流水线处理器

无 91 2019010999 杨坤泽

一、 实验目的

1. 理解并掌握外设的设计；
2. 理解并用合适的方法解决流水线中的冒险和数据关联问题；
3. 测试验证流水线处理器的功能和进行流水线处理器的性能分析。

二、 设计方案

流水线处理器以多周期处理器的设计为基础，共分为16个模块——1个顶层模块*pipelineCPU*与15个功能模块。处理器为5级流水线处理器，将指令分为*ID*、*IF*、*EX*、*MEM*、*WB*阶段，各阶段间控制信号的传递与数据的传递通过寄存器实现，方便起见控制信号直接在*IDIF*寄存器中生成。

四级寄存器分别记为*IFID_Reg*、*IDEX_Reg*、*EXMEM_Reg*、*MEMWB_Reg*；*clock_long*模块为分频模块；*PC*模块为实现*PC*寄存器功能的模块，输入端为经过阻塞模块控制的*PC*信号；*InstructionMem*模块为指令存储器模块，起到存储指令的作用；*RegisterFile*模块为寄存器堆模块，可以实现寄存器的读写与数据存储功能；*ImmProcess*为立即数拓展模块，根据指令集中指令的不同分为有符号拓展与无符号拓展两种，同时还需要*Lui*指令的立即数操作；*ALU_add*模块为组合逻辑加法器；*ALUControl*模块主要功能为根据指令的不同生成*ALU*模块操作的不同控制信号，而*ALU*模块的作用则为根据控制信号和输入数据执行不同的操作；*DataMem*模块为数据存储器模块，主要作用为存储初始数据。*ForwardingUnit*为转发单元，主要功能为生成转发多路选择器的控制信号，再通过组合逻辑和控制信号实现转发功能；*HazardUnit*为阻塞单元，主要功能是生成*Load_Use*冒险出现时所需要的阻塞控制信号，再通过组合逻辑的多路选择器实现阻塞功能。而对于控制冒险所需要的清空操作，通过组合逻辑生成*flush*控制信号，再同样通过组合逻辑和多路选择器实现*flush*操作。

最终在流水线处理器的功能正确后，要实现通过外设的显示，则需要用汇编语言——先将结果转换为16进制表示的4位数字分别存在4个寄存器中，并将数字翻译为外设控制信号，再轮流将控制信号写入数据存储器中的外设部分，通过外设部分实现数据的显示。

