2E	JNZ	3	'00	'0000	'0	'000	'000000	'1000	'0001	r2->spb
2F	JNZ	4	'11	'0000	'0	'110	'000001	'0011	'1011	ans->pc,jmp0
30	JNZR	1	'00	'0000	'0	'000	'000000	'1001	'0000	r1->spc
31	JNZR	2	'00	'0000	'0	'000	'000000	'0111	'0011	pc->spa
32	JNZR	3	'00	'0000	'0	'000	'000000	'1000	'0001	r2->spb
33	JNZR	4	'00	'0000	'0	'001	'110010	'1000	'1011	spa+spb->spb
34	JNZR	5	'11	'0000	'0	'110	'000001	'0011	'1011	ans->pc,jmp0
35	JMP	1	'11	'0000	'0	'000	'000000	'0011	'0000	r1->pc,jmp0
36	JMPR	1	'00	'0000	'0	'000	'000000	'0111	'0011	pc->spa
37	JMPR	2	'00	'0000	'0	'000	'000000	'1000	'0000	r1->spb
38	JMPR	3	'11	'0000	'0	'001	'110010	'0011	'1011	ans->pc,jmp0
39	LD	1	'00	'0000	'0	'000	'000000	'0100	'0001	r2->mar
3A	LD	2	'00	'0000	'0	'000	'000000	'0000	'0000	wait
3B	LD	3	'11	'0000	'0	'000	'000000	'0000	'1111	ram->r1,jmp0
3C	STR	1	'00	'0000	'0	'000	'000000	'0100	'0001	r2->mar
3D	STR	2	'00	'0000	'0	'000	'000000	'1111	'0000	r1->ram
3E	STR	3	'11	'0000	'0	'000	'000000	'0000	'0000	jmp0
3F	MUL	1	'00	'0000	'0	'000	'000000	'1000	'0010	r3->spb
40	MUL	2	'00	'0000	'0	'000	'000000	'0111	'0001	r2->spa
41	MUL	3	'00	'0000	'0	'101	'000010	'1001	'1100	high->spc
42	MUL	4	'11	'0000	'0	'101	'000010	'0000	'1011	low->r1,jmp0

43	DIV	1	'00	'0000	'0	'000	'000000	'1000	'0010	r3->spb
44	DIV	2	'00	'0000	'0	'000	'000000	'0111	'0001	r2->spa
45	DIV	3	'00	'0000	'0	'100	'000010	'1001	'1100	high->spc
46	DIV	4	'11	'0000	'0	'100	'000010	'0000	'1011	low->r1,jmp0
47	MOV	1	'11	'0000	'0	'000	'000000	'0000	'0001	r2->r1,jmp0
48	HALT	1	'00	'0000	'1	'000	'000000	'0000	'0000	G
49	NOP	1	'11	'0000	'0	'000	'000000	'0000	'0000	wait

6. 测试程序

根据指令集编写了两个测试程序，附录中提供了程序的二进制指令码。

```
; 示例程序 1
; 下面的程序对若干个数字求和
; ad1 存放数字的个数，表示接下来有[ad1]个数需要求和
; 计算的结果保存在 ad2 中

  MOV AX, .ad1
  LD  BX, AX
  ADD BX, BX, AX
  MOV CX, 0
.loop
  ADD AX, AX, 1
  LD  DX, AX
  ADD CX, CX, DX
  XOR DX, AX, BX
  JNZ  DX, .loop
  STR  CX, .ad2
  HALT
.ad1
3
101
102
103
104
105
.ad2


; 示例程序 2
; 测试所有的计算类指令

MOV AX, 15
```

```
MOV BX, 3

MOV CX, .esp

ADD DX, AX, BX
STR DX, CX
ADD CX, CX, 1

SUB DX, AX, BX
STR DX, CX
ADD CX, CX, 1

SLLV DX, AX, BX
STR DX, CX
ADD CX, CX, 1

SRLV DX, AX, BX
STR DX, CX
ADD CX, CX, 1

SLTU DX, AX, BX
STR DX, CX
ADD CX, CX, 1

AND DX, AX, BX
STR DX, CX
ADD CX, CX, 1

OR DX, AX, BX
STR DX, CX
ADD CX, CX, 1

XOR DX, AX, BX
STR DX, CX
ADD CX, CX, 1

NXOR DX, AX, BX
STR DX, CX
ADD CX, CX, 1

MUL DX, AX, BX
STR DX, CX
ADD CX, CX, 1
```

```
STR spc, CX
ADD CX, CX, 1

DIV DX, AX, BX
STR DX, CX
ADD CX, CX, 1

STR spc, CX
ADD CX, CX, 1

JNZR 0, AX

JMPR -2

0
.esp
```

