实验报告

实验名称：存储器与控制器实验

院系：计算机学院 19 级计算机科学与技术

班级： 计科 2 班

姓名： 施天予

指导老师：陈刚

一、 实验内容

1. 从实验 5 实验中，导入 Display、clk_div 模块
2. 创建 PC 模块
3. 创建 main_decde, alu_decode 模块
4. 创建 Controller，调用 main_decode, alu_decode （逻辑参照表 2 和表 4）
5. 使用 Block Memory，导入 coe 文件（根据实验 6）
6. 自定义顶层文件，连接相关模块

二、实验原理

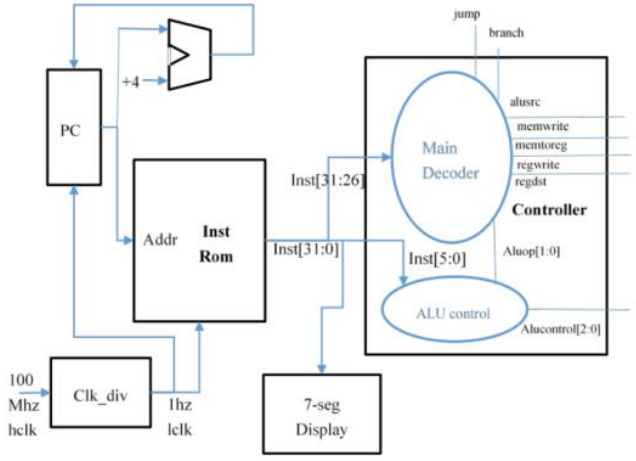


图 1

1. PC。 D 触发器结构，用于储存 PC(一个周期)。需实现 2 个输入，分别为 clk, rst, 分别连接时钟和复位信号；需实现 2 个输出，分别为 pc, inst_ce, 分别连接指令存储器的 addra, ena 端口。其中 addra 位数依据 coe 文件中指令数定义；
2. 加法器。用于计算下一条指令地址，需实现 2 个输入，1 个输出，输入值分别为当前指令地址 PC、32'h4；
3. Controller。其中包含两部分：

(a). main_decoder。负责判断指令类型，并生成相应的控制信号。需实现 1 个输入，为指令 inst 的高 6 位 op，输出分为 2 部分，控制信号有多个，可作为多个输出，也作为一个多位输出，具体参照 4.3 进行设计；aluop 传输至 alu_decoder，使 alu_decoder 配合 inst 低 6 位 funct，进行 ALU 模块控制信号的译码。

(b). alu_decoder。负责 ALU 模块控制信号的译码。需实现 2 个输入，1 个输出，输入分别为 funct, aluop；输出位 alucontrol 信号。

(c). 除上述两个组件，需设计 controller 模块顶层文件调用两个 decoder，对应实现 op,funct 输入信号，并传入调用模块；对应实现控制信号及 alucontrol，并连接至调用模块相应端口。

4. 指令存储器。使用 Block Memory Generator IP 构造。 (参考实验 5)
5. 时钟分频器。将板载 100Mhz 频率降低为 1hz，连接 PC、指令存储器时钟信号 clk。

注意：板子上只有 16 位的 segment 显示，所以只接入低 16 位进入显示（和实验 6 一样）

各控制信号含义

信号	含义
memtoreg	回写的数据来自于 ALU 计算的结果/存储器读取的数据
memwrite	是否需要写数据存储器
pcsrc	下一个 PC 值是 PC+4/跳转的新地址
alusrc	送入 ALU B 端口的值是立即数的 32 位扩展/寄存器堆读取的值
regdst	写入寄存器堆的地址是 rt 还是 rd,0 为 rt,1 为 rd
regwrite	是否需要写寄存器堆
branch	是否为 branch 指令,且满足 branch 的条件
jump	是否为 jump 指令
alucontrol	ALU 控制信号,代表不同的运算类型

三、实验设计

Controller 输出信号与 led 管脚对应关系如下表:

memtoreg	memwrite	pcsrc	alusrc	regdst	regwrite	jump	branch	alucontrol
[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[0:0]	[2:0]
led[0]	led[1]	led[2]	led[3]	led[4]	led[5]	led[6]	led[7]	led[8:10]

按照上表可以分配 led 管脚，其他管脚分配与之前相同

约束文件如下：

```
set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clk]

#7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]
set_property PACKAGE_PIN W6 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U8 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V5 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]