三、 关键代码

流水线处理器的所有代码可见文件夹中的*vivado*程序，由于代码与变量较多，只列出部分代码，与多周期重合度较高的代码未列出。

*pipelineCPU*的数据通路：

```
1. clock_long cl(clk,reset,clk_o);
2. PC PCreg(reset,clk_o,PC_hazard_in,PC_o);
3. InstructionMem InstMem(reset,clk_o,PC_o,inst);
4. IFID_REG ifid_reg(reset,clk_o,IFID_PC_hazard_in_flush,inst_hazard
   _flush,IFID_PC_o,inst_o,IFID_ExtOp,IFID_LuiOp,IFID_ALUOp,IFID_ALU
```

```
SrcA,IFID_ALUSrc,IFID_RegDst,IFID_MemRead,IFID_MemWr,IFID_Branch,
IFID_MemtoReg,IFID_RegWr,Jump);
RegisterFile RF(reset, clk_o, MEMWB_RegWr, inst_o[25:21], inst_o[
20:16], MEMWB_RtorRd_o, write_data, Read_data1, Read_data2);
ImmProcess Imm(IFID_ExtOp,IFID_LuiOp,inst_o[15:0],ImmExtOut,ImmSh
ift);
IDEX_REG idex_reg(reset,clk_o,Read_data1_flush,Read_data2_flush,I
FID_PC_o_flush,ImmShift_flush,ImmExtOut_flush,IDEX_OpCode_in_flus
h,IFID_rs_in_flush,IFID_rt_in_flush,IFID_rd_in_flush,IFID_shamt_i
n_flush,IFID_Funct_in_flush,
                  IDEX_Read_1_o,IDEX_Read_2_o,IDEX_PC_o,IDEX_ImmS
hift_o,IDEX_ImmExtOut_o,IDEX_OpCode_o,IDEX_rs_o,IDEX_rt_o,IDEX_rd
_o,IDEX_shamt_o,IDEX_Funct_o,
                   IFID_ExtOp_flush,IFID_LuiOp_flush,IFID_ALUOp_f
lush,IFID_ALUSrcA_flush,IFID_ALUSrc_flush,IFID_RegDst_flush,IDEX_
MemRead_hazard_flush,IDEX_MemWr_hazard_flush,IDEX_Branch_hazard_f
lush,IFID_MemtoReg_flush,IDEX_RegWr_hazard_flush,
                  IDEX_ExtOp,IDEX_LuiOp,IDEX_ALUOp,IDEX_ALUSrcA,I
DEX_ALUSrc,IDEX_RegDst,IDEX_MemRead,IDEX_MemWr,IDEX_Branch,IDEX_M
emtoReg,IDEX_RegWr);
ALU_add adder(reset,clk_o,IDEX_ImmShift_o,IDEX_PC_o,adder_out);
ALUControl conf(IDEX_ALUOp,IDEX_Funct_o,ALUConf,Sign);
ALU alu(ALUConf,IDEX_OpCode_o,Sign,In1,In2,zero,result);
EXMEM_REG exmem_reg(reset,clk_o,adder_out,zero,result,IDEX_Read_2
_o,EXMEM_RtorRd_in,IDEX_OpCode_o,EXMEM_adder_o,EXMEM_zero,EXMEM_r
esult_o,EXMEM_Read_2_o,EXMEM_RtorRd_o,EXMEM_OpCode_o,
                   IDEX_MemWr,IDEX_MemRead,IDEX_Branch,IDEX_Memt
oReg,IDEX_RegWr,EXMEM_MemWr,EXMEM_MemRead,
                   EXMEM_Branch,EXMEM_MemtoReg,EXMEM_RegWr);
DataMem DM(reset,clk_o,EXMEM_result_o,write_data_hazard,EXMEM_Mem
Wr,EXMEM_MemRead,DM_Read_data,code);
MEMWB_REG memwb_reg(reset,clk_o,DM_Read_data,EXMEM_result_o,EXMEM
_RtorRd_o,MEMWB_Read_data_o,MEMWB_result_o,MEMWB_RtorRd_o,
                   EXMEM_MemtoReg,EXMEM_RegWr,EXMEM_MemRead,MEMW
B_MemtoReg,MEMWB_RegWr,MEMWB_MemRead);
ForwardingUnit FU(reset,clk_o,EXMEM_RegWr,MEMWB_RegWr,EXMEM_RtorR
d_o,MEMWB_RtorRd_o,EXMEM_MemWr,MEMWB_MemRead,IDEX_rs_o,IDEX_rt_o,
forwardA,forwardB,forward_MEM);
HazardUnit HU(inst_o,IDEX_MemRead,IDEX_rt_o,inst_o[25:21],inst_o[
20:16],PC_MUX,IFID_MUX,IDEX_MUX);
```

