计算机组成实验指导书
Pipelined RISC with Forwarding & Stalls

1. OVERVIEW

1.1 实验名称

简单的类 MIPS 多周期流水化处理器实现

1.2 实验目的

理解并实现 CPU 的 Pipeline，以及 Data Hazard, Branch Hazard 的处理。

1.3 实验范围

本次实验将覆盖以下范围

- ISE 的使用
- 使用 VerilogHDL 进行逻辑设计
- VirtexII Pro 实验板的使用（可选）

1.4 实验预计时间

每次 2 小时，6~8 次。

1.5 实验要求

作业/报告邮箱：coa.2012.assignment@gmail.com

- 2～3 人一组，自由组合
- 每周提交周报（组长负责），最终提交实验报告，周报和实验报告模板，见 WeeklyReport.doc 文件
- 第一周提交总体设计和**接口文档**，工程中尽量避免接口修改
- 运行指定代码并给出运行结果；推荐使用自编代码，但代码、实验报告和 PPT 中必须体现各类 Hazard
- ~~针对上面的代码画出 5 级流水时序图~~
- 每组制作 PPT 一份进行课程设计验收

1.6 注意事项

本实验的逻辑设计工具为 Xilinx ISE9.1，但不仅限于此，学生可以使用自己喜欢的逻辑设计工具，如 Snyplify 等，也可直接在 ModelSim 中完成。

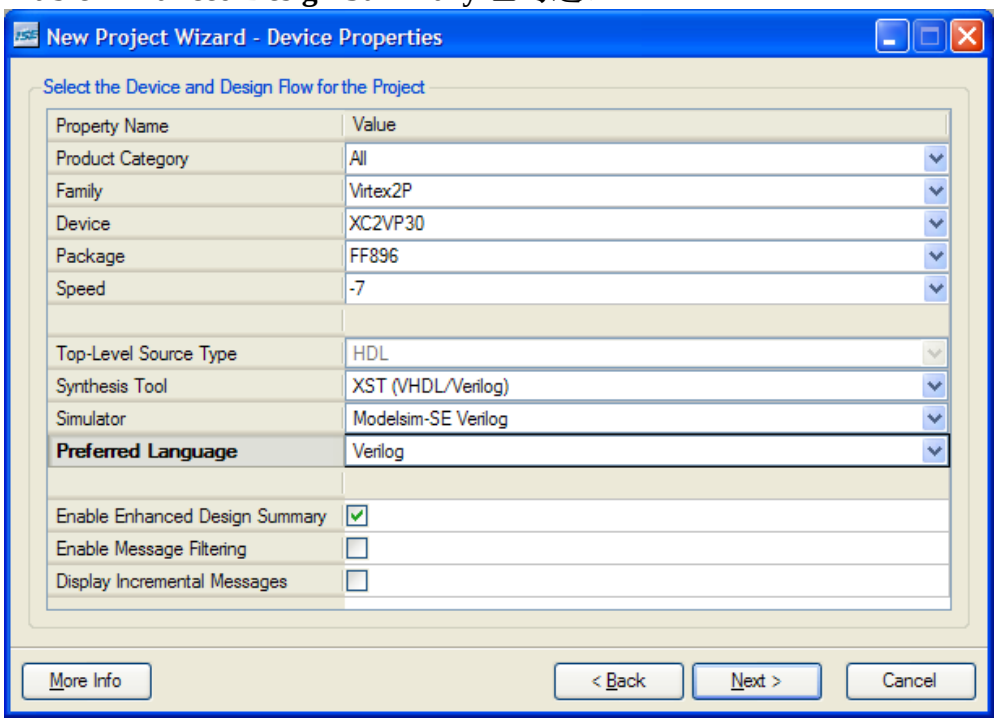
注意实验报告的格式统一，缩进（禁用 Tab）、行间距、字体等。

2. 新建工程

2.1 实验描述

2.1.1 新建工程

1. 启动 ISE 9.1i。
2. 选择 File > New Project… 出现 New Project Wizard。
3. Project Name 填写 lab6，选择工程 Project Location,Top-level Source Type 选择 HDL。点击 Next。
4. Device Properties 中各属性填写如下：
   Product Category: **ALL**
   Family: **Virtex2P**
   Device: **XC2VP30**
   Package: **FF896**
   Speed: **-7**
   Synthesis Tool: **XST(VHDL/Verilog)**
   Simulator: ISE Simulator **Modelsim-SE (verilog)** **[注：这次将用 Modelsim 仿真]**
   Preferred Language: **Verilog**
   确认 **Enable Enhanced Design Summary** 已勾选。



5. 点击 Next。
6. 在 New Project Wizard – Create New Source 中点击 Next。
7. 在 New Project Wizard – Add Existing Sources 中点击 Next。
8. 在 New Project Wizard – Project Summary 中点击 Finish，结束建立工程。