```

```
set_property PACKAGE_PIN U2 [get_ports {ans[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[0]}]
set_property PACKAGE_PIN U4 [get_ports {ans[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[1]}]
set_property PACKAGE_PIN V4 [get_ports {ans[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[2]}]
set_property PACKAGE_PIN W4 [get_ports {ans[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {ans[3]}]
##Buttons
set_property PACKAGE_PIN U18 [get_ports rst]
        set_property IOSTANDARD LVCMOS33 [get_ports rst]
## leds
set_property PACKAGE_PIN U16 [get_ports {memtoreg}]
    set_property IOSTANDARD LVCMOS33 [get_ports {memtoreg}]
set_property PACKAGE_PIN E19 [get_ports {memwrite}]
    set_property IOSTANDARD LVCMOS33 [get_ports {memwrite}]
set_property PACKAGE_PIN U19 [get_ports {pcsrc}]
    set_property IOSTANDARD LVCMOS33 [get_ports {pcsrc}]
set_property PACKAGE_PIN V19 [get_ports {alusrc}]
    set_property IOSTANDARD LVCMOS33 [get_ports {alusrc}]
set_property PACKAGE_PIN W18 [get_ports {regdst}]
    set_property IOSTANDARD LVCMOS33 [get_ports {regdst}]
set_property PACKAGE_PIN U15 [get_ports {regwrite}]
    set_property IOSTANDARD LVCMOS33 [get_ports {regwrite}]
set_property PACKAGE_PIN U14 [get_ports {jump}]
    set_property IOSTANDARD LVCMOS33 [get_ports {jump}]
set_property PACKAGE_PIN V14 [get_ports {branch}]
    set_property IOSTANDARD LVCMOS33 [get_ports {branch}]
set_property PACKAGE_PIN V13 [get_ports {alucontrol[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {alucontrol[0]}]
set_property PACKAGE_PIN V3 [get_ports {alucontrol[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {alucontrol[1]}]
set_property PACKAGE_PIN W3 [get_ports {alucontrol[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {alucontrol[2]}]

```

Ins_Rom 例化

```
ENTITY Ins_Rom IS
  PORT (
    clka : IN STD_LOGIC;
    addra : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    douta : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
END Ins_Rom;
```

display.v

```
`timescale 1ns / 1ps

module display(
    input wire clk,reset,
    input wire [31:0]s,
    output wire [6:0]seg,
    output reg [3:0]ans
    );
    reg [20:0]count;
    reg [4:0]digit;
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;

    always @(posedge clk)
    case(count[20:19])
        0:begin
            ans = 4'b1110;
            digit = s[3:0];
        end

        1:begin
            ans = 4'b1101;
            digit = s[7:4];
        end

        2:begin
            ans = 4'b1011;
            digit =s[11:8];
        end

        3:begin
            ans = 4'b0111;
            digit = s[15:12];
        end
    endcase

    seg7 U4(.din(digit),.dout(seg));
endmodule
```

seg7.v

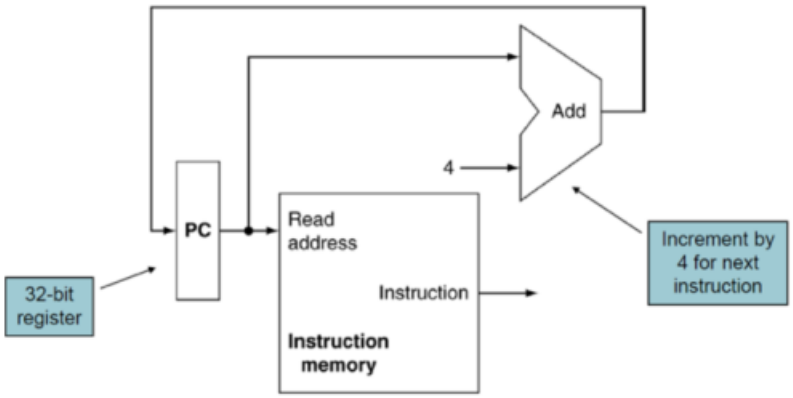
```
`timescale 1ns / 1ps

module seg7(
    input wire [4:0]din,
    output reg [6:0]dout
    );

    always@(*)
    case(din)
        5'h0:dout = 7'b000_0001;
        5'h1:dout = 7'b100_1111;
        5'h2:dout = 7'b001_0010;
        5'h3:dout = 7'b000_0110;
        5'h4:dout = 7'b100_1100;

        5'h5:dout = 7'b010_0100;
        5'h6:dout = 7'b010_0000;
        5'h7:dout = 7'b000_1111;
        5'h8:dout = 7'b000_0000;
        5'h9:dout = 7'b000_0100;
        5'ha:dout = 7'b000_1000;
        5'hb:dout = 7'b110_0000;
        5'hc:dout = 7'b011_0001;
        5'hd:dout = 7'b100_0010;
        5'he:dout = 7'b011_0000;
        5'hf:dout = 7'b011_1000;
        default:dout = 7'b111_1111;
    endcase
endmodule
```

