计算机体系结构实验 2：32 位单周期处理器设计

15307110200 廖文

一、实验要求

1. 仔细阅读**教材** 7.3 节(P231-239)内容
2. 参考**教材** 7.6 节(P270-279)**代码**，完成 MIPS 单周期处理器设计
3. 用**教材**图 7-60(P276)测试代码测试上述设计
4. 增加 ori、bne **等**指令，修改上述 MIPS 单周期处理器：
   a) 修改 MIPS 处理器原理图(cf 图 7-14)
   b) 修改主译码器(cf 表 7-3)、ALU 译码器真值表(cf 表 7-2)
      修改 Verilog 代码
5. 编写 MIPS 汇编测试代码，测试修改后的 MIPS 处理器

二、MIPS 处理器的设计

1. 教材上给出了单周期 MIPS 处理器的设计图如下

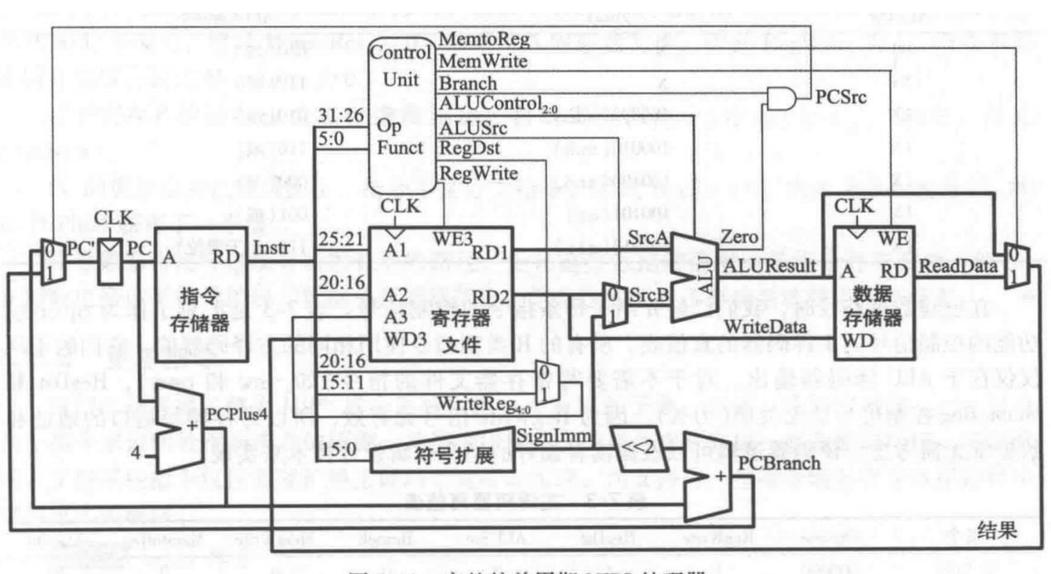
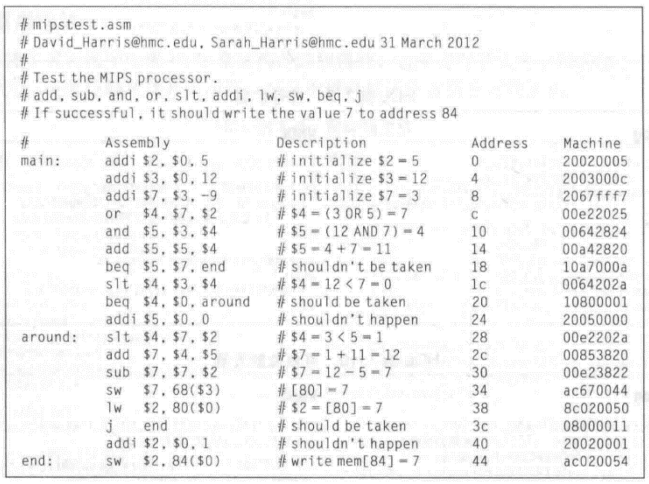