3. TOP 模块

3.1 实验描述

分别实现 Memory, Decoder, ALU, Register Files, Data Hazard。

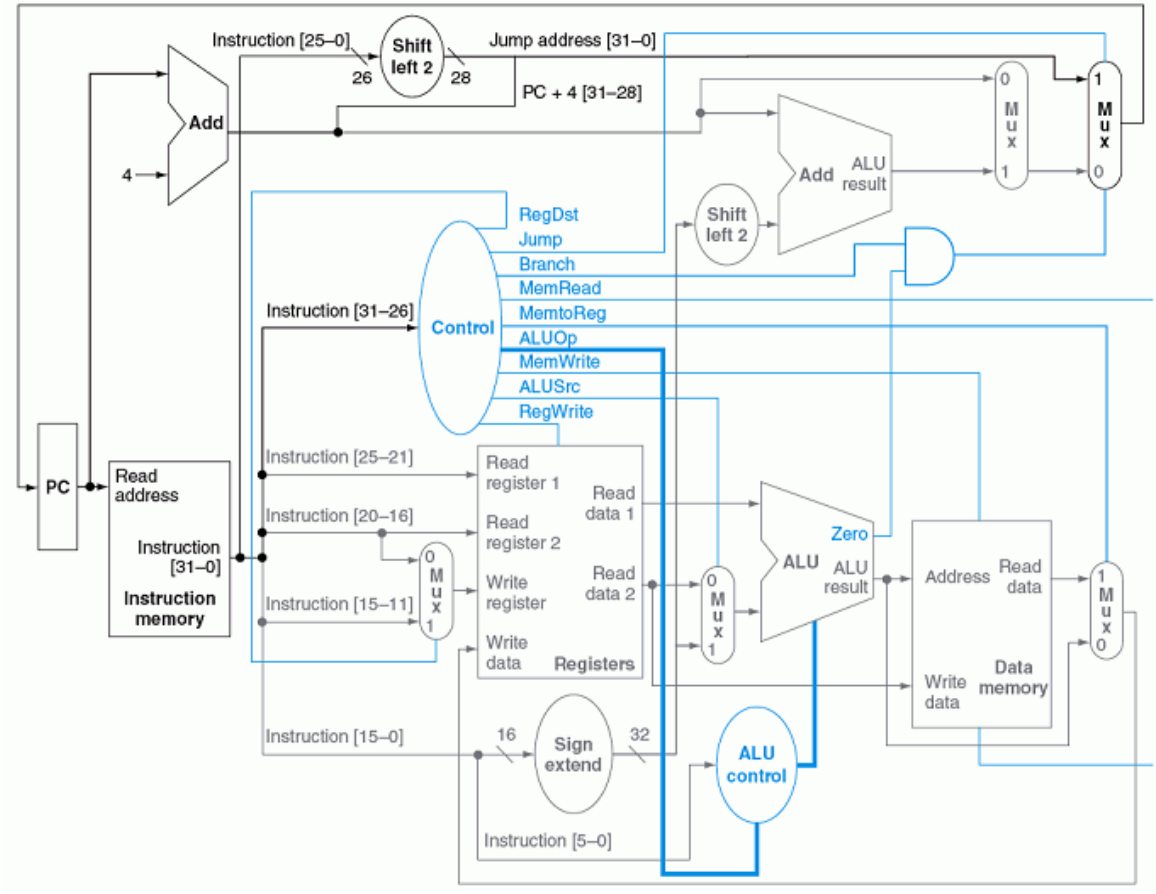
如果有时间可以在此基础上实现分支预测和 L1 Cache 的主要功能。

- 基本模块的实现，插入级间寄存器
- Top 文件编写
- Forwarding 的实现
- Stall 的实现

3.1.1 基本模块

3.1.1.1 单周期结构：

下面是单周期 MIPS 的主要结构：



其中逻辑如下：

一、 译码

Input or output	Signal name	R-format	lw	sw	beq
Inputs	Op5	0	1	1	0
	Op4	0	0	0	0
	Op3	0	0	1	0
	Op2	0	0	0	1
	Op1	0	1	1	0
	Op0	0	1	1	0
Outputs	RegDst	1	0	X	X
	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