pc.v



因为设置 ROM 核时宽度是 32 位，所以这里 addr+1 而不是 addr+4

```
`timescale 1ns / 1ps

module pc(
    input  clk,rst,
    output reg[7:0]addr
    );
    initial addr <= 0;//初始化地址为0
    always@(posedge clk or negedge rst)
        begin
            if (rst) addr <= 0;  //如果reset地址跳到0
            else addr <= addr + 1;  // 跳转到下一指令
        end
endmodule
```

clk_div.v

将 100Mhz 降低到 1hz

（设置 27 位寄存器 count 的分频 clk 刷新约为 1 秒，老师的 32 位寄存器 count 的分频 clk 刷新时间过长）

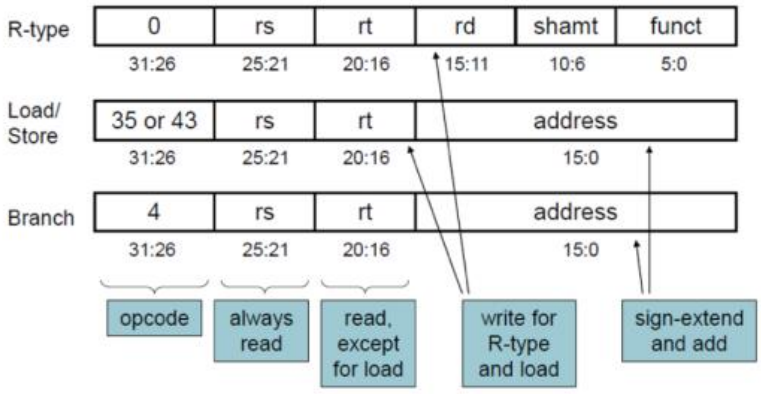
```
`timescale 1ns / 1ps

module clk_div(
    input wire clk,rst,
    output reg clk1
    );
    reg [26:0]count;
    initial clk1 <= 0;
    always@(posedge clk or negedge rst)
        if(rst) begin
            count <= 0;
            clk1 <= 0;
        end
        else begin
            if(count[26]) begin
                count <= 0;
                clk1 <= ~clk1;
            end
            else
                count <= count + 1;
        end
endmodule

```

controller.v

指令译码原理



根据指令的高 6 位得到 op，低 6 位得到 funct，还有一个 ALU 的零输出（这里默认为 1）作为控制器的输入，得到一系列控制信号

```
1  `timescale 1ns / 1ps
2
3  module controller(
4      input wire[5:0] op,
5      input wire[5:0] funct,
6      input wire zero,
7      output wire memtoreg,memwrite,
8      output wire pcsrc,alusrc,
9      output wire regdst,regwrite,
10     output wire jump,branch,
11     output wire[2:0] alucontrol
12     );
13     wire [1:0] aluop;
14
15     maindec md(op,memtoreg,memwrite,branch,alusrc,regdst,regwrite,jump,aluop);
16     aludec ad(funct,aluop,alucontrol);
17
18     assign pcsrc = branch & zero;
19 endmodule
20
```

maindec.v

instruction	op5:0	regwrite	regdst	alusrc	branch	memWrite	memtoReg	aluop1:0
R-type	000000	1	1	0	0	0	0	10
lw	100011	1	0	1	0	0	1	00
sw	101011	0	X	1	0	1	X	00
beq	000100	0	X	0	1	0	X	01
addi	001000	1	0	1	0	0	0	00
j	000010	0	X	X	X	0	X	XX

根据参考信号表，可以设计出该模块代码

```
`timescale 1ns / 1ps

module maindec(
    input wire[5:0] op,
    output wire memtoreg,memwrite,
    output wire branch,alusrc,
    output wire regdst,regwrite,
    output wire jump,
    output wire[1:0] aluop
    );
    reg[8:0] controls;
    assign {regwrite,regdst,alusrc,branch,memwrite,memtoreg,jump,aluop} = controls;
    always @(*) begin
       case(op)
           6'b000000:controls <= 9'b110000010;//R-TYPE
           6'b100011:controls <= 9'b101001000;//LW
           6'b101011:controls <= 9'b001010000;//SW
           6'b000100:controls <= 9'b000100001;//BEQ
           6'b001000:controls <= 9'b101000000;//ADDI
           6'b000010:controls <= 9'b000000100;//J
           default  :controls <= 9'b000000000;//illegal op
       endcase
    end
endmodule
```

aludec.v

opcode	aluop	operation	funct	alu function	alu control
lw	00	Load word	XXXXXX	Add	010
sw	00	Store word	XXXXXX	Add	010
beq	01	Branch equal	XXXXXX	Subtact	110
R-type	10	Add	100000	qdd	010
		subtract	100010	Subtract	110
		and	100100	And	000
		or	100101	Or	001
		set-on-less-than	101010	SLT	111