*pipelineCPU*的组合逻辑与多路选择器实现：

```
assign In1=(IDEX_ALUSrcA==2'b00)?forwardA_in:(IDEX_ALUSrcA==2'b11
)?IDEX_shamt_o:32'b0;
```

```
2. assign In2=(IDEX_ALUSrc==2'b00)?forwardB_in:(IDEX_ALUSrc==2'b10)?IDEX_ImmExtOut_o:(IDEX_ALUSrc==2'b01)?(IDEX_PC_o):32'b0;
3. assign forwardA_in=(forwardA==2'b00)?IDEX_Read_1_o:(forwardA==2'b01)?write_data:(forwardA==2'b10)?EXMEM_result_o:32'b0;
4. assign forwardB_in=(forwardB==2'b00)?IDEX_Read_2_o:(forwardB==2'b01)?write_data:(forwardB==2'b10)?EXMEM_result_o:32'b0;
5. assign EXMEM_RtorRd_in=(IDEX_RegDst==2'b00)?IDEX_rt_o:(IDEX_RegDst==2'b01)?IDEX_rd_o:(IDEX_RegDst==2'b10)?5'd31:5'b0;
6. assign PCSource=(((zero==3'b001)&&(IDEX_OpCode_o==6'h04))||((zero==3'b010)&&(IDEX_OpCode_o==6'h11))||((zero==3'b100)&&(IDEX_OpCode_o==6'h12))||((zero==3'b011)&&(IDEX_OpCode_o==6'h13))||((zero==3'b110)&&(IDEX_OpCode_o==6'h14))||(zero==3'b101&&IDEX_OpCode_o==6'h15))&&IDEX_Branch;
7. assign write_data=(MEMWB_MemtoReg==2'b00)?MEMWB_result_o:(MEMWB_MemtoReg==2'b01)?MEMWB_Read_data_o:32'b0;
8. assign PC_in_first=PCSource?adder_out:(PC_o+4);
9. assign PC_in=(Jump==2'b01)?{IFID_PC_o[31:28],inst_o[25:0]<<2}:(Jump==2'b00)?PC_in_first:(Jump==2'b10)?Read_data1:0;
10. assign IFID_PC_in=PC_o+4;
11. assign PC_hazard_in=(PC_MUX==2'b01)?0:(PC_MUX==2'b10)?PC_o:(PC_MUX==2'b00)?PC_in:0;
12. assign IFID_PC_hazard_in=(IFID_MUX==2'b01)?0:(IFID_MUX==2'b10)?IFID_PC_o:(IFID_MUX==2'b00)?PC_o+4:0;
13. assign inst_hazard=(IFID_MUX==2'b01)?0:(IFID_MUX==2'b10)?inst_o:(IFID_MUX==2'b00)?inst:0;
14. assign IDEX_RegWr_hazard=(IDEX_MUX==2'b01)?0:(IDEX_MUX==2'b10)?IDEX_RegWr:(IDEX_MUX==2'b00)?IFID_RegWr:0;
15. assign IDEX_MemWr_hazard=(IDEX_MUX==2'b01)?0:(IDEX_MUX==2'b10)?IDEX_MemWr:(IDEX_MUX==2'b00)?IFID_MemWr:0;
16. assign IDEX_MemRead_hazard=(IDEX_MUX==2'b01)?0:(IDEX_MUX==2'b10)?IDEX_MemRead:(IDEX_MUX==2'b00)?IFID_MemRead:0;
17. assign IDEX_Branch_hazard=(IDEX_MUX==2'b01)?0:(IDEX_MUX==2'b10)?IDEX_Branch:(IDEX_MUX==2'b00)?IFID_Branch:0;
18. assign write_data_hazard=(forward_MEM==2'b00)?EXMEM_Read_2_o:(forward_MEM==2'b01)?MEMWB_Read_data_o:0;
19.
20. assign flush_IFID=((((zero==3'b001&&IDEX_OpCode_o==6'h04)||(zero==3'b010&&IDEX_OpCode_o==6'h11)||(zero==3'b100&&IDEX_OpCode_o==6'h12)||(zero==3'b011&&IDEX_OpCode_o==6'h13)||(zero==3'b110&&IDEX_OpCode_o==6'h14)||(zero==3'b101&&IDEX_OpCode_o==6'h15))&&IDEX_Branch)||(inst_o[31:26]==6'h02||inst_o[31:26]==6'h03||(inst_o[31:26]==6'h0&&inst_o[5:0]==6'h08)))?1:0;//flush IFID
21. assign IFID_PC_hazard_in_flush=flush_IFID?0:IFID_PC_hazard_in;
22. assign inst_hazard_flush=flush_IFID?0:inst_hazard;
```

```
23.
24.assign flush_IFIDEX=((zero==3'b001&&IDEX_OpCode_o==6'h04)||(zero==3'b010&&IDEX_OpCode_o==6'h11)||(zero==3'b100&&IDEX_OpCode_o==6'h12)||(zero==3'b011&&IDEX_OpCode_o==6'h13)||(zero==3'b110&&IDEX_OpCode_o==6'h14)||(zero==3'b101&&IDEX_OpCode_o==6'h15))&&IDEX_Branch?1:0;//flush IFIDEX
25.assign Read_data1_flush=flush_IFIDEX?0:Read_data1;
26.assign Read_data2_flush=flush_IFIDEX?0:Read_data2;
27.assign IFID_PC_o_flush=flush_IFIDEX?0:IFID_PC_o;
28.assign ImmShift_flush=flush_IFIDEX?0:ImmShift;
29.assign ImmExtOut_flush=flush_IFIDEX?0:ImmExtOut;
30.assign IFID_rs_in_flush=flush_IFIDEX?0:inst_o[25:21];
31.assign IFID_rt_in_flush=flush_IFIDEX?0:inst_o[20:16];
32.assign IFID_rd_in_flush=flush_IFIDEX?0:inst_o[15:11];
33.assign IFID_shamt_in_flush=flush_IFIDEX?0:inst_o[10:6];
34.assign IFID_Funct_in_flush=flush_IFIDEX?0:inst_o[5:0];
35.assign IFID_ExtOp_flush=flush_IFIDEX?0:IFID_ExtOp;
36.assign IFID_LuiOp_flush=flush_IFIDEX?0:IFID_LuiOp;
37.assign IFID_ALUOp_flush=flush_IFIDEX?0:IFID_ALUOp;
38.assign IFID_ALUSrcA_flush=flush_IFIDEX?0:IFID_ALUSrcA;
39.assign IFID_ALUSrc_flush=flush_IFIDEX?0:IFID_ALUSrc;
40.assign IFID_RegDst_flush=flush_IFIDEX?0:IFID_RegDst;
41.assign IDEX_MemRead_hazard_flush=flush_IFIDEX?0:IDEX_MemRead_hazard;
42.assign IDEX_MemWr_hazard_flush=flush_IFIDEX?0:IDEX_MemWr_hazard;
43.assign IDEX_Branch_hazard_flush=flush_IFIDEX?0:IDEX_Branch_hazard;
44.assign IFID_MemtoReg_flush=flush_IFIDEX?0:IFID_MemtoReg;
45.assign IDEX_RegWr_hazard_flush=flush_IFIDEX?0:IDEX_RegWr_hazard;
46.assign IDEX_OpCode_in_flush=flush_IFIDEX?0:inst_o[31:26];
```