图 7-11 完整的单周期 MIPS 处理器

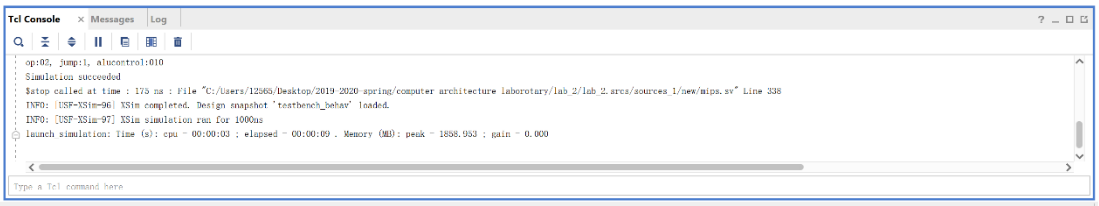
2. 根据教材所给的处理器代码和测试代码测试上述设计。
   其中支持的指令包括
   R 型算术/逻辑指令：and, sub, and, or, slt
   I 型算术/逻辑指令：addi
   存储器指令：lw, sw
   分支指令：beq，j

   测试代码如下右图所示。



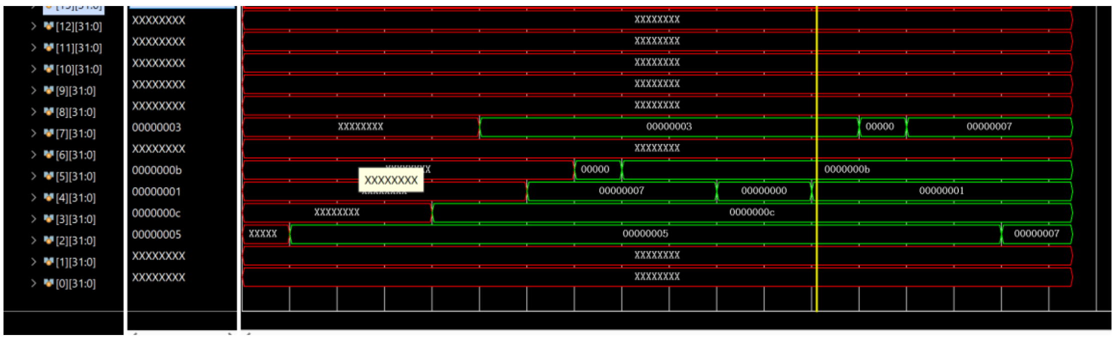
```
# mipstest.asm
# David_Harris@hmc.edu, Sarah_Harris@hmc.edu 31 March 2012
#
# Test the MIPS processor.
# add, sub, and, or, slt, addi, lw, sw, beq, j
# If successful, it should write the value 7 to address 84

#          Assembly              Description             Address   Machine
main:      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
around:    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
end:       sw   $2, 84($0)       # write mem[84] = 7     44        ac020054
```

从命令行的输出可以看到，程序成功执行。



```
op:02, jump:1, alucontrol:010
Simulation succeeded
$stop called at time : 175 ns : File "C:/Users/12565/Desktop/2019-2020-spring/computer architecture laborotary/lab_2/lab_2.srcs/sources_1/new/mips.sv" Line 338
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:03 ; elapsed = 00:00:09 . Memory (MB): peak = 1858.953 ; gain = 0.000
```

为了更加直观的展示测试程序执行期间处理器的状态，我们在模拟的结果中输出寄存器的值如下。



三、增加指令

考虑增加 ori 和 bne 指令。首先在教材附录 1 中找到这两条指令的描述。

001101(13)	ori rt, rs, imm	立即数或	[rt] = [rs] | ZeroImm
000101(5)	bne rs, rt, label	如果不相等则转移	if ([rs] ! = [rt]) PC = BTA

已经支持的两条相似的指令为

001000(8)	addi rt, rs, imm	立即数加法	[rt] = [rs] + SignImm
000100(4)	beq rs, rt, label	如果相等则转移	if ([rs] == [rt]) PC = BTA

ori 指令在 addi 指令的基础上进行修改即可，不需要增加额外的部件和标志位。
bne 指令在 beq 指令的基础上进行修改，我们增加了一个标志位 bne，标识该分支指令在连个寄存器不相同的时候进行跳转。修改后的部件的代码如下。
主译码器的真值表与 ALU 译码器真值表这里没有画出，对真值表的逻辑的修改见代码中背景为黄色的部分。

```
//decide the alu control bit given the funct field and aluop field of
the instruction
//combinational logic
module aludec(
    input logic [5:0] funct,
    input logic [1:0] aluop,
    output logic [2:0] alucontrol);

    always_comb
        case(aluop)
            2'b00: alucontrol <= 3'b010; // add (for lw/sw/addi) end;
```

