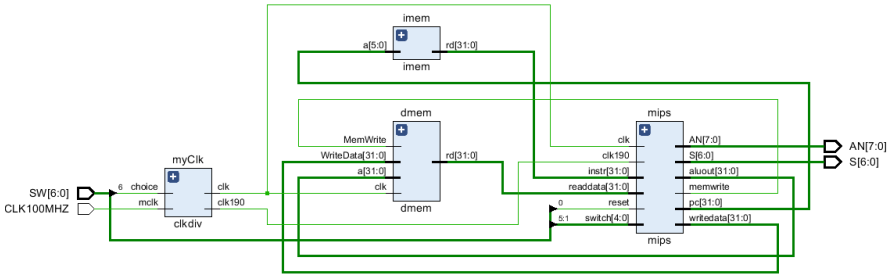
MIPS 单周期实验报告

罗翔

17307130191

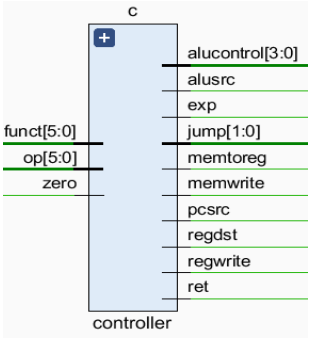
一． 各部件详解

本次 MIPS 单周期的总体框架如下：



mips 为 CPU 核心，包括控制单元(controller)与数据通路(datapath)；imem 为指令存储器,为 CPU 提供指令；dmem 为数据存储器,为 CPU 提供数据存储空间；myClk 为时钟分频器，其中 clk 为 CPU 主体的时钟信号，clk190 为七段码显示的扫描时钟信号。

1. Control Unit



功能说明：

解析指令，根据不同指令设定不同的控制线。需要提的是，branch 为两根线分别对应的 beq 和 bne，jump 也有两根线分别对应的 jump 和 jal。

设计思路：

控制码真值表如下：

操作	类型	op	funct	regwrite	regdst	alusrc	exp	branch	memwrite	memtoreg	aluop	jump
add	R-Type	000000	100000	1	1	0	0	00	0	0	010	00
sub	R-Type	000000	100010	1	1	0	0	00	0	0	010	00
and	R-Type	000000	100100	1	1	0	0	00	0	0	010	00
or	R-Type	000000	100101	1	1	0	0	00	0	0	010	00
slt	R-Type	000000	101010	1	1	0	0	00	0	0	010	00
addi	I-Type	001000		1	0	1	1	00	0	0	000	00

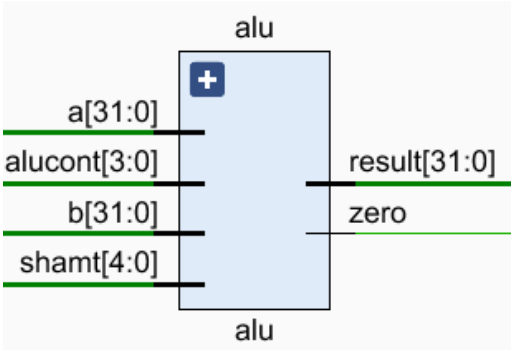
andi	I-Type	001100		1	0	1	0	00	0	0	011	00
ori	I-Type	001101		1	0	1	0	00	0	0	100	00
slti	I-Type	001010		1	0	1	1	00	0	0	101	00
sw	I-Type	101011		0	0	1	1	00	1	0	000	00
lw	I-Type	100011		1	0	1	1	00	0	1	000	00
j	J-Type	000010		0	x	x	x	xx	0	x	xxx	10
nop	R-Type	000000	000000	0	0	0	0	00	0	0	000	00
beq	I-Type	000100		0	0	0	0	10	0	0	001	00
bne	I-Type	000101		0	0	0	0	01	0	0	001	00
jal	J-Type	000011		0	x	x	x	xx	0	x	xxx	11
sll	R-Type	000000	000000	1	1	0	0	00	0	0	010	00
srl	R-Type	000000	000010	1	1	0	0	00	0	0	010	00
sra	R-Type	000000	000011	1	1	0	0	00	0	0	010	00
jr	R-Type	000000	001000	1	1	0	0	00	0	0	010	00