*ForwardingUnit*转发单元：

```
1. module ForwardingUnit(
2. reset,
3. clk,
4. EXMEM_RegWr,
5. MEMWB_RegWr,
6. EXMEM_RtorRd_in,
7. MEMWB_RtorRd_in,
8. EXMEM_MemWr,
9. MEMWB_MemRead,
10.IDEX_rs_in,
11.IDEX_rt_in,
12.forwardA,
```

```
13.forwardB,
14.forward_MEM
15.    );
16.
17.    input reset;
18.    input clk;
19.    input EXMEM_RegWr;
20.    input MEMWB_RegWr;
21.    input [4:0] EXMEM_RtorRd_in;
22.    input [4:0] MEMWB_RtorRd_in;
23.    input EXMEM_MemWr;
24.    input MEMWB_MemRead;
25.    input[4:0] IDEX_rs_in;
26.    input [4:0] IDEX_rt_in;
27.    output reg [1:0] forwardA;
28.    output reg [1:0] forwardB;
29.    output reg [1:0] forward_MEM;
30.
31.    always@(*)
32.    begin
33.      if(EXMEM_RegWr&&(EXMEM_RtorRd_in!=0)&&(EXMEM_RtorRd_in==IDE
   X_rs_in))
34.        forwardA<=2'b10;
35.      else if(MEMWB_RegWr&&(MEMWB_RtorRd_in!=0)&&(MEMWB_RtorRd_in
   ==IDEX_rs_in)&&((EXMEM_RtorRd_in!=IDEX_rs_in)||~EXMEM_RegWr))
36.        forwardA<=2'b01;
37.      else
38.        forwardA<=2'b00;
39.      if(EXMEM_RegWr&&(EXMEM_RtorRd_in!=0)&&(EXMEM_RtorRd_in==IDE
   X_rt_in))
40.        forwardB<=2'b10;
41.      else if(MEMWB_RegWr&&(MEMWB_RtorRd_in!=0)&&(MEMWB_RtorRd_in
   ==IDEX_rt_in)&&((EXMEM_RtorRd_in!=IDEX_rt_in)||~EXMEM_RegWr))
42.        forwardB<=2'b01;
43.      else
44.        forwardB<=2'b00;
45.      if(MEMWB_MemRead&&EXMEM_MemWr&&(MEMWB_RtorRd_in==EXMEM_Rtor
   Rd_in))
46.        forward_MEM<=2'b01;
47.      else
48.        forward_MEM<=2'b00;
49.    end
50.
51.endmodule
```

