Lab5 简单的类 MIPS 单周期处理器的实现 - 整体调试

实验目的

- 理解简单的类 MIPS 单周期处理器的工作原理(即几类基本指令执行时所需的数据通路和与之对应的控制线路及其各功能部件间的互联定义、逻辑选择关系)
- 完成简单的类 MIPS 单周期处理器
  - 9 条 MIPS 指令(lw, sw, beq, add, sub, and, or, slt, j) CPU 的实现与调试
  - 拓展至 16 条指令(增加 addi, andi, ori, sll, srl, jal, jr) CPU 的设计与实现
- 仿真测试

实验原理

主控制单元模块

- 本次实验用到的控制信号有：
  - RegDst：目标寄存器的选择信号。（0：写入rt代表的寄存器，1：写入rd代表的寄存器）
  - ALUSrc：ALU的第二个操作数的来源。（0：使用rt寄存器中的数，1：使用立即数）
  - MemToReg：写寄存器的数据来源。（0：使用ALU的结果，1：使用从内存读取的数据）
  - RegWrite：写寄存器使能信号。（高电平表示当前指令需要写寄存器）
  - MemRead：读内存使能信号。（高电平表示当前指令需要读内存，比如load）
  - MemWrite：写内存使能信号。（高电平表示当前指令需要写内存，比如store）
  - Branch：条件跳转信号。（高电平表示当前指令是条件跳转指令，比如branch）
  - extSign：符号扩展信号，高电平说明当前指令需要对操作数进行符号扩展。
  - JalSign：jal指令信号，高电平说明当前指令是jal指令。
  - ALUOp：发送给ALU控制器，用来进一步解析运算类型的控制信号。
  - Jump：无条件跳转信号。（高电平表示当前指令是无条件跳转指令，比如jump）
- 本次实验译码的指令的操作码如下：

指令	opCode
R型：add，sub，and，or，sll，srl，slt，jr	000000
lw	100011
sw	101011
beq	000100
addi	001000
andi	001100
ori	001101
j	000010
jal	000011

- 不同操作码对应的控制信号：

	000000	100011	101011	000100	001000	001100	001101	000010	000011
RegDst	1	0	0	0	0	0	0	0	0
ALUSrc	0	1	1	0	1	1	1	0	0
MemToReg	0	1	0	0	0	0	0	0	0
RegWrite	1	1	0	0	1	1	1	0	1
MemRead	0	1	0	0	0	0	0	0	0
MemWrite	0	0	1	0	0	0	0	0	0
Branch	0	0	0	1	0	0	0	0	0
ExtSign	0	1	1	1	1	0	0	0	0
JalSign	0	0	0	0	0	0	0	0	1
ALUOp	101	000	000	001	010	011	100	110	110
jump	0	0	0	0	0	0	0	1	1

- 当出现不属于以上情况的操作码时，将所有控制信号置为0，视作空指令。

ALU控制单元模块

- 该单元进一步解析ALUOp信号。ALUOp信号的具体含义如下：

ALUOp	指令	实际动作
000	lw，sw	ALU执行加运算
001	beq	ALU执行减运算
010	addi	ALU执行加运算
011	andi	ALU执行逻辑与运算
100	ori	ALU执行逻辑或运算
101	R型指令	还需要根据指令的Funct区决定
110	j，jal	ALU不做动作

- 指令进一步解析时，还要产生两个控制信号：
  - JrSign：jr指令信号，高电平说明当前指令是jr指令。
  - ShamtSign：shamt选择信号，高电平说明操作数需从指令的shamt区域选取。
- ALU控制单元的输入输出对应关系如下：

	ALUOp	Funct	Operation Control
lw，sw	000	xxxxxx	0010
beq	001	xxxxxx	0110
addi	010	xxxxxx	0010
andi	011	xxxxxx	0000
ori	100	xxxxxx	0001
sll	101	000000	0011，ShamtSign = 1
srl	101	000010	0100，ShamtSign = 1
jr	101	001000	0101，JrSign = 1
add	101	100000	0010
sub	101	100010	0110
and	101	100100	0000
or	101	100101	0001
slt	101	101010	0111
j，jal	110	xxxxxx	0101

ALU模块

- Operation Control 和 ALU 执行运算类型的对应关系如下：

ALU Control input	op
0000	and
0001	or
0010	add
0011	sll
0100	srl
0101	not change
0110	sub
0111	slt