通过将控制线并列、case 语句分类的方式，可以十分简练的完成对不同类型指令控制码的确定。

```
assign {regwrite, regdst, alusrc, exp, branch,
        memwrite, memtoreg, aluop, jump} = controls;

always @ (*)
    case(op)
        6'b000000: controls <= 13'b1100000001000; //Rtype
        6'b100011: controls <= 13'b1011000100000; //LW
```

2. ALU

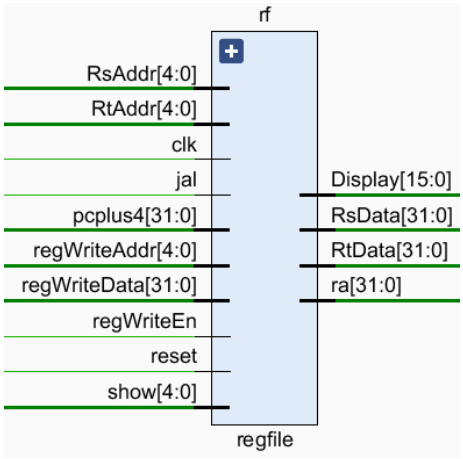


ALU 是 CPU 中的运算部件，可以支持加减、移位和位运算等操作。其中 ALU 的运算结果主要由{ alucont[3], alucont[1:0] }决定，

```
case({alucont[3],alucont[1:0]})
    3'b000:   result <= c0; // a & b
    3'b001:   result <= c1; // a | b
    3'b010:   result <= c2; // add sub
    3'b011:   result <= c3; // slt
    3'b100:   result <= c4; // sll
    3'b101:   result <= c5; // srl
    3'b110:   result <= c6; // sra
    default: result <= c0;
```

而 alucont[2]决定是否对 b 取反，以及作算术运算时是否有进位 carry。其中 shamt 是移位数，zero 是零符号位。

3. RegisterFile



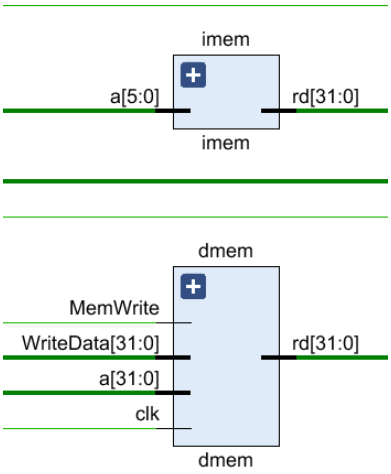
功能说明：

为满足 CPU 高速运行的寄存器，置于芯片内部，通常由快速的静态随机读写存储器实现。且由于造价高昂，所以需要控制可存储空间。

主要用于从寄存器中读写数据，其中读过程为组合逻辑；而写过程为时序逻辑，必须在下一个时钟周期到来时才能写入。

jal 是 jump and link 的标志位。jal 为 1，PC 会跳转至目标地址并且将 PC+4 存储在$ra(31)中，jal 与 jr 配合可以实现 call（调用函数），再加上$sp 可以实现栈（stack）操作。向外输出$ra(31)是为了在执行 jr 指令时，告知 PC 下一步执行的指令的地址。

4. Memory

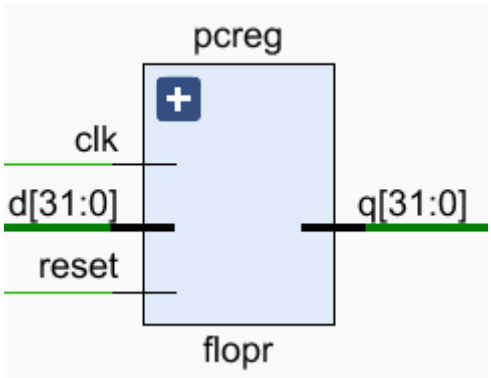


功能说明：

dmem 为数据存储器，主要是从存储器中读写数据信息，其中读过程可视作组合逻辑，而写过程为时序逻辑。实际运用中从存储器中读写数据的速度远远慢于从寄存器中读写数据，但是存储器的存储空间非常大。

imem 为指令存储器。因为 MIPS 存储器模型是字节寻址且指令编码则是 32 位的一个字，所以每条指令的地址都是 4 的倍数，因此每次取值可以直接将 PC 前 30 位（即除以 4 后）传入 imem。