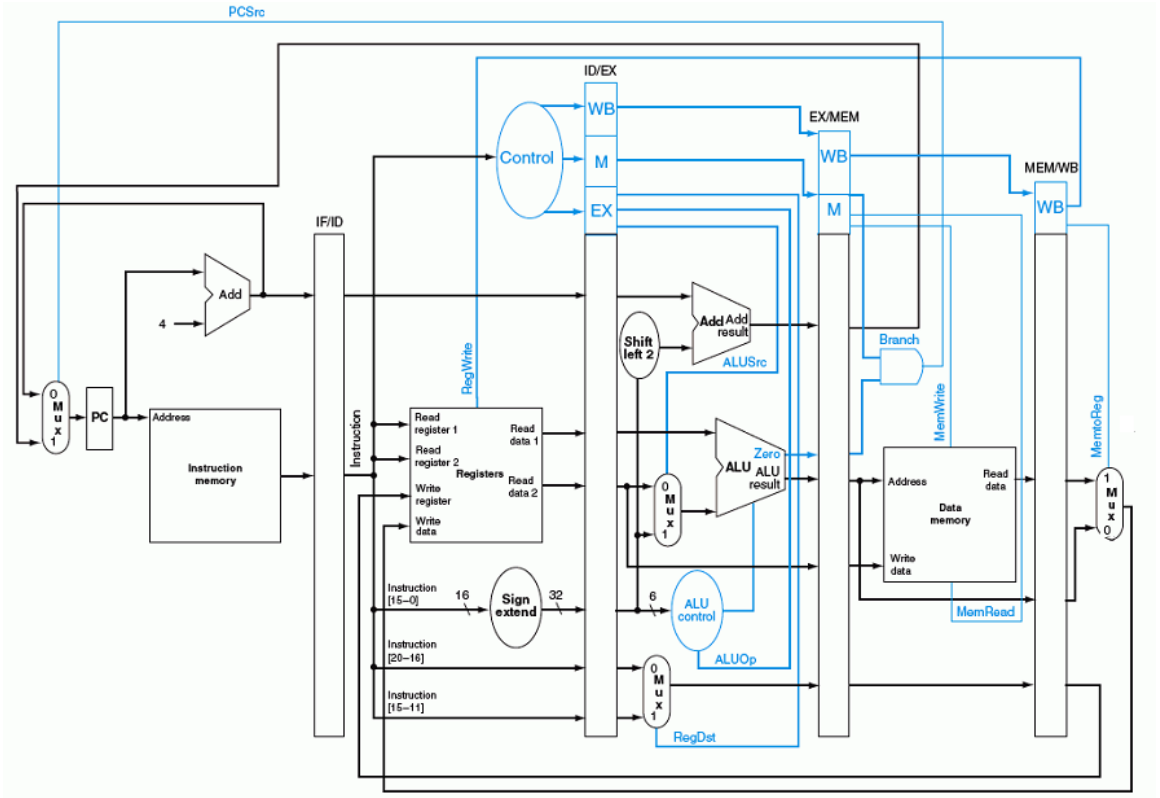
二、 ALUCtrl

Instruction opcode	ALUOp	Instruction operation	Funct field	Desired ALU action	ALU control input
LW	00	load word	XXXXXX	add	0010
SW	00	store word	XXXXXX	add	0010
Branch equal	01	branch equal	XXXXXX	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	and	0000
R-type	10	OR	100101	or	0001
R-type	10	set on less than	101010	set on less than	0111

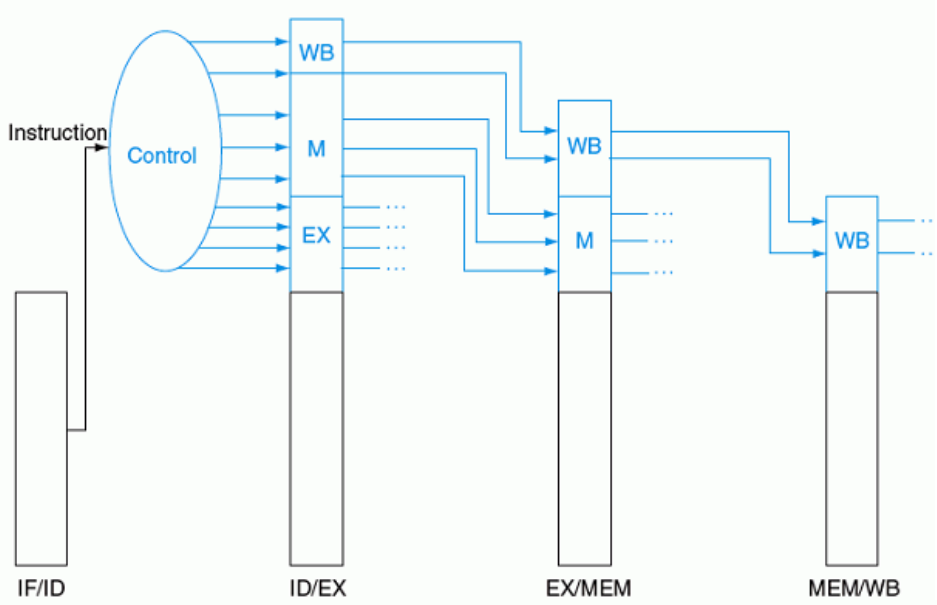
ALUOp		Funct field						
ALUOp1	ALUOp0	F5	F4	F3	F2	F1	F0	Operation
0	0	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	0110
1	X	X	X	0	0	0	0	0010
1	X	X	X	0	0	1	0	0110
1	X	X	X	0	1	0	0	0000
1	X	X	X	0	1	0	1	0001
1	X	X	X	1	0	1	0	0111

3.1.1.2 多周期基本结构：

将单周期 CPU 进行分割，插入 4 级寄存器，将其分割为 IF，ID，EX，M，WB 五大部分：



其中 Control 的输出需要被保存下来，以供后续每级流水使用。如下所示：



Instruction	Execution/address calculation stage control lines				Memory access stage control lines			Write-back stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg Write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

3.1.2 Forwarding

首先，分为两类 Data Hazard：

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

如：

```
Sub $2, $1, $3
And $12, $2, $5
EX/MEM.RegisterRd = ID/EX.RegisterRs = $2
```