根据上表，用 funct 和 aluop 可得出 alucontrol

```
`timescale 1ns / 1ps

module aludec(
    input wire[5:0] funct,
    input wire[1:0] aluop,
    output reg[2:0] alucontrol
    );
    always @(*) begin
       case(aluop)
           2'b00 : alucontrol <= 3'b010;//add (for lw/sw/addi/j)
           2'b01 : alucontrol <= 3'b110;//sub (for beq)
           default : case (funct)
               6'b100000:alucontrol <= 3'b010;//add
               6'b100010:alucontrol <= 3'b110;//sub
               6'b100100:alucontrol <= 3'b000;//and
               6'b100101:alucontrol <= 3'b001;//or
               6'b101010:alucontrol <= 3'b111;//slt
               default:  alucontrol <= 3'b000;
           endcase
       endcase

    end
endmodule
```

top.v（顶层文件，连接各模块）

```
`timescale 1ns / 1ps

module top(
    input clk,rst,
    output [3:0] ans, //select for seg
    output [6:0] seg, //segment digital
    output wire memtoreg,memwrite,
    output wire pcsrc,alusrc,
    output wire regdst,regwrite,
    output wire jump,branch,
    output wire [2:0] alucontrol
    );
    wire [7:0] addr;
    wire clk1;//分频后的clk
    wire [31:0] inst;//32位指令
    wire [15:0] inst1;
    wire [5:0] op;
    wire [5:0] funct;
    wire zero;

```

```
    assign inst1 = inst[15:0];//指令低16位，用于display
    assign op = inst[31:26];//op取指令高6位
    assign funct = inst[5:0];//funct取指令低6位
    assign zero = 1;//该实验没有ALU的零输出，默认设为1

    clk_div cd(.clk(clk),.rst(rst),.clk1(clk1));
    pc p(.clk(clk1),.rst(rst),.addr(addr));
    Ins_Rom rom(.clka(clk1),.addra(addr),.douta(inst));
    controller con(
        .op(op),
        .funct(funct),
        .zero(zero),
        .memtoreg(memtoreg),
        .memwrite(memwrite),
        .pcsrc(pcsrc),
        .alusrc(alusrc),
        .regdst(regdst),
        .regwrite(regwrite),
        .jump(jump),
        .branch(branch),
        .alucontrol(alucontrol)
    );
    display U2(.clk(clk),.reset(rst),.s(inst1),.ans(ans),.seg(seg));
endmodule
```