5. PC



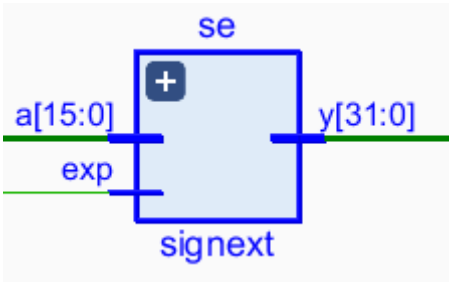
功能说明：

用于指示下一条指令的地址，即在时钟信号到来时更新下一条指令的 PC 地址，使 CPU 执行下一条指令。

PC 一般为顺序执行，即下一条指令的地址一般为 PC + ‘1’。但也存在转移执行的情况，比如分支指令 beq、bne 和跳转指令 j。

其中分支指令需要将给定的 16 位（目标指令与现有指令的偏移量）进行符号扩展加上现有地址得到。而跳转指令则是将给定的 26 位伪地址头部加上现有指令地址的前 4 位后左移 2 位得到实际地址。{PC [31:28], instr [25:0], 2'b00}

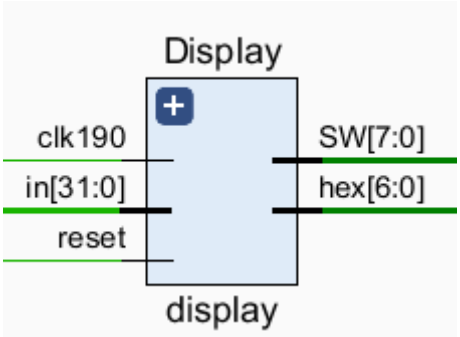
6. ImmExtend



功能说明：

由于 I 型指令的立即数为 16 位，而实际运算时为 32 位，所以需要对其进行扩展，其中 I 型指令中的逻辑运算应为零扩展，而其他 I 型指令均为符号扩展。所以需要一个控制位决定扩展方式。

7. Display



功能说明：

因为需要将观察的内容显示在七段数码管上，所以需要一个 Display 的元件。并且考虑到要同时在 8 个七段数码管上显示内容，需要新增一个更快的时钟信号来扫描，利用人眼的视觉限制造成 8 个数码管同时显示的假象。

8. Datapath

功能说明：

数据通路通过实例化各元件，将整个 CPU 串接起来。

```
display Display(clk190, {pc[15:0], Real}, S, AN, reset);
flopr #(32) pcreg(clk, reset, pcnext, pc);
adder pcadd1(pc, 32'b100, pcplus4);
sl2 immsh(signimm, signimmsh);
adder pcadd2(pcplus4, signimmsh, pcbranch);
mux2 #(32) pcbrmux(pcplus4, pcbranch, pcsrc, pcnextbr);
mux2 #(32) pcmux(pcnextbr, {pcplus4[31:28], instr[25:0], 2'b00}, jump[1], pcnextj);
mux2 #(32) pcjrmux(pcnextj, ra, ret, pcnext);
regfile rf(clk, regwrite, switch, instr[25:21], instr[20:16], writereg, result, srca, writedata, Real, reset, pcplus4, jump[0], ra);
mux2 #(5) wrmux(instr[20:16], instr[15:11], regdst, writereg);
mux2 #(32) resmux(aluout, readdata, memtoreg, result);
signext se(exp, instr[15:0], signimm);
```