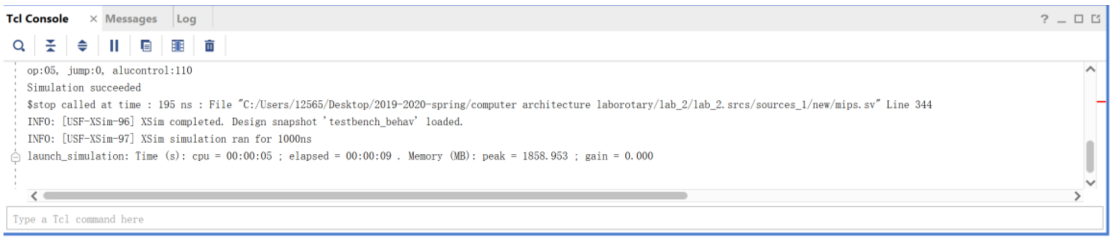
```
          2'b01: alucontrol <= 3'b110; // sub (for beq) architecture
behave of aludec is
          2'b11: alucontrol <= 3'b001;
          default: case(funct) // R-type instructions begin
              6'b100000: alucontrol <= 3'b010; // add process(all )
begin
              6'b100010: alucontrol <= 3'b110; // sub case aluop is
              6'b100100: alucontrol <= 3'b000; // and when "00" =>
alucontrol <= "010" ; -- add (for lw/sw/addi )
              6'b100101: alucontrol <= 3'b001; //or when "01" =>
alucontrol <= "110"; -- sub (for beq)
              6'b101010: alucontrol <= 3'b111; // sit when others =>
case funct is - R-type instructions
              default: alucontrol <= 3'bxxx; // ??? when "100000" =>
alucontrol <= "010"; - add
          endcase // when "100010" => alucontrol <= "110"; - sub
      endcase // when "100100" => alucontrol <= "000"; -- and
endmodule

//decide controls according to the opcode of instruction
//combinational logic
module maindec(
    input logic [5:0] op,
    output logic memtoreg, memwrite,//
    output logic branch, bne, alusrc,
    output logic regdst, regwrite,
    output logic jump,
    output logic [1:0] aluop
    );

    logic [9:0] controls;

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

    always_comb
        case(op)
            6'b000000: controls <= 10'b1100000010; // RTYPE
            6'b100011: controls <= 10'b1010001000; // LW
            6'b101011: controls <= 10'b0010010000; // SW
            6'b000100: controls <= 10'b0001000001; // BEQ
            6'b000101: controls <= 10'b0001100001; // BNE
            6'b001000: controls <= 10'b1010000000; // ADDI
            6'b001101: controls <= 10'b1010000011; // ORI
          6'b000010: controls <= 10'b0000000100; // J
          default: controls <= 10'bxxxxxxxxxx; // illegal op
       endcase
endmodule

//combinational logic
//parse the instruction and decide the params of the datapath
module controller(
    input logic [5:0] op, funct,
    input logic zero,
    output logic memtoreg, memwrite,
    output logic pcsrc, alusrc,
    output logic regdst, regwrite,
    output logic jump,
    output logic [2:0] alucontrol);

    logic [1:0] aluop;
    logic branch;
    logic bne;

    maindec md(op, memtoreg, memwrite, branch, bne, alusrc, regdst, 
regwrite, jump, aluop);
    aludec ad(funct, aluop, alucontrol );

    assign pcsrc = branch & (bne^zero);

endmodule
```