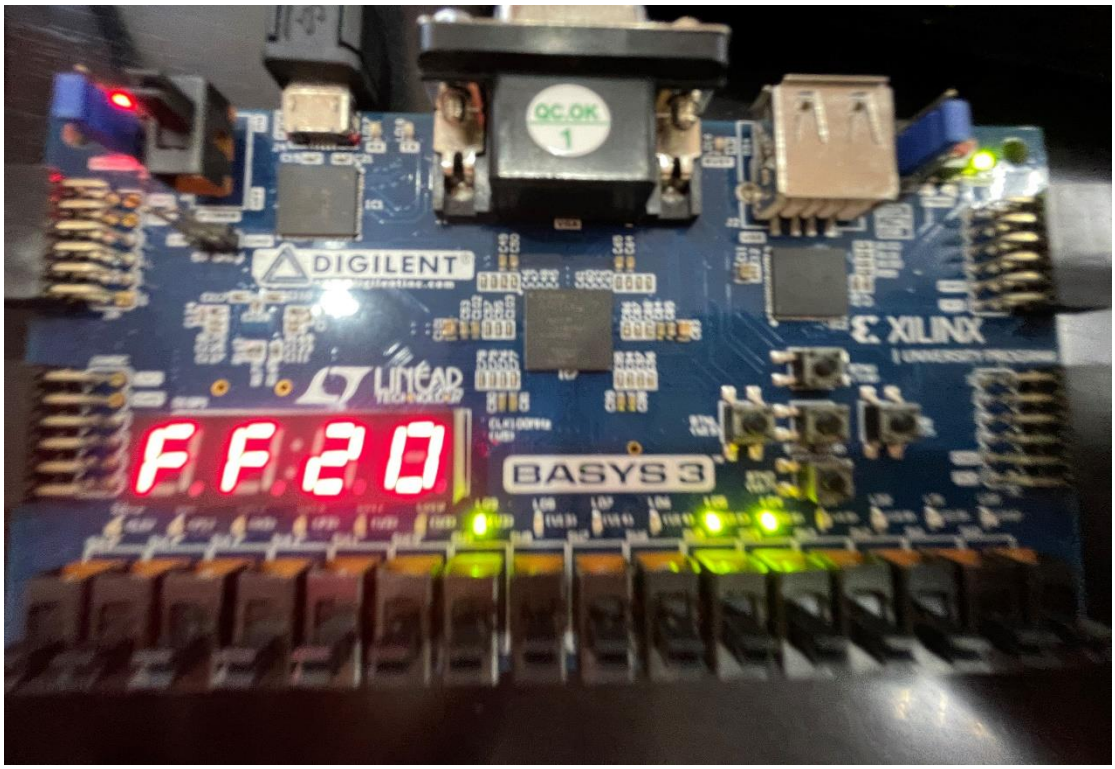
四、实验结果与分析

设置 Rom 初值的文件 prgmip32.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
0001ff20,
003cf022,
8c140018,
ac180008,
1017000c,
20160010,
08150014
```

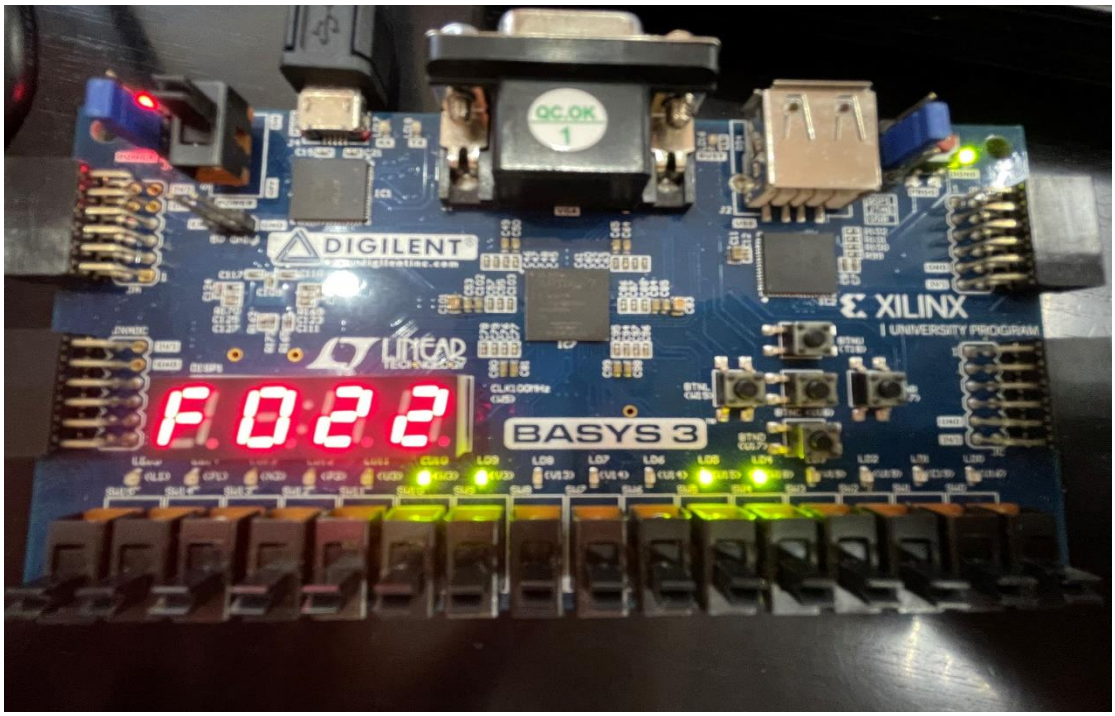
0001ff20　　op=6’b000000 funct=6’b100000 执行 add 指令