二． 展示

1. 使用部件

七段数码管 * 8、 开关 * 7

2. 部件分配

（1） 开关 1： reset，使 pc、regfile、memory 复位。

（2） 开关 2 – 6： 选择显示的 register。

（3） 开关 7： 速度选择器（时钟周期 T = 0.16s / 1.32s）。

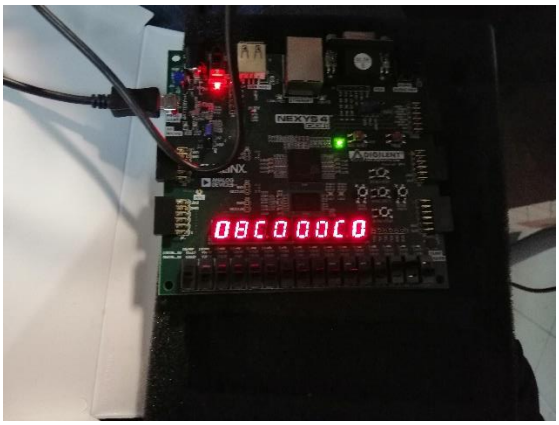
（4） 数码管 1 – 4： 显示当前选定的 register 内容。

（5） 数码管 5 – 8： 显示当前 pc 值。

三． 遇到的问题及解决方法

七段数码管显示问题：

因为要同时在 8 个数码管上显示内容，故需要扫描 8 个数码管。考虑人的眼睛的反应周期在 0.05s 左右。最开始我通过分频设置的扫描周期为 q[18](0.005s)，此时七段数码管闪烁明显，考虑到应该是扫描周期过长，使得肉眼观测到了扫描过程，我又将扫描周期置为 q[16](0.00125s)，则七段数码管正常显示，没有明显的闪烁。但是在制作过程中，我还遇见了本该正常显示的 pc 和 register，在七段数码管上集体向左偏移一位的情况，如下图。后发现，我的 Display 元件 always 语句中全为非阻塞赋值，因此每次被扫描的数码管显示的都是上一个数码管应该显示的内容（即右侧数码管本该显示的内容），故造成了整体左移的情况。



四．添加指令及模拟测试

首先测试基本功能，使用书上的代码进行随机测试：

具体指令如下：

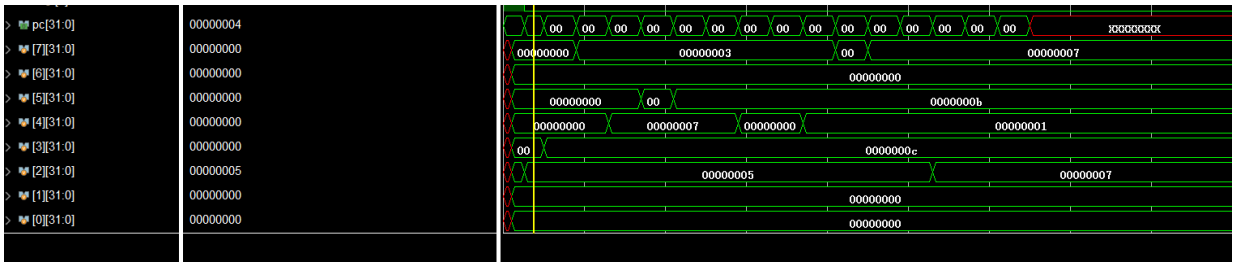
Assembly	Description	Address	Machine
addi $2, $0, 5	# initialize $2 = 5	0	20020005
addi $3, $0, 12	# initialize $3 = 12	4	2003000c
addi $7, $3, -9	# initialize $7 = 3	8	2067fff7
or $4, $7, $2	# $4 = (3 OR 5) = 7	c	00e22025
and $5, $3, $4	# $5 = (12 AND 7) = 4	10	00642824
add $5, $5, $4	# $5 = 4 + 7 = 11	14	00a42820
beq $5, $7, end	# shouldn't be taken	18	10a7000a
slt $4, $3, $4	# $4 = 12 < 7 = 0	1c	0064202a
beq $4, $0, around	# should be taken	20	10800001
addi $5, $0, 0	# shouldn't happen	24	20050000
slt $4, $7, $2	# $4 = 3 < 5 = 1	28	00e2202a
add $7, $4, $5	# $7 = 1 + 11 = 12	2c	00853820
sub $7, $7, $2	# $7 = 12 - 5 = 7	30	00e23822
sw $7, 68($3)	# [80] = 7	34	ac670044
lw $2, 80($0)	# $2 = [80] = 7	38	8c020050
j end	# should be taken	3c	08000011
addi $2, $0, 1	# shouldn't happen	40	20020001
sw $2, 84($0)	# write mem[84] = 7	44	ac020054

其中$2: 5 → 7

$4: 7 → 0 → 1

$5: 4 → 11

$7: 3 → 12 → 7



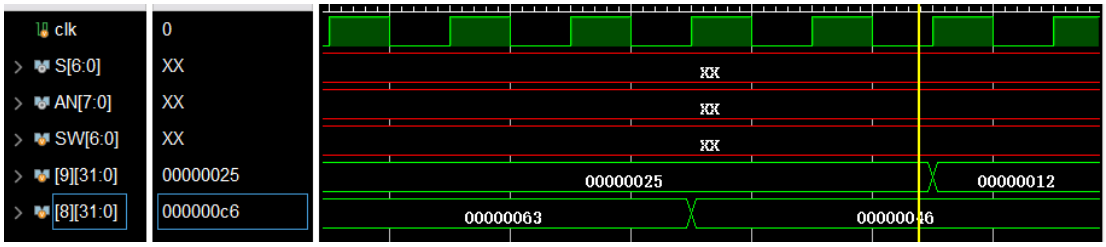
接下来测试添加的 srl、sll、beq、j 指令，使用的是张作柏同学在 github 上的代码:

