Lab2 32 位 MIPS 单周期处理器设计

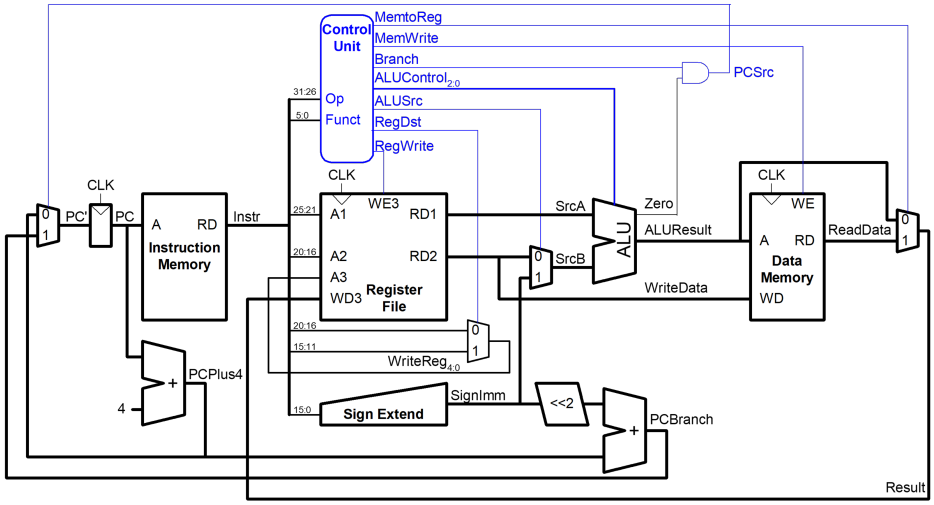
1 实验目的

熟悉 MIPS 单周期 CPU 的工作原理
熟悉 Verilog HDL 基本语法及其设计思想
熟悉 Vivado 仿真测试的使用

2 实验过程

2.1 学习 MIP 单周期处理器原理

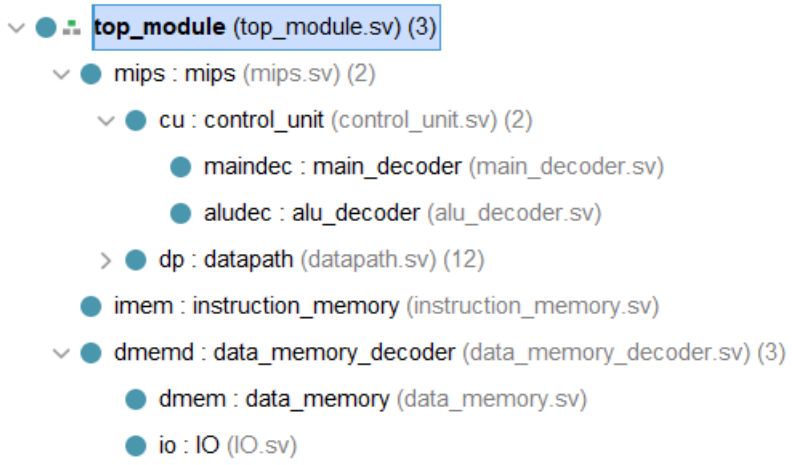
首先在阅读了教材（《计算机组成与体系结构（第三版）》）的第四、五章后，对于 MIPS 单周期处理器的工作原理我有了更深刻的理解。对于寄存器，ALU，译码器等模块的工作原理以及共同实现的效果有了更深刻的了解。并且对如下图所示的 MIPS 单周期处理器的工作原理以及使用它完成各种指令时各模块的工作方式有了较全面的了解。



2.2 完成 MIPS 单周期处理器设计

2.2.1 添加代码框架

根据给出的实验材料上 PPT 中的代码，写出代码的整体结构，如下图所示：



2.2.2 添加指令集

2.2.2.1 指令集概览

书中只给出了一部分 MIPS 指令的实现方法，但可以仿照书中的方法，编写出更多 MIPS 指令，从而满足更多汇编代码的需求。经过编写，目前本 MIP 单周期处理器实现的指令集如下：

指令	指令类型	功能
add	R	加
sub	R	减
add	R	按位与
or	R	按位或
slt	R	小于则置 1
lw	I	读内存
sw	I	写内存
addi	I	加立即数
andi	I	按位与立即数
ori	I	按位或立即数
beq	I	相等则跳转
j	J	跳转

2.2.2.2 指令集实现

指令集中多数指令都已经在书中给出了其具体实现方式，这里不再赘述。一些新增的指令并不需要对处理器的结构做出大修改，只需要调整译码器和 ALU，使其可以识别新的指令。

2.2.2.3 主译码器真值表

指令	opcode	RegWrite	RegDst	ALUSrc	Branch	MemWrite	MemtoReg	Jump	ALUop
R 类型	000000	1	1	0	0	0	0	0	111
lw	100011	1	0	1	0	0	1	0	000
sw	101011	0	0	1	0	1	0	0	000
addi	001000	1	0	1	0	0	0	0	000
andi	001100	1	0	1	0	0	0	0	011
ori	001101	1	0	1	0	0	0	0	100
beq	000100	0	0	0	1	0	0	0	001
j	000010	0	0	0	0	0	0	1	000