四、 课程设计总结

1. 设计过程总体感受

硬件电路设计的好坏对整体架构的要求相当高，这是与软件设计最大的不同。顶层架构的设计直接关系到硬件电路的性能和设计难度，一旦顶层架构确定后期将很难修改。

硬件设计工具的功能十分有限，大量的时间花费在极其有限的调试手段中，很多情况下要针对特定硬件设计定制化开发软件辅助工具，且网络上相关教程等资源十分匮乏，与软件开发相比有更高的学习门槛。

2. 遇到的困难以及解决方法

本次是课程设计中对于 BRAM 间隔一个周期才能将数据读出的问题，我重新设计了微指令的访存操作，调整了访存顺序，额外增加了等待周期。

对于数码管数量不足的问题，我设计了 soc 顶层电路，增加了调试输出选择部件。

对于 quartus 无法仿真的问题，我尝试使用 modelsim，但是并没有解决，最终回退到了 quartus ii13.0 版本才得到解决。

对于寄存器三态 bidir 编译报错的问题，我放弃了用 bidir 引脚输入输出的方式，将三态输出和输入接到同一条总线上。

3. 致谢

感谢陈正同学交流探讨基于片内总线的结构实现。
感谢黄敬成学长提供的实验报告示例。

五、 附录

1. 微指令编码含义配置

```
CM_CAPACITY_BITS = 7
CM_CAPACITY = 2 ** CM_CAPACITY_BITS

U_INS_ACTIONS = [
   {
      "name": "取指",
      "uaddr": 0,
      "operand": 0,
      "actions": [
         {   # 0
            "read": "pc",
            "write": "mar"
         }, {  # 1
            "read": "pc",
            "write": "spa"
         }, {  # 2
            "read": "ram",
            "write": "ir",
         }, {  # 3
            "read": "alu_l",
            "write": "pc",
            "alu_op": "a+1",
            "alu_ena": "add",
            "jmp": "if"
         }, {  # 4
            "read": "pc",
            "write": "mar"
         }, {  # 5
            "read":  "pc",
            "write": "spa"
         }, {  # 6
            "read": "ram",
```

```
            "write": "imm",
        }, {  # 7
            "read": "alu_l",
            "write": "pc",
            "alu_op": "a+1",
            "alu_ena": "add",
            "jmp": "force"
        },
    ]
}, {
    "name": "SLLV",
    "operand": 3,
    "actions": [
        {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "sh_left",
            "alu_ena": "sh",
            "jmp": "0"
        }
    ]
}, {
    "name": "SLLV.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "sh_left",
            "alu_ena": "sh",
            "jmp": "0"
        }
    ]
}, {
    "name": "SRLV",
    "operand": 3,
    "actions": [
        {
```

```
            "read": "r2",
            "write": "spa"
        }, {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "sh_right",
            "alu_ena": "sh",
            "jmp": "0"
        }
    ]
}, {
    "name": "SRLV.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "sh_right",
            "alu_ena": "sh",
            "jmp": "0"
        }
    ]
}, {
    "name": "SLTU",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a<b",
            "alu_ena": "cmp",
            "jmp": "0"
        }
    ]
}, {
```

```
    "name": "SLTU.i",
    "operand": 1,
    "actions": [
      {
        "read": "alu_l",
        "write": "r1",
        "alu_op": "a<b",
        "alu_ena": "cmp",
        "jmp": "0"
      }
    ]
  }, {
    "name": "ADD",
    "operand": 3,
    "actions": [
      {
        "read": "r3",
        "write": "spb"
      }, {
        "read": "r2",
        "write": "spa"
      }, {
        "read": "alu_l",
        "write": "r1",
        "alu_op": "a+b",
        "alu_ena": "add",
        "jmp": "0"
      }
    ]
  }, {
    "name": "ADD.i",
    "operand": 1,
    "actions": [
      {
        "read": "alu_l",
        "write": "r1",
        "alu_op": "a+b",
        "alu_ena": "add",
        "jmp": "0"
      }
    ]
  }, {
    "name": "SUB",
    "operand": 3,
```

```
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a-b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "SUB.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a-b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "AND",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a&b",
            "alu_ena": "add",
            "jmp": "0"
        }
```

```
    ]
}, {
    "name": "AND.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a&b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "OR",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a|b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "OR.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a|b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
```

```
    "name": "XOR",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a^b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "XOR.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a^b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "NXOR",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a nxor b",
            "alu_ena": "add",
```

```
            "jmp": "0"
        }
    ]
}, {
    "name": "NXOR.i",
    "operand": 1,
    "actions": [
        {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a nxor b",
            "alu_ena": "add",
            "jmp": "0"
        }
    ]
}, {
    "name": "JNZ",
    "operand": 2,
    "actions": [
        {
            "read": "r1",
            "write": "spc"
        }, {
            "read": "pc",