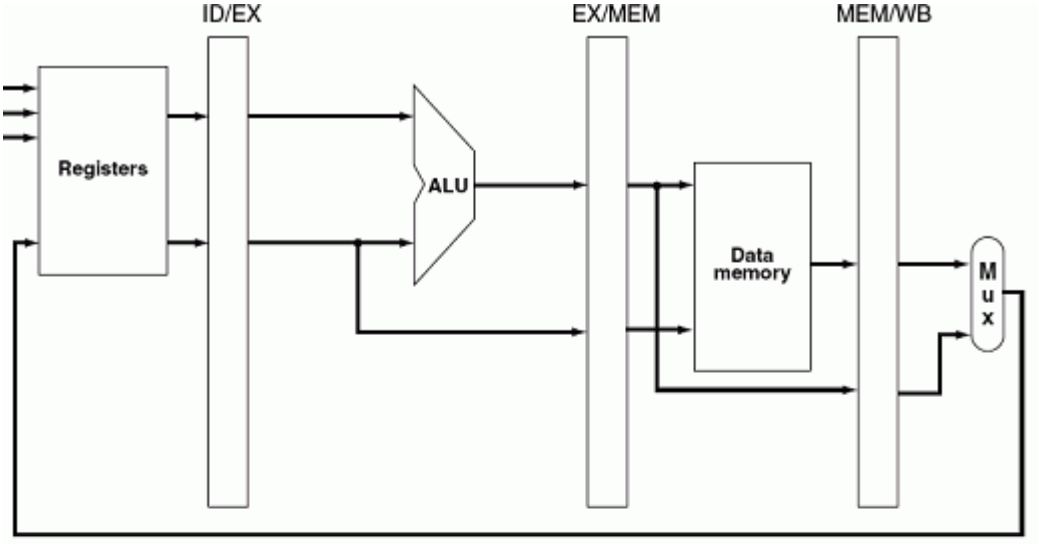
Mux control	Source	Explanation
ForwardA = 00	ID/EX	The first ALU operand comes from the register file.
ForwardA = 10	EX/MEM	The first ALU operand is forwarded from the prior ALU result.
ForwardA = 01	MEM/WB	The first ALU operand is forwarded from data memory or an earlier ALU result.
ForwardB = 00	ID/EX	The second ALU operand comes from the register file.
ForwardB = 10	EX/MEM	The second ALU operand is forwarded from the prior ALU result.
ForwardB = 01	MEM/WB	The second ALU operand is forwarded from data memory or an earlier ALU result.

(一) EX 级，1a 和 1b：

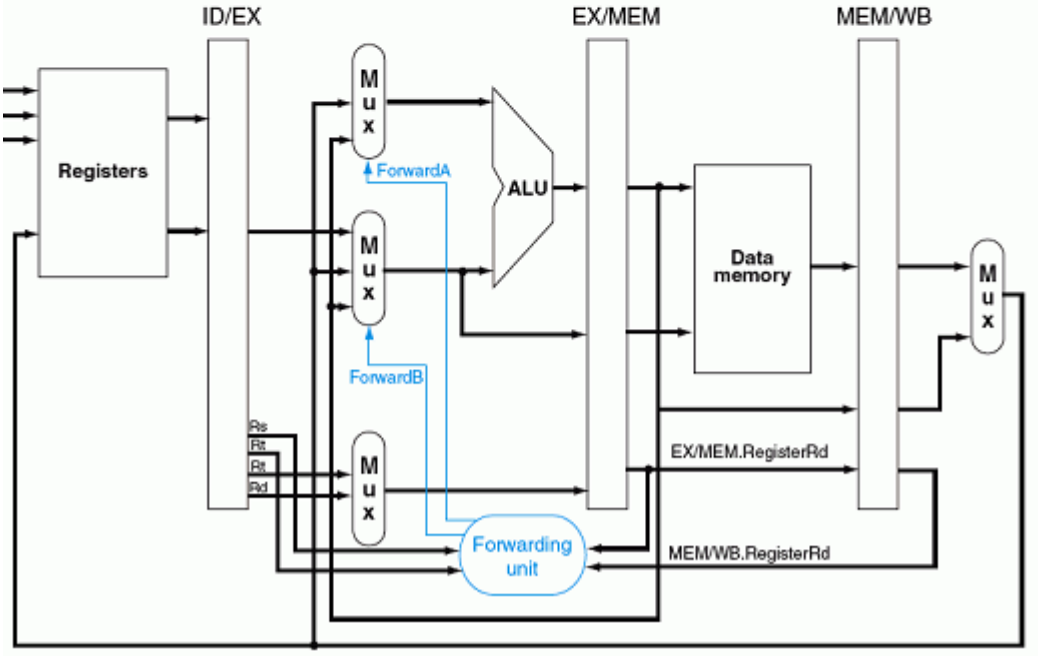
```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs)) ForwardA = 10

if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt)) ForwardB = 10
```