alucontrol=010 regwrite=1 regdst=1



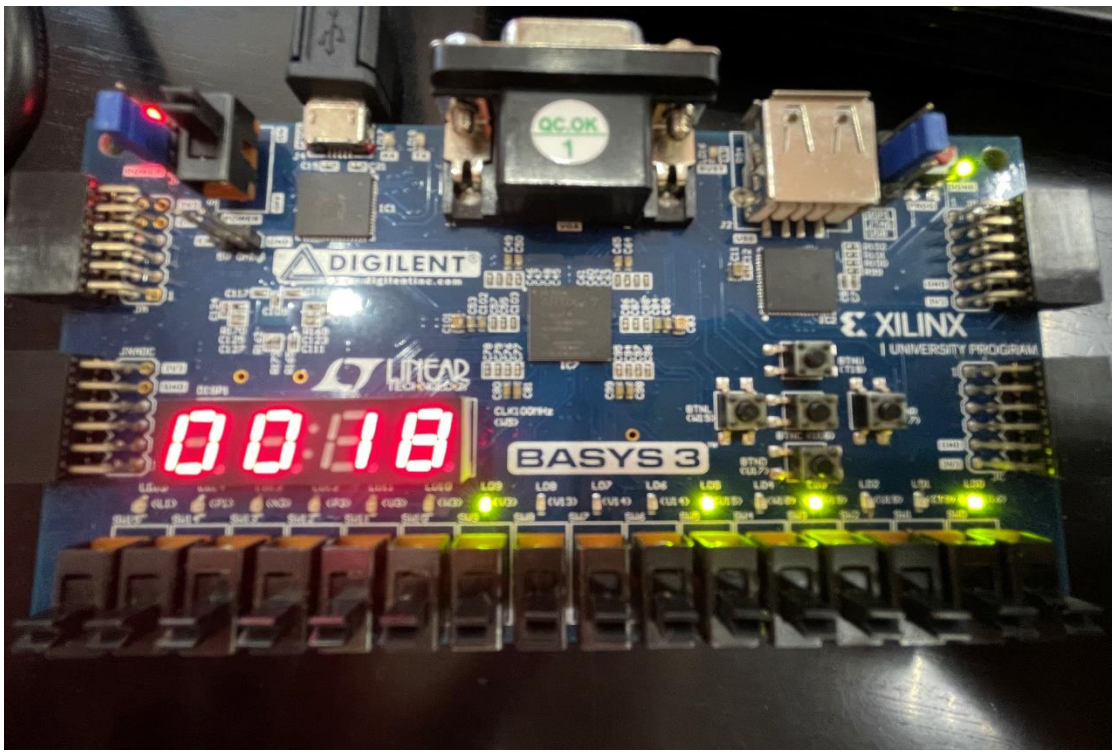
003cf022　　op=6’b000000 funct=6’b100010 执行 sub 指令

alucontrol=110 regwrite=1 regdst=1



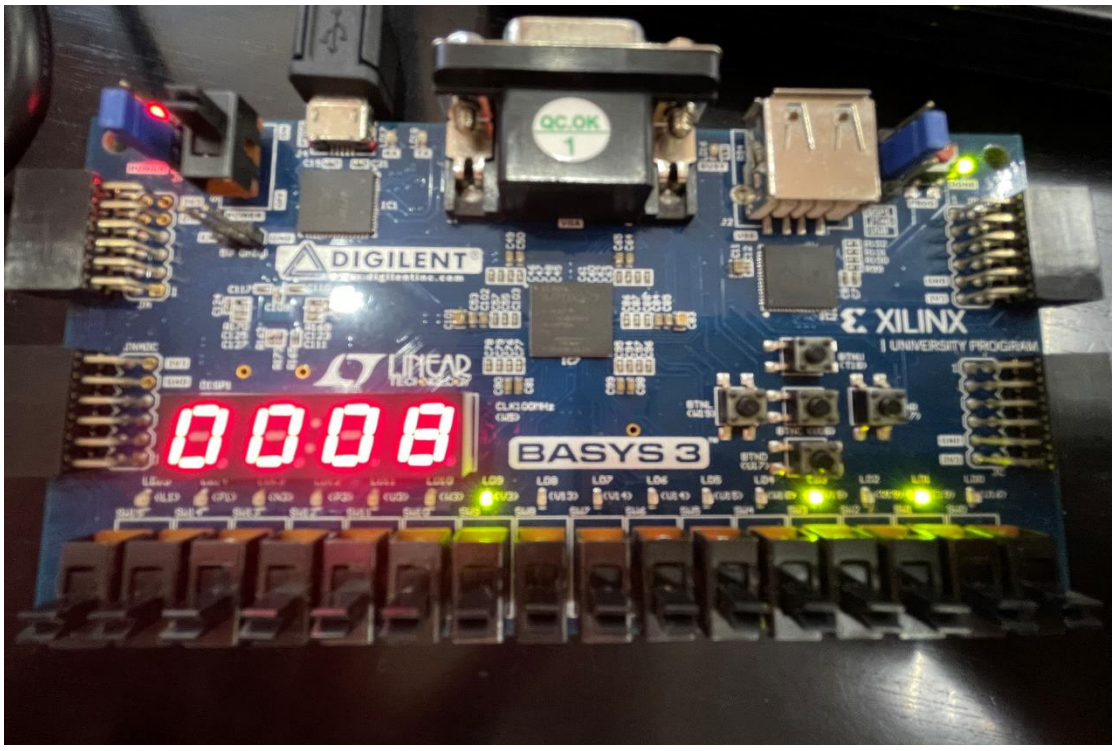
8c140018 op=6’b100011 执行 lw 指令

alucontrol=010 regwrite=1 alusrc=1 memtoreg=1



