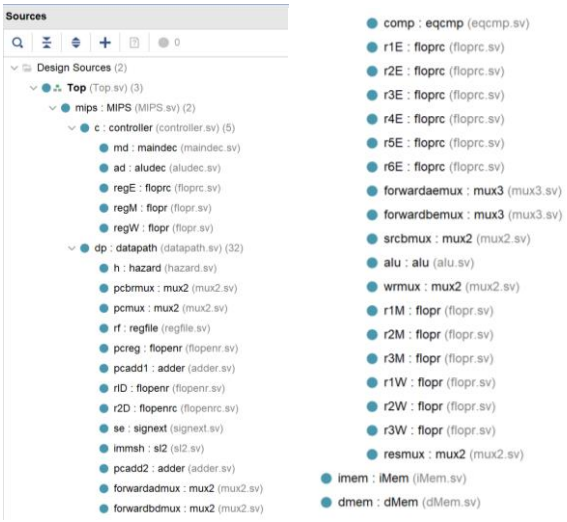
实验三　32 位 MIPS 流水线处理器的设计

【一】设计原理

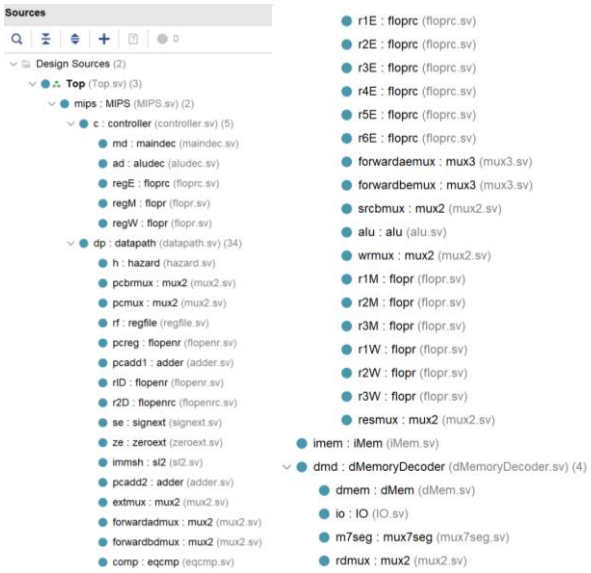
流水线处理器将指令分为五个阶段：取指令（IF），指令译码（ID），取操作数（OF），执行（EX），写回（WB），后一条指令的第 i 步与前一条指令的第 i + 1 步同时进行。
流水线的设计原则是指令流水段个数以最复杂指令所用的功能段个数为准，流水段长度以最复杂的操作所花时间为准。在此种原则下，流水线处理器并不能缩短一条指令的执行时间，但能大大增加指令执行的吞吐率，使得所有指令执行的总时间减少。
流水线处理器的许多部件设计与单周期处理器相同，但主要区别是数据通路部分引入了指令执行阶段的概念，每条指令的执行都将经历五个流水段，每个流水段都在不同的功能代码中执行。
流水段之间存在一个流水段寄存器，用来存放从当前流水段到后续流水段的信息，在每个时钟都会写入一次。

【二】实验方案与部分关键代码

本次实验中我一共实现了两个流水线处理器，其中 PipelineProcessor 是普通的流水线处理器，而后一个 PipelineProcessor_io 添加了 io 接口。
PipelineProcessor 的结构如下：



PipelineProcessor_io 的结构如下：



相较于 ppt 和教材上的代码，我做出的修改主要是 andi 指令的扩展和 io 接口部分的编写。

1.andi 指令的扩展：

流水线处理器的基本框架与单周期处理器相似，andi 指令的扩展也有相近之处。

首先我在主译码器中增加了对 andi 指令操作码的识别：

```
always_comb
    case(op)
        6'b000000: controls = 11'b11000000100; // Rtype
        6'b100011: controls = 11'b10100100000; // LW
        6'b101011: controls = 11'b00101000000; // SW
        6'b000100: controls = 11'b00010000010; // BEQ
        6'b001000: controls = 11'b10100000000; // ADDI
        6'b000010: controls = 11'b00000010000; // J
        6'b001100: controls = 11'b10100001001; // ANDI
        default:   controls = 11'bxxxxxxxxxxx; // ???
    endcase
```

另外 andi 指令是 i 型指令，其 instr[0:5]是立即数的一部分，而非 R 型指令那样的 funct，所以有必要对两者加以区分。我将 aluop 由两位扩展至三位，除了 andi 指令外的所有 aluop 统一在左侧添加 0，andi 指令对应的 aluop 是 100。相应地，我在 alu 译码器中进行了调整：

```
module aludec(input  logic [5:0] funct,
              input  logic [2:0] aluop,
              output logic [2:0] alucontrol);

    always_comb
        case(aluop)
            3'b000: alucontrol = 3'b010; // add
            3'b001: alucontrol = 3'b110; // sub
            3'b100: alucontrol = 3'b000; // andi
```