其中，ForwardA，ForwardB 参见下面的结构图：



a. No forwarding



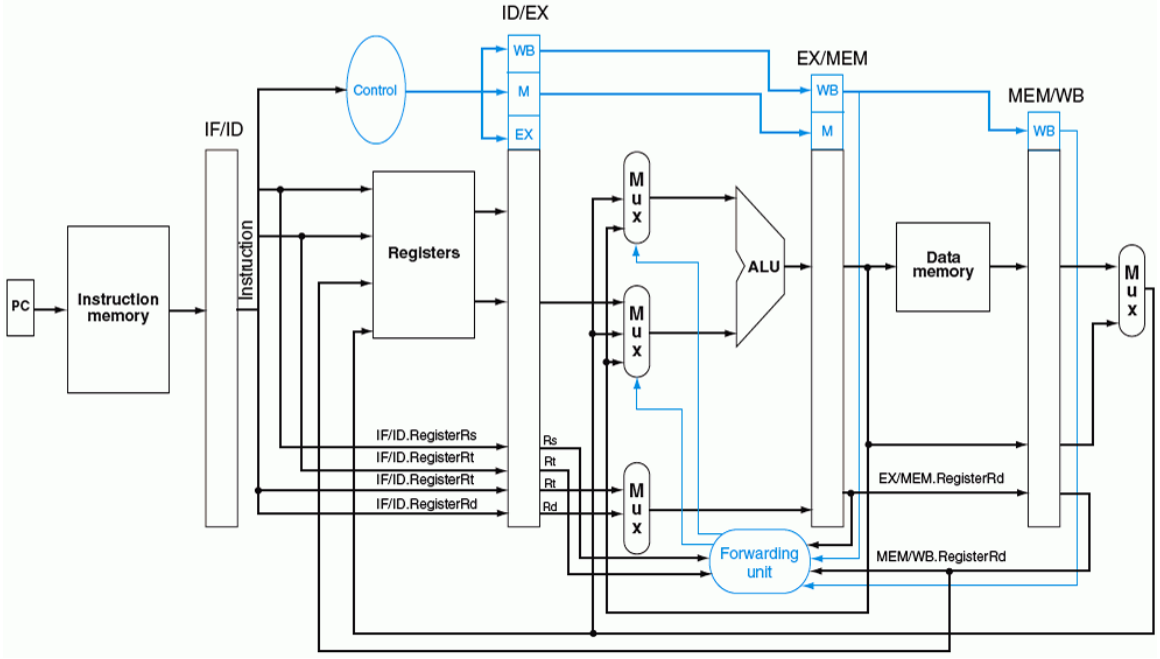
b. With forwarding

(二) MEM 级 2a,2b：

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs)) ForwardA = 01

if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd ≠ 0)
and (EX/MEM.RegisterRd ≠ ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt)) ForwardB = 01
```

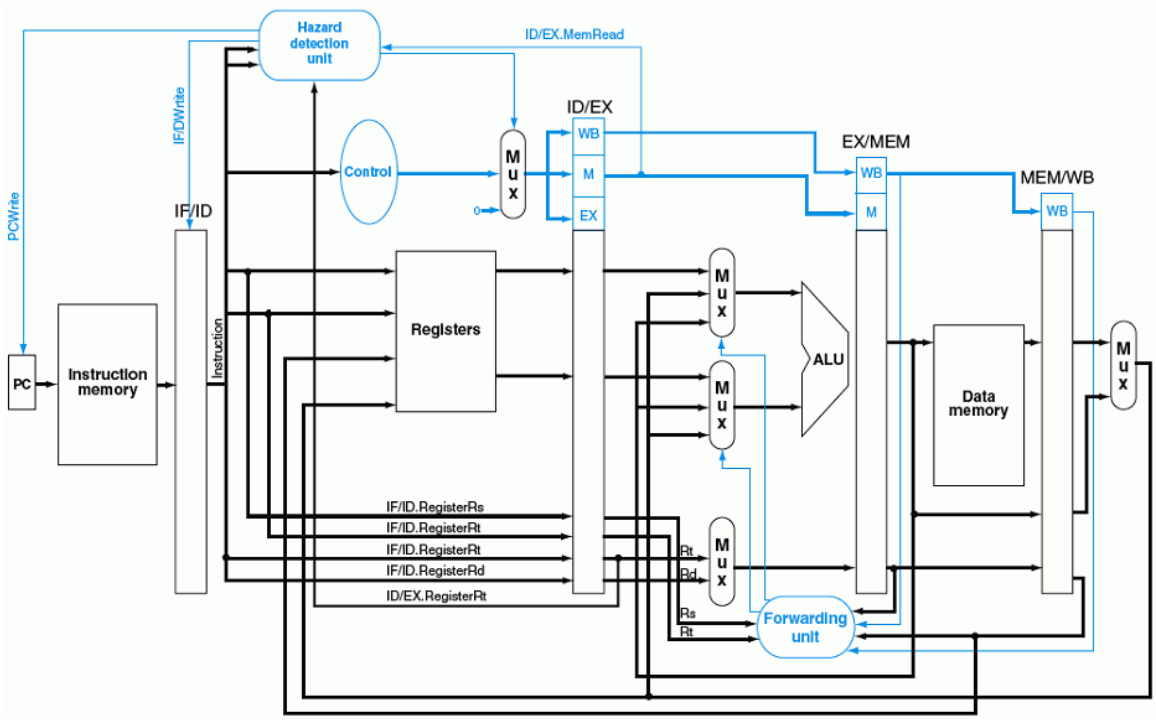
结构图如下:



3.1.3 Stalls 的实现：