            "write": "spa"
        }, {
            "read": "r2",
            "write": "spb"
        }, {
            "read": "alu_l",
            "write": "pc",
            "alu_op": "=0",
            "alu_ena": "se",
            "jmp": "0"
        }
    ]
}, {
    "name": "JNZR",
    "operand": 2,
    "actions": [
        {
            "read": "r1",
            "write": "spc"
        }, {
```

```
            "read": "pc",
            "write": "spa"
        }, {
            "read": "r2",
            "write": "spb"
        }, {
            "read": "alu_l",
            "write": "spb",
            "alu_op": "a+b",
            "alu_ena": "add",
        }, {
            "read": "alu_l",
            "write": "pc",
            "alu_op": "=0",
            "alu_ena": "se",
            "jmp": "0"
        }
    ]
}, {
    "name": "JMP",
    "operand": 1,
    "actions": [
        {
            "read": "r1",
            "write": "pc",
            "jmp": "0"
        }
    ]
}, {
    "name": "JMPR",
    "operand": 1,
    "actions": [
        {
            "read": "pc",
            "write": "spa"
        }, {
            "read": "r1",
            "write": "spb"
        }, {
            "read": "alu_l",
            "write": "pc",
            "alu_op": "a+b",
            "alu_ena": "add",
            "jmp": "0"
```

```
        }
    ]
}, {
    "name": "LD",
    "operand": 2,
    "actions": [
        {
            "read": "r2",
            "write": "mar"
        }, {
            # WAIT
        }, {
            "read": "ram",
            "write": "r1",
            "jmp": "0"
        }
    ]
}, {
    "name": "STR",
    "operand": 2,
    "actions": [
        {
            "read": "r2",
            "write": "mar"
        }, {
            "read": "r1",
            "write": "ram",
        }, {
            # WAIT
            "jmp": "0"
        }
    ]
}, {
    "name": "MUL",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_h",
```

```
            "write": "spc",
            "alu_op": "a*b",
            "alu_ena": "mul",
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a*b",
            "alu_ena": "mul",
            "jmp": "0"
        }
    ]
}, {
    "name": "DIV",
    "operand": 3,
    "actions": [
        {
            "read": "r3",
            "write": "spb"
        }, {
            "read": "r2",
            "write": "spa"
        }, {
            "read": "alu_h",
            "write": "spc",
            "alu_op": "a/b",
            "alu_ena": "div",
        }, {
            "read": "alu_l",
            "write": "r1",
            "alu_op": "a/b",
            "alu_ena": "div",
            "jmp": "0"
        }
    ]
}, {
    "name": "MOV",
    "operand": 2,
    "actions": [
        {
            "read": "r2",
            "write": "r1",
            "jmp": "0"
        }
    ]
```

```
    }, {
        "name": "HALT",
        "operand": 0,
        "actions": [
            {
                "G": "1"
            }
        ]
    }, {
        "name": "NOP",
        "operand": 0,
        "actions": [
            {
                # WAIT
                "jmp": "0"
            }
        ]
    }
]

U_BUS_CODES = [
    "jmp",  # 22-23
    "blank",  # 18---21
    "G",  # 17
    "alu_ena",  # 14-16
    "alu_op",  # 8----13
    "write",  # 4--7
    "read",  # 0--3
]

REG_CODES = {
    "none": "0000",
    "r1": "0000",
    "r2": "0001",
    "r3": "0010",
    "pc": "0011",
    "mar": "0100",
    "mdr": "0101",
    "ir": "0110",
    "spa": "0111",
    "spb": "1000",
    "spc": "1001",
    "imm": "1010",
    "alu_l": "1011",
```

```
    "alu_h": "1100",
    "ram": "1111",
}

U_CODES = {
    "alu_ena": {
        "none": "000",
        "add": "001",
        "sh": "010",
        "cmp": "011",
        "div": "100",
        "mul": "101",
        "se": "110"
    },
    "alu_op": {
        "none": "000000",
        # add
        # CN S3 S2 S1 S0 M
        "a+1": "000000",
        "a+b": "110010",
        "a-b": "001100",
        "a&b": "010111",
        "a|b": "011101",
        "a^b": "001101",
        "a nxor b": "010011",
        # sh
        "sh_left": "000001",
        "sh_right": "000010",
        # cmp
        "a<b": "000001",
        "a<=b": "000011",
        "a=b": "000010",
        # se
        "=0": "000001",
        # mul
        "a*b": "000010",  # 未用
        # "a*b_l": "000001",
        # div
        "a/b": "000010",  # 未用
        # "a mod b": "000001",
    },
    "jmp": {
        "none": "00",
        "if": "01",
```

```
        "force": "10",
        "0": "11"
    },
    "read": REG_CODES,
    "write": REG_CODES,
    "blank": {
        "none": "0000"
    },
    "G": {
        "none": "0",
        "1": "1"
    }
}
```