*HazardUnit*阻塞单元：

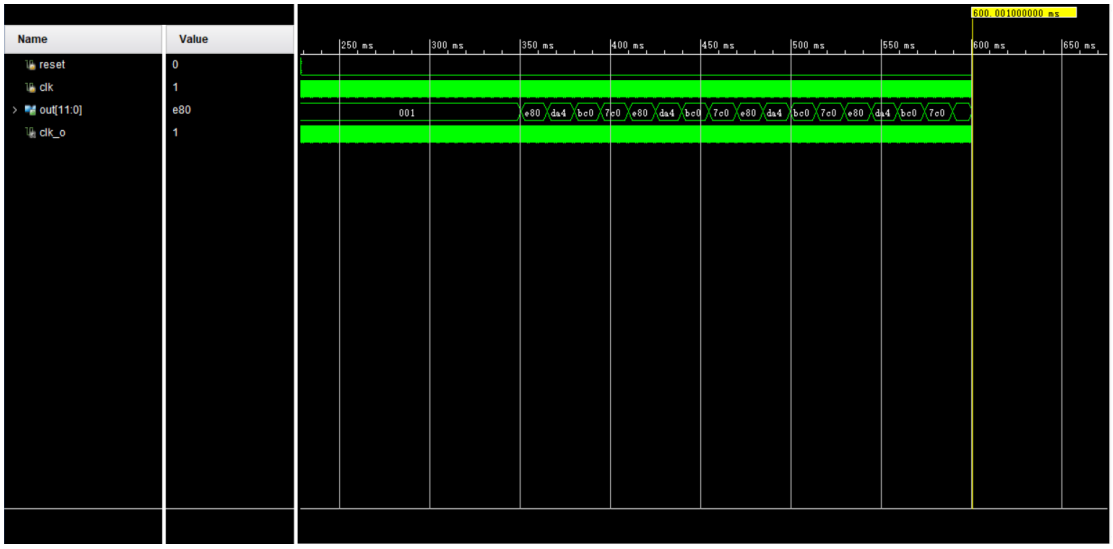
```
module HazardUnit(
IFID_inst_o,
IDEX_MemRead,
IDEX_rt_o,
IFID_rs_o,
IFID_rt_o,
PC_MUX,
IFID_MUX,
IDEX_MUX
    );

   input [31:0] IFID_inst_o;
   input IDEX_MemRead;
   input [4:0] IDEX_rt_o;
   input [4:0] IFID_rs_o;
   input [4:0] IFID_rt_o;
   output reg [1:0] PC_MUX;
   output reg [1:0] IFID_MUX;
   output reg [1:0] IDEX_MUX;

   always@(*)
     begin
       if((IDEX_MemRead&&((IDEX_rt_o==IFID_rs_o)||(IDEX_rt_o==IFID_rt_o))))
         begin
         PC_MUX<=2'b10;
         IFID_MUX<=2'b10;
         IDEX_MUX<=2'b01;
         end
       else
         begin
         PC_MUX<=2'b00;
         IFID_MUX<=2'b00;
         IDEX_MUX<=2'b00;
         end
       end
endmodule
```

四、 仿真结果及分析

仿真结果如下：



理论上数据的输出结果为“0028”，翻译成控制信号则为“7c0”“bc0”“da4”“e80”，根据仿真结果4位数字循环显示，间隔约为14ms，则仿真结果符合预期，且硬件显示无误。

五、 综合情况

查找表占用情况如下：

pipelineCPU	7834
exmem_reg (EXMEM_REG)	4019
DM (DataMem)	2208
idex_reg (IDEX_REG)	671
RF (RegisterFile)	544
memwb_reg (MEMWB_REG)	133
ifid_reg (IFID_REG)	110
InstMem (InstructionMem)	91
PCreg (PC)	47
cl (clock_long)	15

寄存器占用情况如下：

Name	Used
pipelineCPU	9614
DM (DataMem)	8192
RF (RegisterFile)	992
idex_reg (IDEX_REG)	141
exmem_reg (EXMEM_REG)	87
ifid_reg (IFID_REG)	85
memwb_reg (MEMWB_REG)	72
PCreg (PC)	32
cl (clock_long)	13

符合预期结果。

六、 思想体会

本次流水线处理器的代码编写与多周期处理器有所不同，除了一些模块是在多周期的基础上有所改动完成的之外全部是我自己编写的，包括思考数据通路的构成，转发、阻塞和清空的具体实现方式，以及每一个变量在数据通路中的传递等等，让我充分理解了流水线处理器解决各种冒险的方式的实际实现是怎么样的；并且在*debug*的过程中让我明白了一个体量比较大的程序的代码编写细节有多么重要，一点微小的错误可能就会导致十分奇怪的结果。十分感谢张老师和助教一个学期的辛苦付出，让我们能够通过这门课程学到许多宝贵的知识！