实验项目名称：<u>Pipelined CPU with forwarding</u>

一、 实验目的和要求

1. Design the Bypass Unit of Datapath of 5-stages Pipelined CPU
2. Modify the CPU Controller
1) Conditions in Which Pipeline Forwards.
2) Conditions in Which Pipeline Stalls.
3. Verify the Pipeline CPU with program and observe the execution of program

二、 实验内容和原理

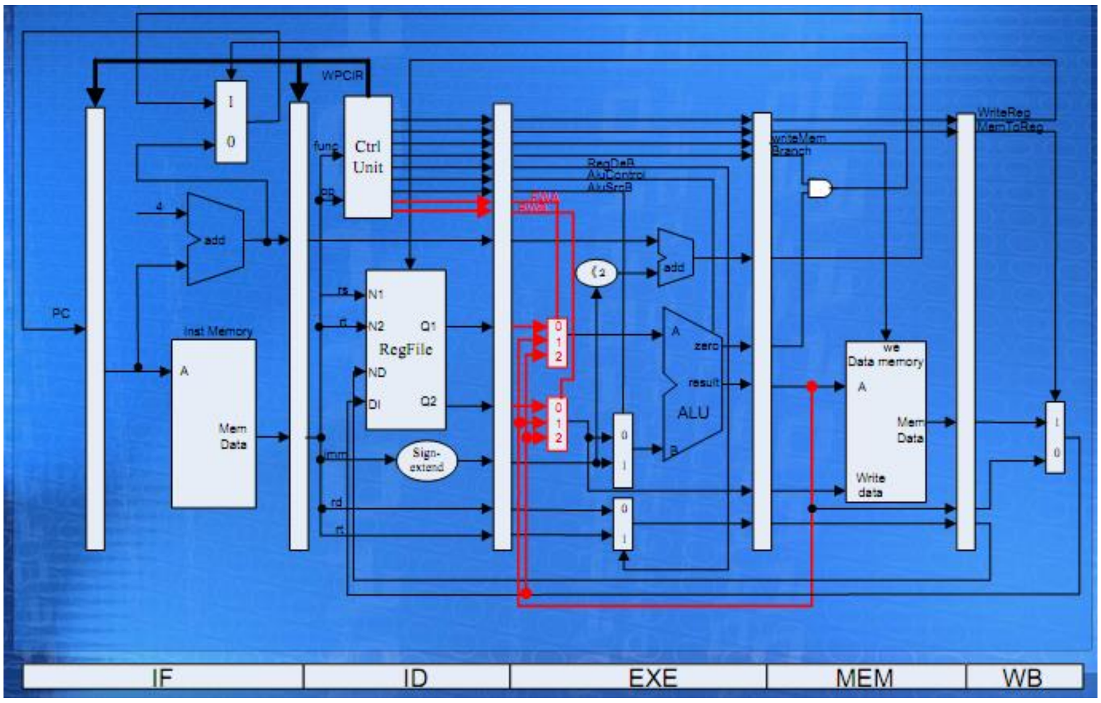
1. Minimizing Data Hazard Stalls by Forwarding

Forwarding, also called bypassing, short-circuiting, immediately pass the result to the instruction that wants it instead of waiting to store it. In most cases, data hazards problem can be resolved by forwarding.