寄存器组模块

- 与Lab4相同。

数据存储器模块

- 与Lab4的存储器相同。

指令存储器模块

- 与数据存储器基本相同。
- 不同之处：
    - 读入初始指令后，后续不支持修改。
    - 根据输入的PC值取指令。

有符号扩展单元模块

- 与Lab4相同。

程序计数器模块

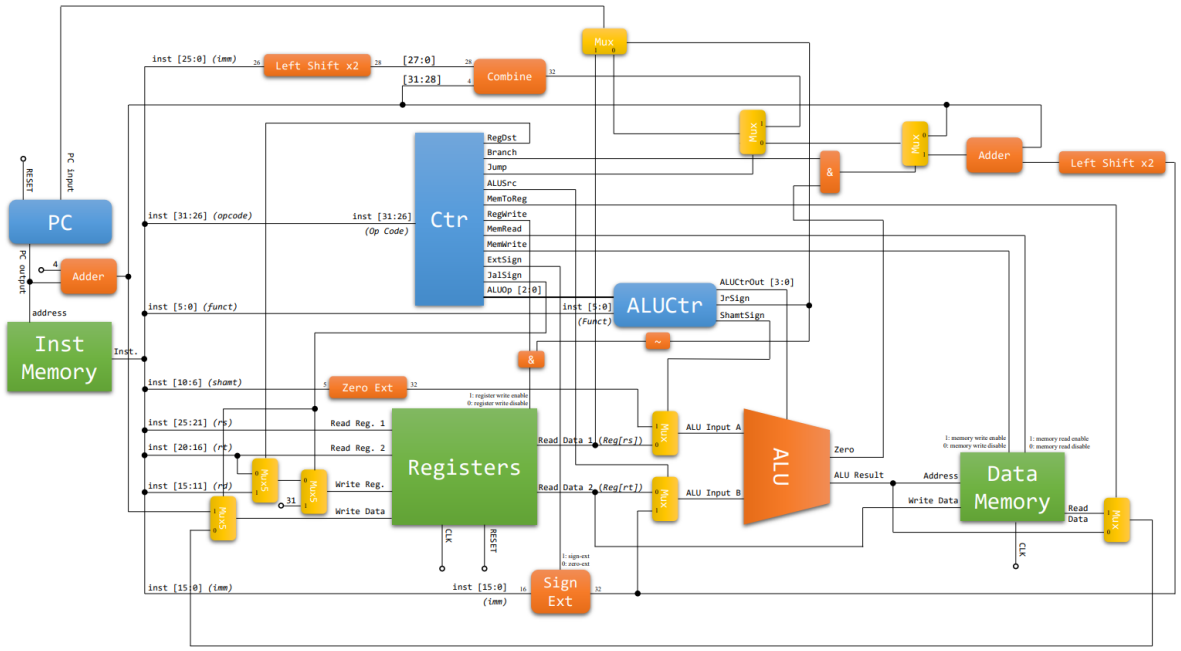
- 在时钟上升沿 —— 一个周期的开始 —— 将PC修改后输出。
- 如果接受到reset信号则归零。

数据选择器模块

- 根据输入的指定信号，选择两个输入数据通路中的一个作为输出。
- 分为两种数据选择器：
    - 输入输出均为5位数据通路。
    - 输入输出均为32位数据通路。

顶层模块

- 顶层模块由各个单元模块的实例化对象通过各种总线连接组装而成。整体组装后的单周期MIPS处理器电路设计图如下：【图源自网络】



实现细节

- Ctr.v
  - 对比Lab3，添加了新的信号和新的操作码分支。
- ALUCtr.v
  - 对比Lab3，添加了新的ALUOp分支。
- ALU.v
  - 对比Lab3，添加了新的运算类型分支。
- Registers.v
  - 与Lab4基本一致。
  - 添加信息输出代码，以用作调试。
- dataMemory.v
  - 对比Lab4，进行了一定的扩容。
- signext.v
  - 对比Lab4，加入了对ExtSign信号的判断。
- instMemory.v
  - 与dataMemory基本一致。
  - 需要对取值地址进行一些译码操作。
- PC.v
  - 在时钟上升沿根据输入修改PC后输出。
  - 若接收到reset信号则将PC归零。
  - 添加信息输出代码，以用作调试。
- Mux32.v
  - 根据控制信号决定将两个输入中的哪个作为输出。
- Mux5.v
  - 与Mux32相同，只有总线位数不一样。
- Top.v
  - 根据电路图创建各模块的实例化对象，创建恰当的总线进行连接。
- mem_data.dat
  - 在内存中放入了如下数据：

地址	数据	地址	数据	地址	数据	地址	数据
1	000000FF	9	00000004	17	00000008	25	000002FF
2	00000100	10	00000005	18	00000009	26	000003FF
3	00000101	11	00000006	19	0000000A	27	000004FF
4	00000102	12	00000007	20	00000107	28	000005FF
5	00000000	13	00000103	21	00000108	29	000006FF
6	00000001	14	00000104	22	00000109	30	000007FF
7	00000002	15	00000105	23	0000010A	31	000008FF
8	00000003	16	00000106	24	000001FF	32	000009FF