2.2.2.4 ALUOP 编码表

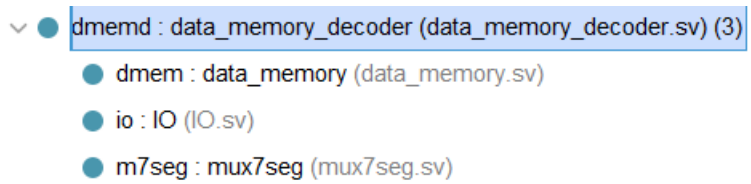
ALUOP	含义
000	加法
001	减法
011	按位与
100	按位或
111	根据 funct 决定
other	未使用

2.2.2.5 ALU 译码器真值表

ALUop	Funct	ALUControl
000	x	加
001	x	减
011	x	按位与
100	x	按位或
111	100000	加
111	100010	减
111	100100	按位与
111	100101	按位或
111	101010	小于则置 1
other	x	未使用

2.3 测试代码

为了便于测试，本 CPU 还实现了 IO 接口，可通过开关实时控制 CPU 中运算的数值。原理为在内存中高位设置一块接口区，其中包含两个操作数的值、一个结果的值和一个状态数，当访问到内存中的高位时，改为从接口区读取数值，同时在代码中通过判断状态数决定是否进行跳转，从而使输入可以即时反映到代码。具体则是通过扩展 DataMemory 实现。最后添加和 7 段数码管的衔接，从而将结果显示出来。结构图如下图所示：$data_memory_decoder$：将下列三个模块衔接起来；



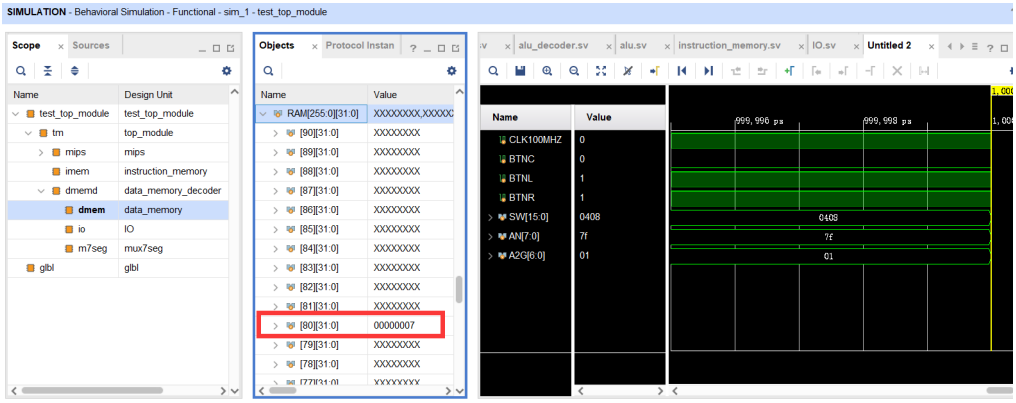
$data_memory$：修改前的 DataMemory，扩展了内存用来进行 IO 接口；

IO：判断状态数，存储三个数的数值，并将数值传输给下一个模块；

$mux7seg$：用 7 段数码管显示三个数；

接下来先进行实验材料中 PPT 上的 MIPS 测试代码的仿真，测试结果如下图：

可以看到在运行结束后内存中的 80 位存入了 7，与代码设计相符。



然后进行关于 IO 接口的 MIPS 测试代码的仿真，其中 IO 接口的 MIPS 测试代码如下：

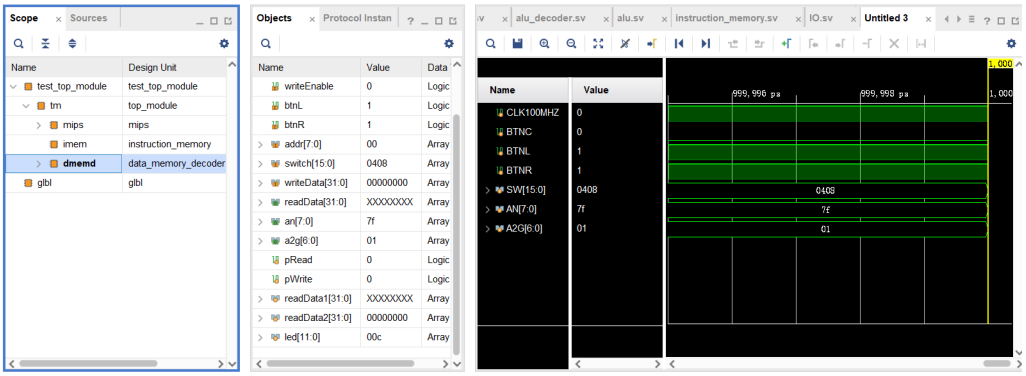
```
main:
    addi $s0, $0, 0
    sw   $s0, 0x80($0)
chkSwitch:
    lw   $s1, 0x80($0)
    andi $s2, $s1, 0x2
    beq  $s2, $0, chkSwitch
    lw   $s3, 0x88($0)
    lw   $s4, 0x8C($0)
    add  $s5, $s4, $s3
chkLED:
    lw   $s1, 0x80($0)
    andi $s2, $s1, 0x1
    beq  $s2, $0, chkLED
    sw   $s5, 0x84($0)
    j    chkSwitch
```