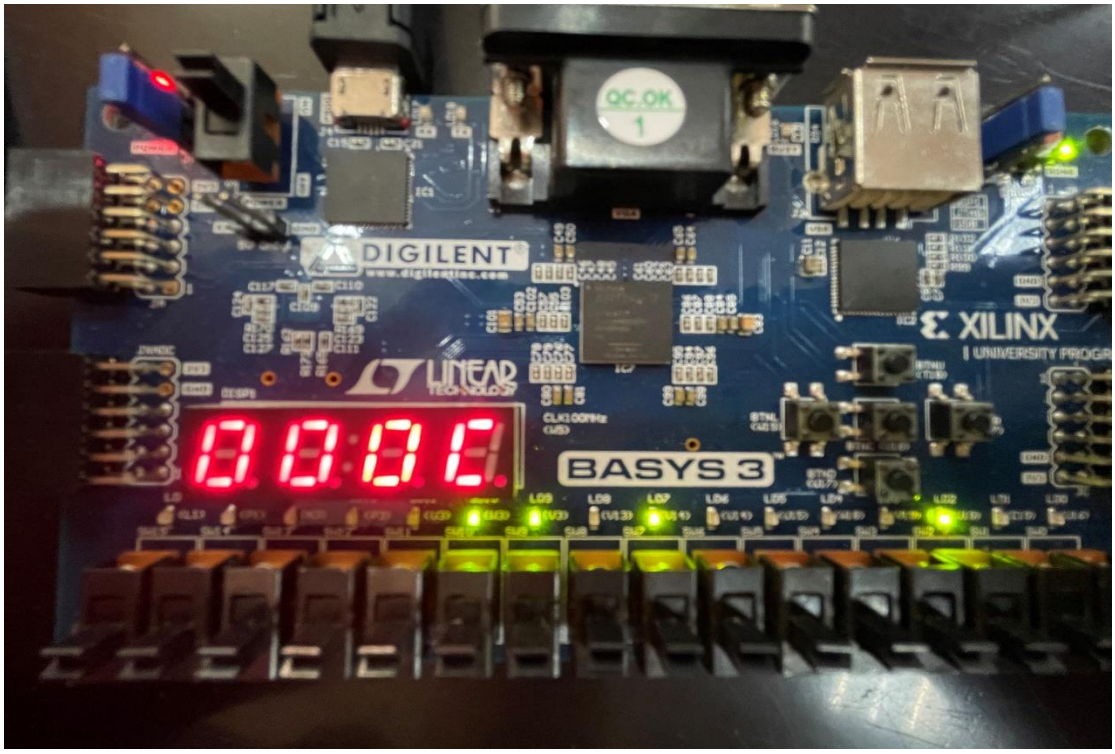
ac180008 op=6’b101011 执行 sw 指令

alucontrol=010 alusrc=1 memwrite=1



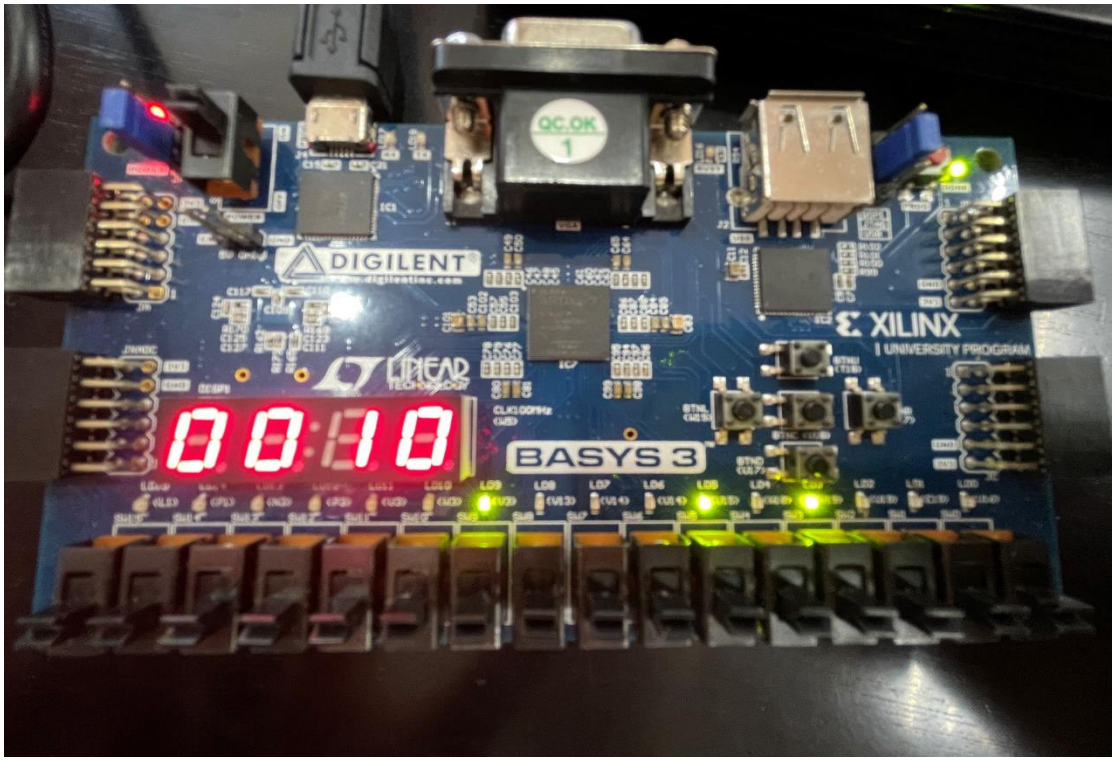
1017000c　op=6’b000100 执行 beq 指令

alucontrol=110 branch=1 pcsrc=1



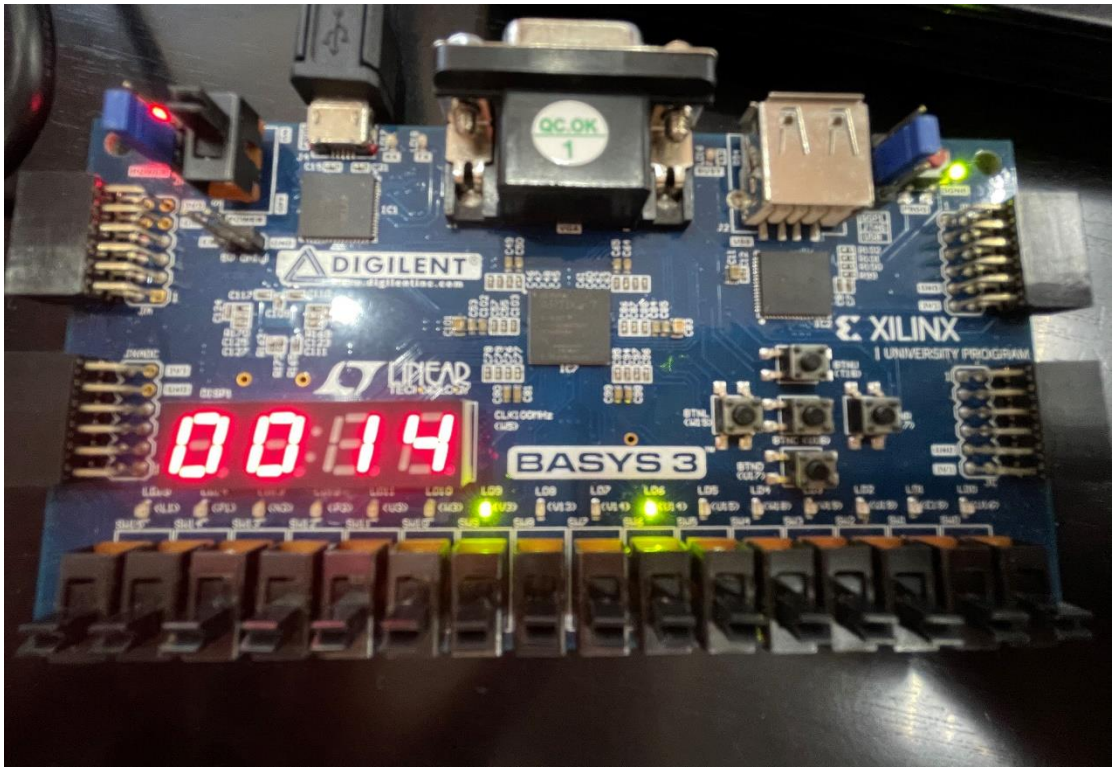
20160010　op=6’b001000 执行 addi 指令

alucontrol=010 regwrite=1 alusrc=1



08150014　op=6’b000010 执行 j 指令

alucontrol=010 jump=1



五、实验总结

这次存储器与控制器的实验在上次实验的基础上加上了 clk 分频，pc 以及控制器模块。在本次实验中，我对数据通路、控制器的概念更加熟悉了，对一个 CPU 的取指、译码阶段如何执行有了深入的了解，并且掌握了单周期 CPU 控制器的工作原理及设计方法，各控制信号的作用和生成过程，以及执行指令的过程。总之，我在这次实验中学到了很多，希望下次设计完整单周期 CPU 的实验我也能更熟练地“搭积木”，在此实验基础上迎刃而解。