其中指令如下

```
 0x0 : addi $t0, $0, 99     | 20080063
 0x4 : addi $t1, $0, 37     | 20090025
 0x8 : addi $s0, $0, 0      | 20100000
 0xc :                      |
 0xc : while:               |
 0xc : beq $t1, $0, done    | 10090006
0x10 : andi $t2, $t1, 1     | 312a0001
0x14 : beq $t2, $0, target  | 100a0001
0x18 : add $s0, $s0, $t0    | 02088020
0x1c : target:              |
0x1c : sll $t0, $t0, 1      | 00084040
0x20 : srl $t1, $t1, 1      | 00094842
0x24 : j while              | 08000003
0x28 :                      |
0x28 : done:                |
```

$t0($8): 99 → 198(0x63 → 0xc6) t0 逻辑左移 1 位

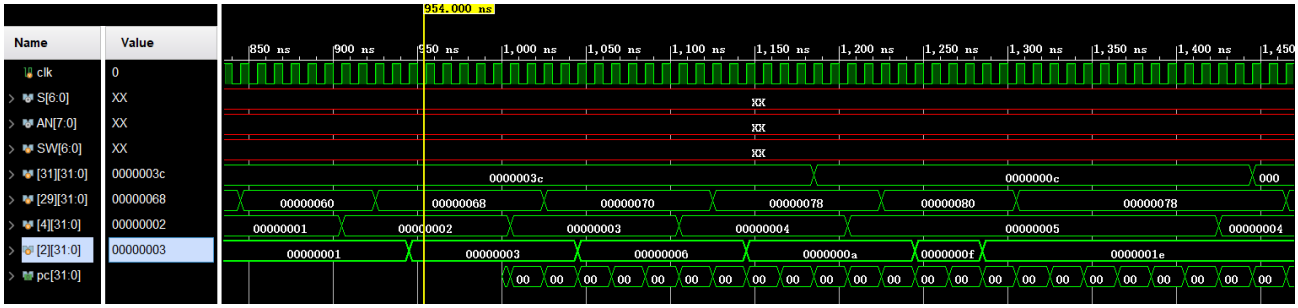
$t1($9): 37 → 18(0x25 → 0x12) t1 逻辑右移 1 位



接下来测试添加的 jal、jr 指令，使用的同样也是张作柏同学在 GitHub 上的累加代码：

```
 0x0 : addi $sp, $0, 128     | 201d0080
 0x4 : addi $a0, $0, 5       | 20040005
 0x8 : jal factorial         | 0c000004
 0xc : sll $v0, $v0, 1       | 00021040
0x10 :                       |
0x10 : factorial:            |
0x10 : addi $sp, $sp, -8     | 23bdfff8
0x14 : sw $a0, 4($sp)        | afa40004
0x18 : sw $ra, 0($sp)        | afbf0000
0x1c : addi $t0, $0, 2       | 20080002
0x20 : slt $t0, $a0, $t0     | 0088402a
0x24 : beq $t0, $0 ,else     | 10080003
0x28 : addi $v0, $0, 1       | 20020001
0x2c : addi $sp, $sp, 8      | 23bd0008
0x30 : jr $ra                | 03e00008
0x34 : else:                 |
0x34 : addi $a0, $a0, -1     | 2084ffff
0x38 : jal factorial         | 0c000004
0x3c : lw $ra, 0($sp)        | 8fbf0000
0x40 : lw $a0, 4($sp)        | 8fa40004
0x44 : addi $sp, $sp, 8      | 23bd0008
0x48 : add $v0, $a0, $v0     | 00821020
0x4c : jr $ra                | 03e00008
```

$v0($2) 1 → 3 → 6 → 0xa → 0xf



五．参考文献

1. 戴维 • 莫尼 • 哈里斯，莎拉 L. 哈里斯. 数字设计和计算机体系结构（原书第 2 版）

2. https://github.com/Oxer11/MIPS