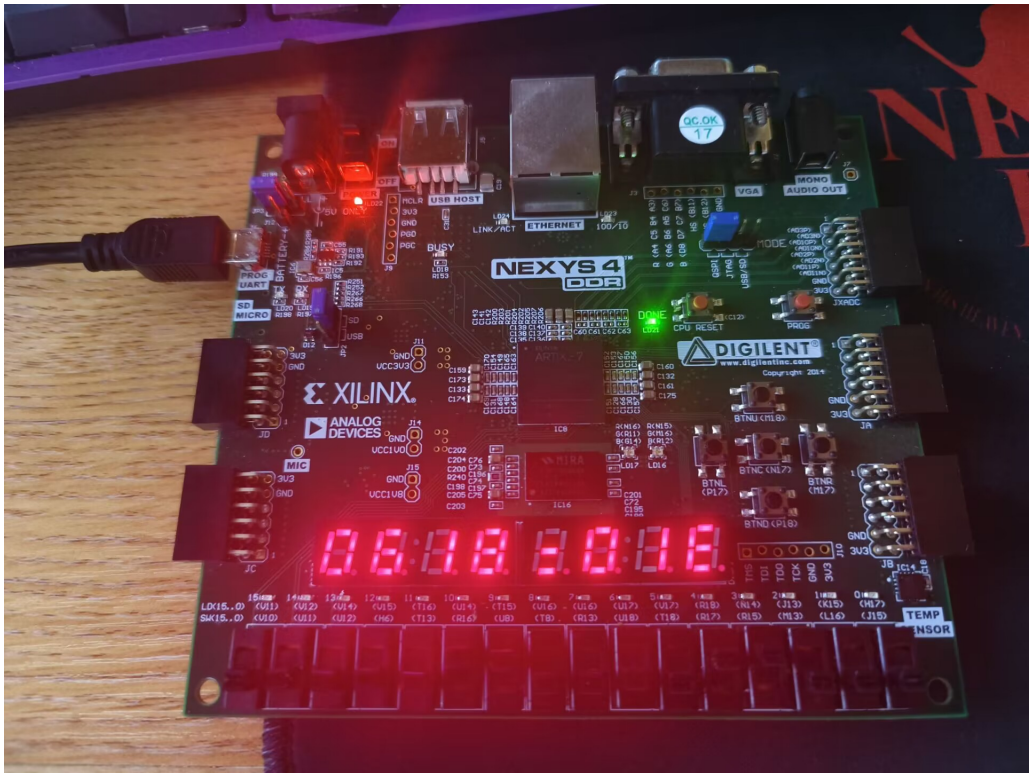
```
20100000
ac100080
8c110080
32320002
1240fffd
8c130088
8c14008c
0293a820
8c110080
32320001
1240fffd
ac150084
08000002
```

测试结果如下图：

可以看到在运行结束后 led 变量存入了 0000000c，即十进制的 12，与代码中应存入的 4+8 的结果相符。

最后进行实机测试，测试结果如下图：


Scope
Sources
Name
Design Unit
test_top_module
test_top_module
tm
top_module
mips
mips
imem
instruction_memory
dmemd
data_memory_decoder
glbl
glbl
Objects
Protocol Instan
Name
Value
writeEnable
0
btnL
1
btnR
1
addr[7:0]
00
switch[15:0]
0408
writeData[31:0]
00000000
readData[31:0]
XXXXXXXX
an[7:0]
7f
a2g[6:0]
01
pRead
0
pWrite
0
readData1[31:0]
XXXXXXXX
readData2[31:0]
00000000
led[11:0]
00c
alu_decoder.sv
alu.sv
instruction_memory.sv
IO.sv
Untitled 3
Name
Value
CLK100MHZ
0
BTNC
0
BTNL
1
BTNR
1
SW[15:0]
0408
AN[7:0]
7f
A2G[6:0]
01
999,998 ps
999,998 ps
0408
7f
01


NEXYS 4 DDR
DIGILENT
XILINX
ANALOG DEVICES
ETHERNET
USB HOST
VGA
MONO AUDIO OUT
CPU RESET
TEMP SENSOR

3 实现体会

在本次实验过程中我有很多的收获。首先在阅读教材之后，使我更加熟悉了 MIPS 单周期处理器的寄存器，加法器，左移单位，符号扩展单元，alu，可复位触发器和复用器等模块的工作原理以及其在 MIPS 处理器设计时的代码设计思路。在使用 vivado 的过程中，我学习到了软件的相关操作，主要是对项目以及模块的创建过程以及仿真测试，上板测试的操作过程，尤其是使用仿真对代码进行调试，使用下来发现非常方便，令我受益匪浅。