2. When to use Forwarding:

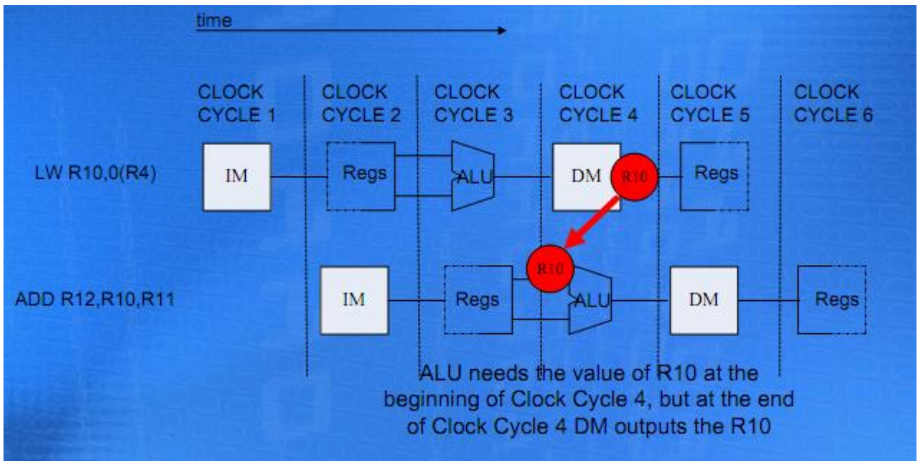
1) EX/MEM.ALUoutput → ALU input
   a) The previous instruction in EX/MEM is an ALU instruction.
   b) The instruction in ID/EX has rs or rt source register.
   c) (EX/MEM.rd == ID/EX.rs) or (EX/MEM.rd == ID/EX.rt)
2) MEM/WB.ALUoutput → ALU input
   a) The previous instruction in MEM/WB is an ALU instruction.
   b) The instruction in ID/EX has rs or rt source register.
   c) (MEM/WB.rd == ID/EX.rs) or (MEM/WB.rd == IF/ID.rt)
   d) (EX/MEM.rd != ID/EX.rs) or (EX/MEM.rd != ID/EX.rt): this condition guarantees that only the newest modified result can be forwarded.
3) MEM/WB.LW → ALU input
   a) The previous instruction in MEM/WB is Load.
   b) The instruction in ID/EX has rs or rt source register.
   c) (MEM/WB.rd == ID/EX.rs) or (MEM/WB.rd == ID/EX.rt)
   d) (EX/MEM.rd != ID/EX.rs) or (EX/MEM.rd != ID/EX.rt): this condition guarantees that only the newest modified result can be forwarded.

3. Datapath with Forwarding Unit



To implement forwarding unit, buses from later stages to earlier stages should be added.

4. Condition in Which Bypass Unit doesn't work:



In such conditions, stalls should be inserted to avoid data hazard.

三、 实验过程和数据记录

The modified code of the ctrl_unit module:

```
module ctrl_unit(clk, rst, if_instr, instr, cu_branch, cu_wreg, cu_m2reg,
cu_wmem, cu_aluc, cu_shift, cu_aluimm, cu_sext, cu_regrt, cu_wpcir,
cu_fwda, cu_fwdb);
    //forwarding signals
    output [1:0] cu_fwda;
    output [1:0] cu_fwdb;
    wire [1:0]cu_fwda;
    wire [1:0]cu_fwdb;
    assign AfromEx = ((ex_op==`OP_ALUOp)&&(ex_rd==rs)&&(ex_rd!=0));
    assign BfromEx = ((ex_op==`OP_ALUOp)&&(ex_rd==rt)&&(ex_rd!=0));
    assign AfromMem = ((mem_op==`OP_ALUOp) || (mem_op==`OP_LW)) &&
(ex_rd!=rs) && (mem_rd==rs) && (mem_rd!=0);
    assign BfromMem = ((mem_op==`OP_ALUOp) || (mem_op==`OP_LW)) &&
(ex_rd!=rt) && (mem_rd==rt) && (mem_rd!=0);
    assign stall =  AfromExLW || BfromExLW ;
    assign cu_wpcir = stall;
    assign cu_fwda[1:0] = (AfromEx == 1)? 2'b01: ((AfromMem==1)? 2'b10:
2'b00);
    assign cu_fwdb[1:0] = (BfromEx == 1)? 2'b01: ((BfromMem==1)? 2'b10:
2'b00);
    assign   mem_rd[4:0]   =   (mem_op==`OP_LW)?   mem_instr[20:16]:
mem_instr[15:11];
    assign    wb_rd[4:0]    =    (wb_op==`OP_LW)?    wb_instr[20:16]:
wb_instr[15:11];
```

The modified code of the id_stage module:

```
module id_stage (clk, rst, if_inst, if_pc4, wb_destR, wb_dest,wb_wreg,
 cu_wreg, cu_m2reg, cu_wmem, cu_aluc, cu_shift, cu_aluimm, cu_branch,
 id_pc4, id_inA, id_inB, id_imm, cu_regrt, rt, rd, cu_wpcir, id_fwda,
 id_fwdb,  IF_ins_type,  IF_ins_number,  ID_ins_type,  ID_ins_number,
 which_reg, reg_content);
    //forwarding signals
    output [1:0] id_fwda;
    output [1:0] id_fwdb;
    ctrl_unit  x_ctrl_unit(clk,  rst,  if_inst[31:0],  reg_inst[31:0],
cu_branch, cu_wreg, cu_m2reg, cu_wmem, cu_aluc, cu_shift, cu_aluimm,
cu_sext, cu_regrt, cu_wpcir, id_fwda, id_fwdb);
```