同时我注意到 andi 指令需要对立即数进行 0 扩展而非符号扩展，所以需要额外写一个 zeroext 模块，并设置变量 immextD，在主译码器中给它赋值，当 immextD=0 时进行符号扩展，当其等于 1 时进行 0 扩展。0 扩展的调用语句如下：

```
signext se(instrD[15:0], signimmD);
zeroext ze(instrD[15:0], zeroimmD);
```

模块内容如下：

```
module zeroext(input  logic [15:0] a,
               output logic [31:0] y);
    assign y[15:0]  = a;
    assign y[31:16] = 16'b0000_0000_0000_0000;
endmodule
```

通过 immext 来选择扩展类型的语句如下：

```
mux2 #(32) extmux(signimmD, zeroimmD, immextD, immD);
```

另外还要根据时钟信号将 immD 存储到 D->E 的步骤间寄存器中：

```
floprc #(32) r3E(clk, reset, flushE, immD,  immE);
```

2.io 接口的编写：

流水线处理器的 io 接口与单周期处理器的 io 接口高度相似。

首先在顶层模块中添加模块 dMemoryDecoder：

```
        .aluoutM(dataadr),
        .writedataM(writedata));
iMem imem(pc[7:2], instr);
dMemoryDecoder dmd(.clk(CLK100MHZ),
                   .writeEN(Write),
                   .addr(dataadr),
                   .writedata(writedata),
                   .readdata(readdata), // output
                   //下面是IO接口部分
                   .IOclock(IOclock),
                   .reset(BTNC),
                   .btnL(BTNL),
                   .btnR(BTNR),
                   .switch(SW),
                   .an(AN),             // output
                   .a2g(A2G));          // output
```

并将原本的 dmem 写入该模块中。

模块 dMemoryDecoder 的完整代码如下：

```
module dMemoryDecoder(input  logic        clk,
                      input  logic        writeEN,
                      input  logic [31:0] addr,
                      input  logic [31:0] writedata,
                      output logic [31:0] readdata,
                      input  logic        IOclock,
                      input  logic        reset,
                      input  logic        btnL,
                      input  logic        btnR,
                      input  logic [15:0] switch,
                      output logic [7:0]  an,
                      output logic [6:0]  a2g);

    logic [11:0] led;
    logic [31:0] readdata1, readdata2;
    logic [31:0] digit;
    logic pWrite, mWrite;

    assign pWrite = (addr[7] == 1'b1) ? writeEN : 0;
    assign mWrite = writeEN & (addr[7] == 1'b0);
    dMem dmem(.clk(clk),
              .we(mWrite),
              .a(addr),
              .wd(writedata),
              .rd(readdata1));

    IO io(.clk(IOclock),
          .reset(reset),
          .pRead(addr[7]),
          .pWrite(pWrite),
          .addr(addr[3:2]),
          .pWriteData(writedata[11:0]),
          .pReadData(readdata2),
          .buttonL(btnL),
          .buttonR(btnR),
          .switch(switch),
          .led(led));

    assign digit = {switch, 4'b0000, led};

    mux7seg m7seg(.clk(IOclock),
                  .reset(reset),
                  .x(digit),
                  .an(an),
                  .a2g(a2g));

    mux2 #(32) rdmux(readdata1, readdata2, addr[7], readdata);
endmodule
```