```
if (ID/EX.MemRead and
   ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
      stall the pipeline
```

Stall 的实现通过插入 nops，可通过 mux 选择来源为本级输出或上级级间寄存器。结构图：



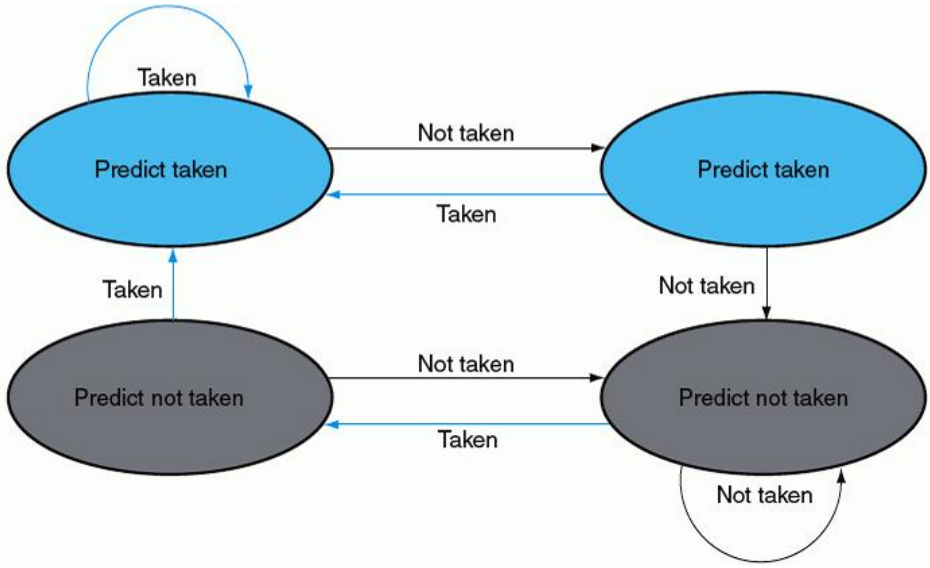
3.1.4 分支预测（非必需实验）

需要加入 flush 流水线的功能。

(一) 静态分支预测

- Assume Branch Taken
- Assume Branch Not Taken

(二) 动态分支预测（1 位或 2 位预测器）

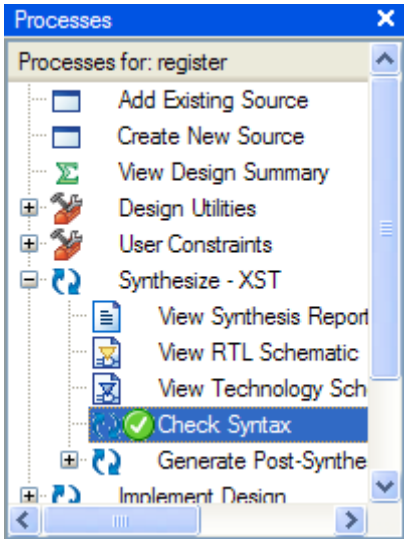


3.1.5 编写功能

注意，由于各种变量名称极为复杂，推荐在着手编码之前为自己选择一套命名规范。

另外，由于 MEM 级的 Branch 会影响 PCSrc 的值，从而影响下次 PC，因此需要为 Control 加入 RESET 功能，将 Branch 置零。

写完代码后在综合选项中运行语法检查。



3.1.6 代码指导

3.1.6.1 命名规范

所有寄存器、线、模块名称应符合统一的命名规范，不做硬性规定，但命名应当能够反映功能与类型，并在整个工程中保持统一。如：ID_EX_WriteMem 等。

3.1.6.2 模块划分

顶层模块 top 下，分别设计 IF，ID，EX，MEM，WB 五个子模块，模块中包括逻辑功能模块与栈间寄存器模块（如果包含）。

3.1.6.3 接口定义

在工程设计初期，应当协同确定各个模块的功能、划分，并且确定模块间的接口与时序。接口定义必须在编写代码前完成，并给出接口定义文档。

3.1.6.4 栈间寄存器

栈间寄存器的编写应当注意触发条件以及对流水线 Stall 的考虑。流水线应对 Stall 可以有两种方式，一种是栈间寄存器保持原值，另一种是清空前面的流水级。（考虑一下这两种方法各自有什么优缺点）

这里给出一个使用 Reset 前几级栈间寄存器的例子，实际编写时根据需要编写自己的栈间寄存器。

```
module reg_b #
(
    parameter WIDTH = 32,
    parameter INIT_VALUE = 'b0
)
(
    clk,//input
    rst,//input
    d,//input
    q//input
);

input clk;
input rst;
input[WIDTH-1:0] d;
output[WIDTH-1:0] q;
reg[WIDTH-1:0] q;

always@(posedge clk) begin
    `ifdef RST_AT_HIGH
        if(rst == 1) q<=INIT_VALUE;
    `else
        if(rst == 0) q<=INIT_VALUE;
    `endif
        else q<=d;
end

endmodule
```

使用 reg_b 类型可以方便创建栈间寄存器的实例：

```
reg_b #
    (
        .WIDTH(2)
    )
    WB
    (
        .clk(clk),
        .rst(rst),
        .d(WB_in),
        .q({RegWrite_out,MemtoReg_out})
    );
```

其中，rst 用于栈间寄存器的清零，RST_AT_HIGH 表示高位清零。

3.1.6.5 ID 模块中寄存器堆

零号寄存器永远为 0x0。注意，由于寄存器堆的输入并非同时到达，因此需要使用时钟下降沿进行寄存器写操作以保证写操作在 WB 阶段一个周期内完成。同时，在系统启动时，应当将寄存器初始化为零。这里提供一个样例，请根据功能自行填充代码，以实现读、写功能：