2. 测试程序机器码

测试程序 1:

地址	标记	类型	内容	字节码
0		指令	MOV AX, .ad1	'1000111000111000
1		指令		'0000000000010000
2		指令	LD BX, AX	'0111001001000000
3		指令	ADD BX, BX, AX	'0010100001001000
4		指令	MOV CX, 0	'1000111010111000
5		指令		'0000000000000000
6		指令	ADD AX, AX, 1	'0010100000000111
7		指令		'0000000000000001
8	.loop	指令	LD DX, AX	'0111001011000000
9		指令	ADD CX, CX, DX	'0010100010010011
10		指令	XOR DX, AX, BX	'0100100011000001
11		指令	JNZ DX, .loop	'0101100011111000
12		指令		'0000000000000110
13		指令	STR CX, .ad2	'0111100010111000
14		指令		'0000000000010110
15		指令	HALT	'1001000000000000
16	.ad1	数据	3	'0000000000000011
17		数据	101	'0000000001100101
18		数据	102	'0000000001100110
19		数据	103	'0000000001100111
20		数据	104	'0000000001101000
21		数据	105	'0000000001101001
22	.ad2	数据	0	'0000000000000000

测试程序 2:

地址	标记	类型	内容	字节码

0		指令	MOV AX, 15	'1000111000111000
1		指令		'0000000000001111
2		指令	MOV BX, 3	'1000111001111000
3		指令		'0000000000000011
4		指令	MOV CX, .esp	'1000111010111000
5		指令		'0000000000111101
6		指令	ADD DX, AX, BX	'0010100011000001
7		指令	STR DX, CX	'0111100011010000
8		指令	ADD CX, CX, 1	'0010100010010111
9		指令		'0000000000000001
10		指令	SUB DX, AX, BX	'0011000011000001
11		指令	STR DX, CX	'0111100011010000
12		指令	ADD CX, CX, 1	'0010100010010111
13		指令		'0000000000000001
14		指令	SLLV DX, AX, BX	'0001000011000001
15		指令	STR DX, CX	'0111100011010000
16		指令	ADD CX, CX, 1	'0010100010010111
17		指令		'0000000000000001
18		指令	SRLV DX, AX, BX	'0001100011000001
19		指令	STR DX, CX	'0111100011010000
20		指令	ADD CX, CX, 1	'0010100010010111
21		指令		'0000000000000001
22		指令	SLTU DX, AX, BX	'0010000011000001
23		指令	STR DX, CX	'0111100011010000
24		指令	ADD CX, CX, 1	'0010100010010111
25		指令		'0000000000000001
26		指令	AND DX, AX, BX	'0011100011000001
27		指令	STR DX, CX	'0111100011010000
28		指令	ADD CX, CX, 1	'0010100010010111
29		指令		'0000000000000001
30		指令	OR DX, AX, BX	'0100000011000001
31		指令	STR DX, CX	'0111100011010000
32		指令	ADD CX, CX, 1	'0010100010010111
33		指令		'0000000000000001
34		指令	XOR DX, AX, BX	'0100100011000001
35		指令	STR DX, CX	'0111100011010000
36		指令	ADD CX, CX, 1	'0010100010010111
37		指令		'0000000000000001
38		指令	NXOR DX, AX, BX	'0101000011000001
39		指令	STR DX, CX	'0111100011010000
40		指令	ADD CX, CX, 1	'0010100010010111
41		指令		'0000000000000001
42		指令	MUL DX, AX, BX	'0111111011000001
43		指令	STR DX, CX	'0111100011010000
44		指令	ADD CX, CX, 1	'0010100010010111

45		指令		'0000000000000001
46		指令	STR spc, CX	'0111100110010000
47		指令	ADD CX, CX, 1	'0010100010010111
48		指令		'0000000000000001
49		指令	DIV DX, AX, BX	'1000011011000001
50		指令	STR DX, CX	'0111100011010000
51		指令	ADD CX, CX, 1	'0010100010010111
52		指令		'0000000000000001
53		指令	STR spc, CX	'0111100110010000
54		指令	ADD CX, CX, 1	'0010100010010111
55		指令		'0000000000000001
56		指令	JNZR 0, AX	'0110000111000000
57		指令		'0000000000000000
58		指令	JMPR -2	'0110110111000000
59		指令		'1111111111111110
60		数据	0	'0000000000000000
61	.esp	数据	0	'0000000000000000
62		数据	0	'0000000000000000
63		数据	0	'0000000000000000
64		数据	0	'0000000000000000
65		数据	0	'0000000000000000
66		数据	0	'0000000000000000
67		数据	0	'0000000000000000
68		数据	0	'0000000000000000
69		数据	0	'0000000000000000
70		数据	0	'0000000000000000
71		数据	0	'0000000000000000