- mem_inst.dat
    - 用下列指令进行测试：

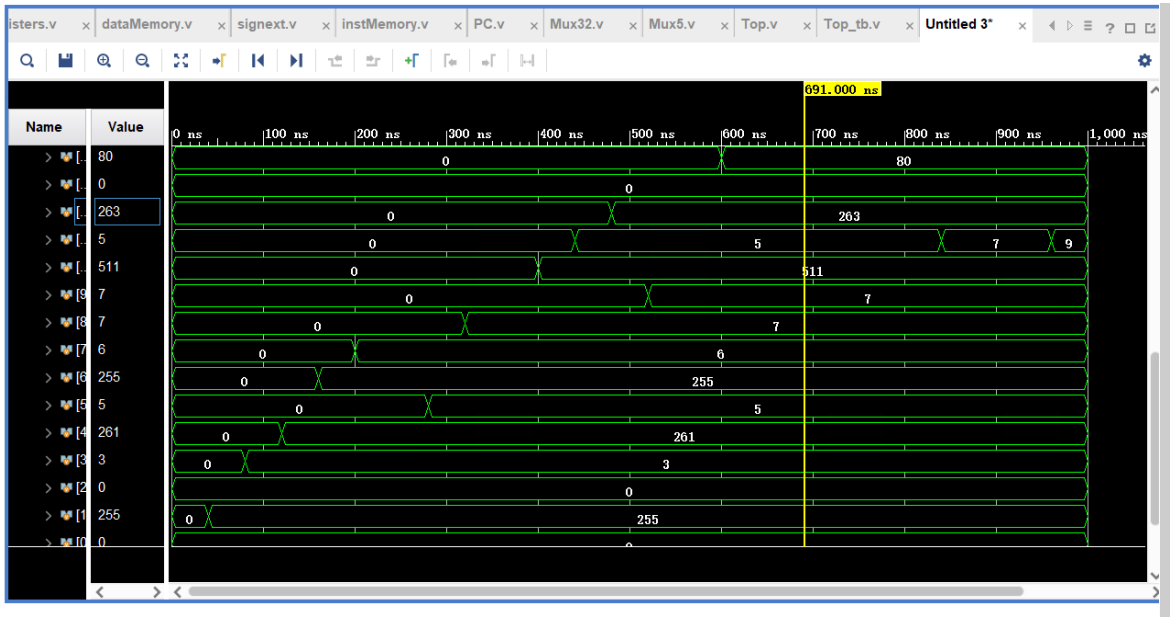
指令地址	指令	执行结果
0	lw $1, 0($0)	$1 = Mem[0] = 255
1	lw $2, 1($0)	$2 = Mem[1] = 256
2	lw $3, 7($0)	$3 = Mem[7] = 3
3	lw $4, 11($3)	$4 = Mem[14] = 261
4	add $6, $1, $2	$6 = 255 + 256 = 511
5	sub $7, $4, $1	$7 = 261 - 255 = 7
6	lw $16, 5($7)	$16 = Mem[11] = 7
7	and $5, $1, $4	$5 = 255 & 261 = 5
8	or $8, $16, $2	$8 = 7 | 256 = 263
9	sw $8, 7($5)	Mem[12] = 263
10	addi $10, $1, 256	$10 = 255 + 256 = 511
11	addi $11, $4, 255	$11 = 261 & 255 = 5
12	ori $12, $16, 256	$12 = 7 | 256 = 263
13	lw $9, 12($0)	$9 = Mem[12] = 263
14	beq $9, $8, 1	go to 16
15	sll $13, $11, 4	not executed
16	sll $14, $11, 4	$14 = $11 << 4 = 80
17	sll $15, $10, 4	$15 = $10 >> 4 = 31
18	slt $17, $15, $14	$17 = 1
19	slt $18, $15, $16	$18 = 0
20	j 22	go to 22
21	slt $18, $15, $14	not executed
22	jal 24	go to 24
23	beq $16, $11, 3	go to 27
24	addi $11, $11, 2	$11 = 5 + 2 = 7
25	jr $31	go to 23
26	addi $11, $11, 2	not executed
27	addi $11, $11, 2	$11 = 7 + 2 = 9

- Top_tb.v
    - 创建Top模块的实例化对象processor。
    - 从外部文件读入指令数据和内存数据。

- 生成周期为40ns的时钟信号，作为激励信号。

实验结果

- 通过在console输出信息，可以得到部分指令执行的结果，以此检查设计是否正确。
- 仿真结果：



总结与反思

- 复习了MIPS单周期处理器的数据通路，对各个指令需要用到的控制信号，各个指令在数据通路上的执行过程有了更加深刻的理解。
- 学到了Verilog的一些库函数，如用来读入数据的 read 和用来在 console 打印信息的display。
- 感谢学长学姐以及网络上的一些陌生人提供开源的学习资料、电路设计图、测试样例。
- 尽管在网上可以查到很多参考资料、电路设计图，但是将代码完篇依然是一件非常考验耐心和细心的事情。需要通过对一些具体指令在数据通路上运行的手动模拟，去将每条总线连在对应的地方，在这个过程中还需要不断检查，以免缺漏线路。