模块 mux7seg 的功能是七段显示管的译码，其完整代码如下：

```
module mux7seg(input    logic        clk,
               input    logic        reset,
               input    logic [31:0] x,
               output   logic [7:0]  an,
               output   logic [6:0]  a2g);

    logic [2:0] s;
    logic [3:0] digit;
    logic [19:0] clkdiv;

    assign s = clkdiv[19:17];

    always_comb
    case(s)
      0: digit = x[3:0];
      1: digit = x[7:4];
      2: digit = x[11:8];
      3: digit = x[15:12];
      4: digit = x[19:16];
      5: digit = x[23:20];
      6: digit = x[27:24];
      7: digit = x[31:28];
      default: digit = x[3:0];
    endcase
```

```
always_comb
case(s)
  0: an = 8'b1111_1110;
  1: an = 8'b1111_1101;
  2: an = 8'b1111_1011;
  3: an = 8'b1111_0111;
  4: an = 8'b1110_1111;
  5: an = 8'b1101_1111;
  6: an = 8'b1011_1111;
  7: an = 8'b0111_1111;
  default: an = 8'b1111_1110;
endcase

 always @(posedge clk or posedge reset)
   begin
     if(reset == 1)
       clkdiv <= 0;
     else
       clkdiv <= clkdiv+1;
   end
```

```
always_comb
  begin
  if (s == 3 & x[11:0] != 0)
    a2g = 7'b0110111;
  else
    case(digit)
      'h0: a2g = 7'b1000000;
      'h1: a2g = 7'b1111001;
      'h2: a2g = 7'b0100100;
      'h3: a2g = 7'b0110000;
      'h4: a2g = 7'b0011001;
      'h5: a2g = 7'b0010010;
      'h6: a2g = 7'b0000010;
      'h7: a2g = 7'b1111000;
      'h8: a2g = 7'b0000000;
      'h9: a2g = 7'b0010000;
      'hA: a2g = 7'b0001000;
      'hB: a2g = 7'b0000011;
      'hC: a2g = 7'b1000110;
      'hD: a2g = 7'b0100001;
      'hE: a2g = 7'b0000110;
      'hF: a2g = 7'b0001110;
      default: a2g = 7'b1000000;
    endcase
  end
endmodule
```

当 SW 输入的数值之和不为 0 时，第五个七段显示管会变成“=”，否则它将显示 0。

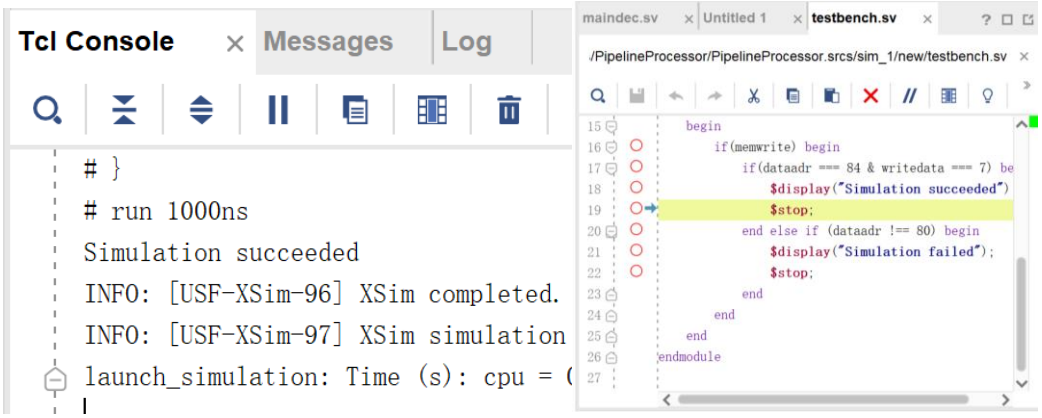
【三】仿真截图

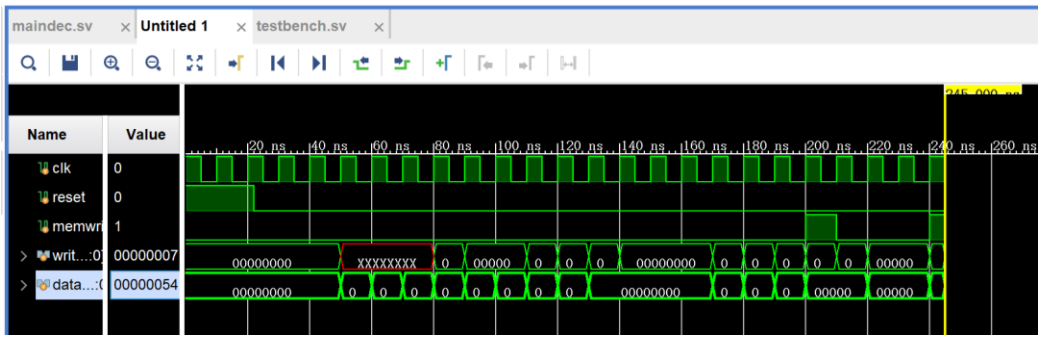
1.PipelineProcessor 的仿真结果：

仿真测试代码截图如下：

```
20020005
2003000c
2067fff7
00e22025
00642824
00a42820
10a7000a
0064202a
10800001
20050000
00e2202a
00853820
00e23822
ac670044
8c020050
08000011
20020001
ac020054
```

仿真结果截图如下：





2.PipelineProcessor_io 的仿真结果：

仿真测试代码截图如下：

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

仿真结果截图如下：

【四】实验开发板照片

测试输入是 12+34，开发板截图如下：