The modified code of the ex_stage module:

```
module ex_stage (clk,  id_imm, id_inA, id_inB, id_wreg, id_m2reg, id_wmem,
id_aluc, id_aluimm,id_shift, id_branch, id_pc4,id_regrt,id_rt,id_rd,
    ex_wreg, ex_m2reg, ex_wmem, ex_aluR, ex_inB, ex_destR, ex_branch,
ex_pc,   ex_zero,   id_fwda,id_fwdb,mem_aluR,wb_dest,   ID_ins_type,
ID_ins_number, EXE_ins_type, EXE_ins_number);
    //entering forwarding signals from id_stage
    input [1:0] id_fwda;
    input [1:0] id_fwdb;
    //forwarding data from EX/MEM, and it’s provided by mem_stage module
    input [31:0] mem_aluR;
    //forwarding data from MEM/WB, and it’s provided by wb_stage module
    input [31:0] wb_dest;
    wire [1:0] ex_fwda;      //latched forward control singal
    wire [1:0] ex_fwdb;      //latched forward control singal
    //add multiplexors for forwading
    assign   a_in   =   eshift?   sa:   ((ex_fwda==2'b10)?   wb_dest:
((ex_fwda==2'b01)? mem_aluR: edata_a));
    assign  b_in  =  ealuimm?  odata_imm:  ((ex_fwdb==2'b10)?  wb_dest:
((ex_fwdb==2'b01)? mem_aluR: edata_b));
    //latch the entering forward control singals
    Reg_ID_EXE x_Reg_ID_EXE(clk,  id_wreg,  id_m2reg,  id_wmem,  id_aluc,
id_shift, id_aluimm,  id_inA, id_inB, id_imm, id_branch, id_pc4, id_regrt,
id_rt, id_rd, ex_wreg, ex_m2reg, ex_wmem, ealuc, eshift,     ealuimm,
edata_a,  edata_b,  odata_imm,  ex_branch,  epc4,  e_regrt,  e_rt,  e_rd,
ID_ins_type,  ID_ins_number,  EXE_ins_type,  EXE_ins_number,  id_fwda,
id_fwdb, ex_fwda, ex_fwdb);
```

The modified code of the Reg_ID_EXE module:

```
module Reg_ID_EXE(clk,  wreg,  m2reg,  wmem,  aluc,  shift,  aluimm,
data_a,  data_b,  data_imm,  id_branch,id_pc4,  id_regrt,  id_rt,  id_rd,
ewreg, em2reg, ewmem, ealuc, eshift, ealuimm, odata_a, odata_b, odata_imm,
e_branch,  e_pc4,  e_regrt,  e_rt,  e_rd,  ID_ins_type,  ID_ins_number,
EXE_ins_type, EXE_ins_number, id_fwda, id_fwdb, ex_fwda, ex_fwdb);
    //forwarding signals
    input [1:0] id_fwda;
    input [1:0] id_fwdb;
    output [1:0]ex_fwda;
    output [1:0]ex_fwdb;
    reg [1:0]ex_fwda;
    reg [1:0]ex_fwdb;
    //latch the entering forward control singals
    always@(posedge clk) begin
            ex_fwda <= id_fwda;
            ex_fwdb <= id_fwdb;
    end
endmodule
```