我们修改后的测试代码如下，其中使用了 ori 和 bne 指令。

```
       RAM[0] = 32'h20020004;
       RAM[1] = 32'h20030008;
       RAM[2] = 32'h34420001;//ori $2 1
       RAM[3] = 32'h34630004;//ori $3 4
       RAM[4] = 32'h2067fff7;
       RAM[5] = 32'h00e22025;
       RAM[6] = 32'h00642824;
       RAM[7] = 32'h00a42820;
       RAM[8] = 32'h10a7000a;
       RAM[9] = 32'h0064202a;
       RAM[10] = 32'h10800001;
       RAM[11] = 32'h20050000;
       RAM[12] = 32'h00e2202a;
       RAM[13] = 32'h00853820;
       RAM[14] = 32'h00e23822;
```

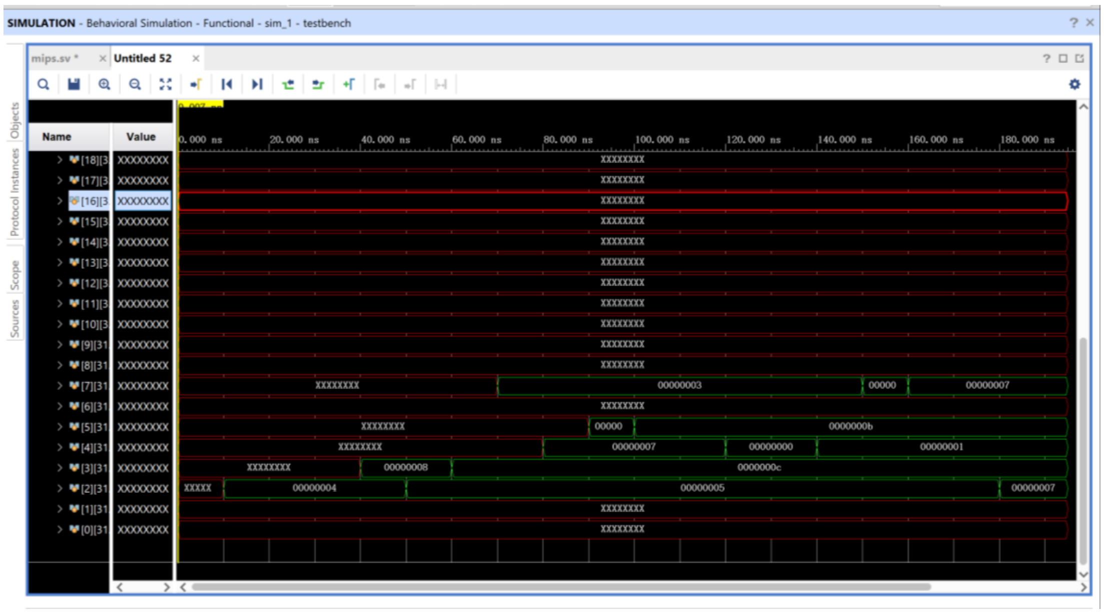
```
        RAM[15] = 32'hac670044;
        RAM[16] = 32'h8c020050;
        RAM[17] = 32'h14440001;//bne $2, $4, #end
        RAM[18] = 32'h20020001;
        RAM[19] = 32'hac020054;
```

模拟后的命令行输出如下，表明模拟成功。



```
op:05, jump:0, alucontrol:110
Simulation succeeded
$stop called at time : 195 ns : File "C:/Users/12565/Desktop/2019-2020-spring/computer architecture laborotary/lab_2/lab_2.srcs/sources_1/new/mips.sv" Line 344
INFO: [USF-XSim-96] XSim completed. Design snapshot 'testbench_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:09 . Memory (MB): peak = 1858.953 ; gain = 0.000
```

寄存器的值为


SIMULATION - Behavioral Simulation - Functional - sim_1 - testbench
mips.sv
Untitled 52
Name
Value
0.000 ns
20.000 ns
40.000 ns
60.000 ns
80.000 ns
100.000 ns
120.000 ns
140.000 ns
160.000 ns
180.000 ns
[18][3
[17][3
[16][3
[15][3
[14][3
[13][3
[12][3
[11][3
[10][3
[9][31
[8][31
[7][31
[6][31
[5][31
[4][31
[3][31
[2][31
[1][31
[0][31
XXXXXXXX
00000003
00000007
0000000b
00000000
00000001
00000008
0000000c
00000004
00000005