The modified code of the top module:

```
    //forwarding signals
    wire [1:0]id_fwda;
    wire [1:0]id_fwdb;

   if_stage x_if_stage(clk0, BTN3, rst, pc, mem_pc, mem_branch, id_wpcir,
if_npc,           if_pc4,           if_inst,            IF_ins_type,
IF_ins_number,ID_ins_type,ID_ins_number);

    id_stage   x_id_stage(BTN3,   rst,   if_inst,   if_pc4,   wb_destR,
wb_dest,wb_wreg,  id_wreg,  id_m2reg,  id_wmem,  id_aluc,  id_shift,
id_aluimm, id_branch, id_pc4, id_inA, id_inB, id_imm, id_regrt, id_rt,
id_rd,   id_wpcir,id_fwda,   id_fwdb,   ID_ins_type,   ID_ins_number,
EX_ins_type, EX_ins_number, {1'b0,which_reg}, reg_content);

    ex_stage x_ex_stage(BTN3, id_imm, id_inA, id_inB, id_wreg, id_m2reg,
id_wmem,        id_aluc,        id_aluimm,id_shift,        id_branch,
id_pc4,id_regrt,id_rt,id_rd, ex_wreg, ex_m2reg, ex_wmem, ex_aluR, ex_inB,
ex_destR, ex_branch, ex_pc, ex_zero, id_fwda, id_fwdb, mem_aluR, wb_dest,
EX_ins_type, EX_ins_number, MEM_ins_type, MEM_ins_number);

    mem_stage  x_mem_stage(BTN3,  ex_destR,  ex_inB,  ex_aluR,  ex_wreg,
ex_m2reg,  ex_wmem,  ex_branch,ex_pc,ex_zero,     mem_wreg,  mem_m2reg,
mem_mdata,  mem_aluR,  mem_destR,  mem_branch,  mem_pc,  MEM_ins_type,
MEM_ins_number, WB_ins_type, WB_ins_number);

    wb_stage x_wb_stage(BTN3, mem_destR, mem_aluR, mem_mdata, mem_wreg,
mem_m2reg,     wb_wreg,     wb_dest,     wb_destR,     WB_ins_type,
WB_ins_number,OUT_ins_type, OUT_ins_number);
```

四、 实验结果分析

Program for verification is as follows:

MIPS code	Bin data	Address	Inst. Type
lw r1, $20(r0)	0x8c01_0014	0	6
lw r2, $21(r0)	0x8c02_0015	1	6
add r3, r1, r2	0x0022_1820	2	1
add r2,r0,r0	0x0000_1020	3	1

sub r4, r1, r3	0x0023_2022	4	2
and r5, r3, r4	0x0064_2824	5	3
nor r6, r4, r5	0x0085_3027	6	5
sw r6, $22(r0)	0xac06_0016	7	7
beq r6,r7,-8	0x10c7_fff8	8	8

Data Mem[20]=1

Data Mem[21]=4

The result of execution is shown as follows:

Clock Count	Instruction Code	Instruction	“stage name” /number/type	Comment
00	01234567		f01d01e01m01w01	The first instruction “lw r1, $20(r0)” enters the IF stage.
01	8c010014	lw r1, $20(r0)	f1fd06e00m00w00	The first instruction “lw r1, $20(r0)” enters the ID stage. And a new instruction “lw r2, $21(r0)” enters the IF stage.
02	8c020015	lw r2, $21(r0)	f2fd16e06m00w00	The first instruction “lw r1, $20(r0)” enters the EX stage. And a new instruction “add r3, r1, r2” enters the IF stage.
03	00221820	add r3, r1, r2	f3fd2fe16m06w00	The first instruction “lw r1, $20(r0)” enters the MEM stage. And the instruction “add r3,

				r1, r2” enters the ID stage and find that the RAW data dependency with the first instruction “lw r1, $20(r0)” on r1 is resolved by forwarding. However, there still exits RAW data dependency with the instruction “lw r2, $21(r0)” on r2. And the data hazard can’t be resolved by forwarding, so a stall is inserted.
04	00221820	add r3, r1, r2	f3fd21e1fm16w06	The first instruction “lw r1, $20(r0)” enters the WB stage. Now the instruction “add r3, r1, r2” can actually enter the ID stage.
05	00001020	add r2,r0,r0	f4fd31e21m1fw16	The first instruction “lw r1, $20(r0)” completes the WB stage, so the content of r1 becomes 1. In addition, the instruction “lw r2,

				$21(r0)” completes the MEM stage, so the content of r2 is forwarding to “add r3, r1, r2”.
06	00232022	sub r4, r1, r3	f5fd42e31m21w1f	The instruction “lw r2, $21(r0)” completes the WB stage, so the content of r2 becomes 4. In addition, the instruction “add r3, r1, r2” completes the EX stage, so the content of r3 is forwarding to “sub r4, r1, r3”.
07	00642824	and r5, r3, r4	f6fd53e42m31w21	The inserting stall completes the WB stage. In addition, the instruction “add r3, r1, r2” completes the MEM stage, so the content of r3 is forwarding to “and r5, r3, r4”.
08	00853027	nor r6, r4, r5	f7fd65e53m42w31	The instruction “add r3, r1, r2” completes the WB stage, so the content of r3 becomes

				5. In addition, the instruction “sub r4, r1, r3” completes the EX stage, so the content of r4 is forwarding to “and r5, r3, r4” and “nor r6, r4, r5” exactly when the former instruction enters the EX stage.
09	ac060016	sw r6, $22(r0)	f8fd77e65m53w42	The instruction “add r2, r0, r0” completes the WB stage, so the content of r2 becomes 0. In addition, the instruction “and r5, r3, r4” completes the EX stage, so the content of r5 is forwarding to “nor r6, r4, r5” exactly when it enters the EX stage.
0a	10c7fff8	beq r6,r7,-8	f9fd88e77m65w53	The instruction “sub r4, r1, r3” completes the WB stage, so the content of r4 becomes 0x ffff fffc. In addition, the instruction “nor r6, r4, r5” completes the

				EX stage, so the content of r6 is forwarding to “sw r6, $22(r0)” exactly when it enters the EX stage.
0b	00000000	nop	fafd9ce88m77w65	The instruction “and r5, r3, r4” completes the WB stage, so the content of r5 becomes 4.
0c	00000000	nop	fbfdace9cm88w77	The instruction “nor r6, r4, r5” completes the WB stage, so the content of r6 becomes 3.
0d	00000000	nop	fcfdbceacm9cw88	The instruction “sw r6, $22(r0)” completes the WB stage.
0e	00000000	nop	fdfdccebcmacw9c	The instruction “beq r6,r7,-8” completes the WB stage. The content of r6 is not equal to that of r7, so no jumps occur.

The result indicates that the code implements the required functions correctly.

五、 讨论与心得

This experiment requires careful thought and a little more debugging work for there’re some problems in the referenced codes:

1. In the referenced code:

```
if(WB.WriteReg
and(WB.RegisterRd!=0)
and(MEM.RegisterRd==EXE.RegisterRs)
and(WB.RegisterRd==EXE.RegisterRs)) ForwardA=10
if(WB.WriteReg
and(WB.RegisterRd!=0)
and(MEM.RegisterRd==EXE.RegisterRt)
and(WB.RegisterRd==EXE.RegisterRt)) ForwardB=10
```

`MEM.RegisterRd==EXE.RegisterRs` and `MEM.RegisterRd==EXE.RegisterRt` should be

`MEM.RegisterRd!=EXE.RegisterRs` and `MEM.RegisterRd!=EXE.RegisterRt`

These conditions guarantee that only the newest modified result can be forwarded.

2. The MEM.Register and WB.Register in the above referenced code exactly represents the instruction that enters the MEM stage or WB stage. However, this method leads to one stage late. For example, assume that forwarding “EX/MEM.ALUoutput → ALU input” occurs, then the result of ALU, when completes the EX stage, can be forwarded to the next instruction. This condition should be detected when it just enters the EX stage rather than after the EX stage.

Therefore, we use the following method in our ctrl_unit module:

```
    assign AfromEx = ((ex_op==`OP_ALUOp)&&(ex_rd==rs)&&(ex_rd!=0));
    assign BfromEx = ((ex_op==`OP_ALUOp)&&(ex_rd==rt)&&(ex_rd!=0));
    assign AfromMem = ((mem_op==`OP_ALUOp) || (mem_op==`OP_LW)) &&
(ex_rd!=rs) && (mem_rd==rs) && (mem_rd!=0);
    assign BfromMem = ((mem_op==`OP_ALUOp) || (mem_op==`OP_LW)) &&
(ex_rd!=rt) && (mem_rd==rt) && (mem_rd!=0);
```

Actually, in order to verify our thoughts above, we’ve implemented the former method and found that it was wrong.

3. There’s an error in the former code of the project:

```
    assign mem_rd[4:0] = mem_instr[15:11];
    assign wb_rd[4:0] = wb_instr[15:11];
```

It should be modified into:

```
    assign mem_rd[4:0] = (mem_op==`OP_LW)? mem_instr[20:16]:
mem_instr[15:11];
    assign wb_rd[4:0] = (wb_op==`OP_LW)? wb_instr[20:16]:
wb_instr[15:11];
```

4. The forwarding signals id_fwda、id_fwdb should be latched. We’ve done that in Reg_ID_EXE module.