```
module register_set(
       clk,//input
       rst,//input
      //Other Inputs or Outpus…
  );
  input  clk;
  input rst;

  // Other in/outputs
```

```
  reg[`WORDWIDTH-1:0] gpreg[31:0]; //32 registers in MIPS CPU
  always@(negedge clk) begin
      if(rst == 1) begin
            gpreg[0]<=0;
            gpreg[1]<=0;
            ....
      end
      else if(RegWrite==1) begin
            .....
      end
  end
end module
```

3.1.7 仿真测试

1. 编写汇编代码，手工计算或者使用我们提供的翻译器转化为二进制代码文件。如下面的代码：

```
lw $1, 40($0)     ; 1
lw $2, 44($0)     ; 5
lw $3, 48($0)     ; 8
add $4, $1, $2    ; $4=6
sub $5, $3, $1    ; $5=7
and $6, $2, $1    ; $6=1
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
or $7, $3, $1     ; $7=9
slt $8, $3, $1    ; $8=0
beq $0, $0, end   ; to end
add $9, $7, $8    ; $9=9, not executed
end:
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
lw $10, 40($0)    ; 1
```

2. 将上述代码转为二进制 Codes，保存为文件，文件名自定，这里假定是 result.txt，将数据

```
1
5
8
```

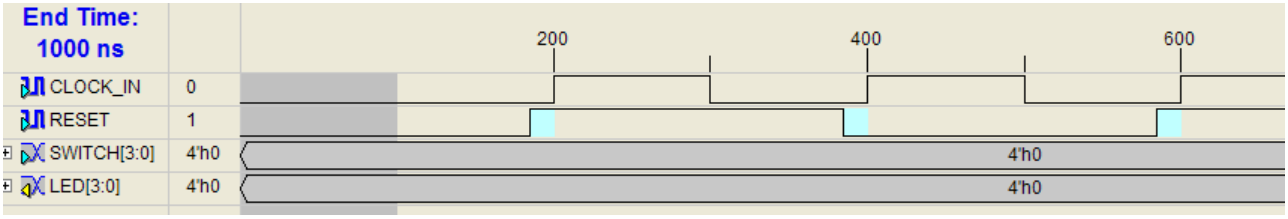
保存为 data.txt。

然后在 Top 中加入下面面代码：

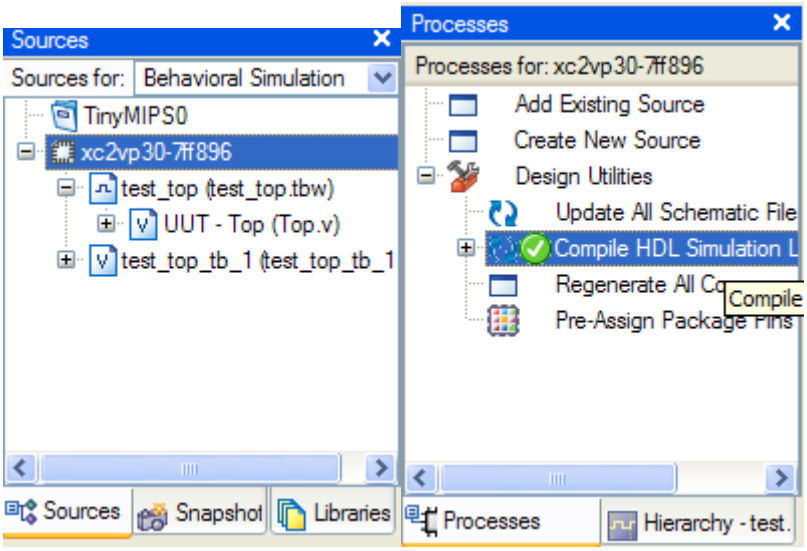
```
initial
      $readmemb("result.txt", instMem.membuf, 8'h0);
      $readmemh("data.txt", dataMem.membuf, 8'ha);
end
```

3. 在 Sources 窗口中，将 Sources for 下拉框选择为 Behavioral Simulation
4. 在 Top(Top.v)模块上点击右键，选择 New Source。
5. 文件类型为 Test Bench WaveForm，文件名自定
6. Associate Source 中选择 Top，Next..

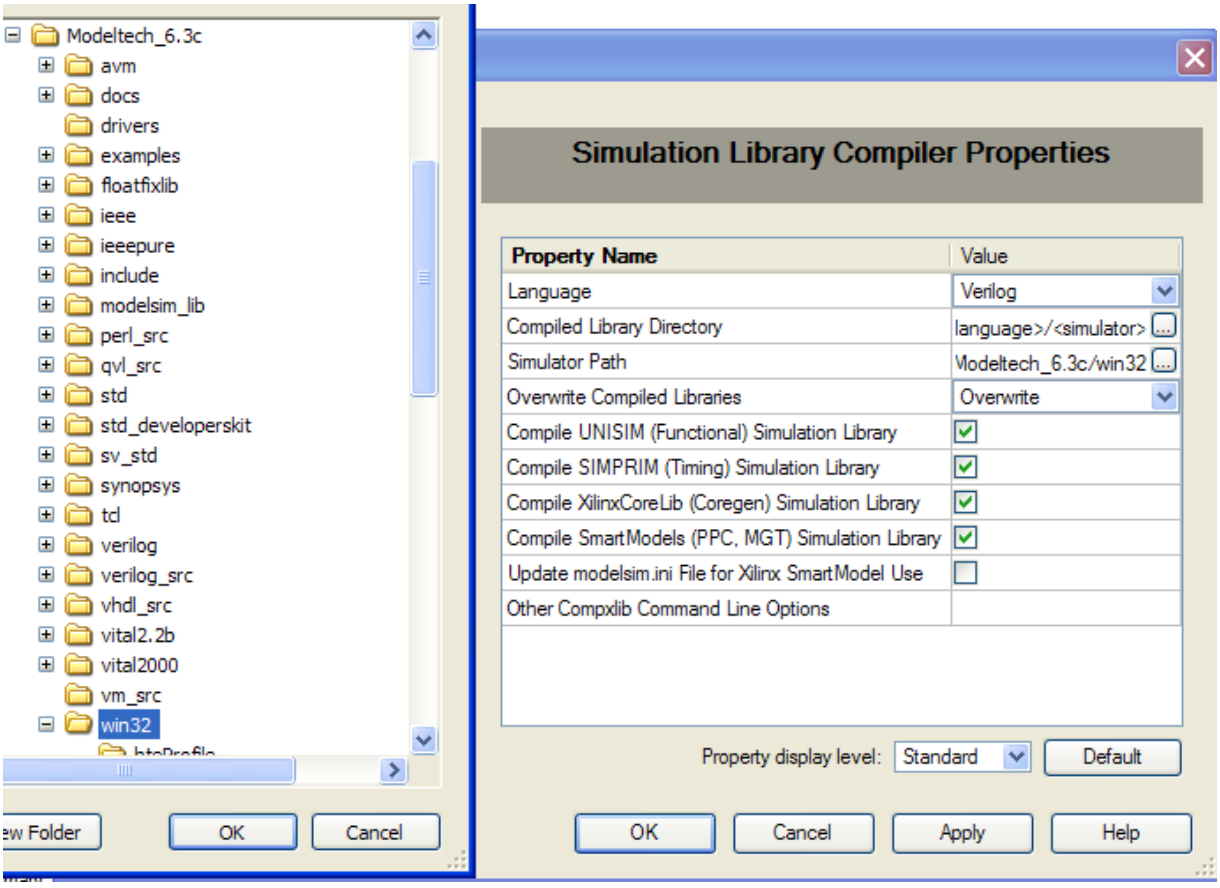
7. Finish
8. 弹出的 Initial Timing and Clock Wizard – Initialize Timing 中，使用 clock_in 作为时钟输入，仿真周期自定
9. 设置输入波形。下面给出一个样例：



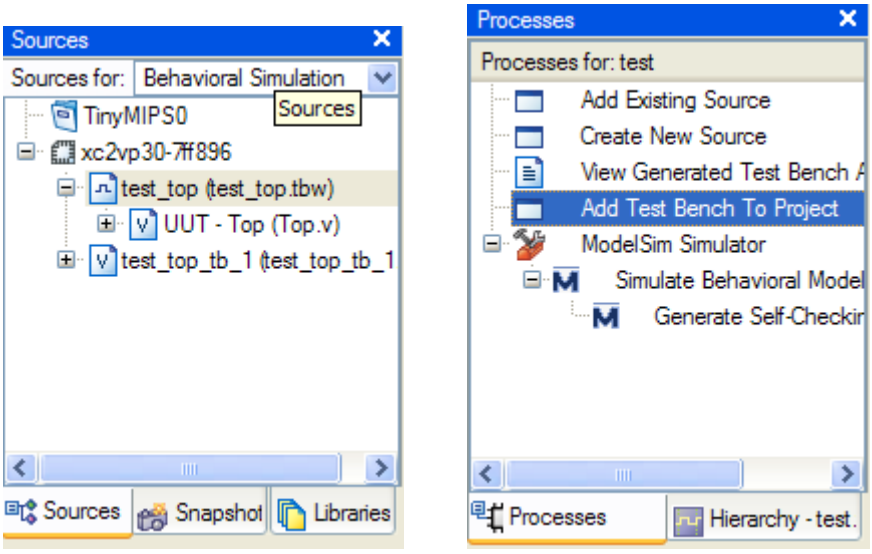
10. 选中 Sources 中的设备，在 Processes 中运行 Compile HDL Simulation Lib:



如若找不到 excutable simulator,右键选中图中 Compile HDL Simulation Lib, 点击属性



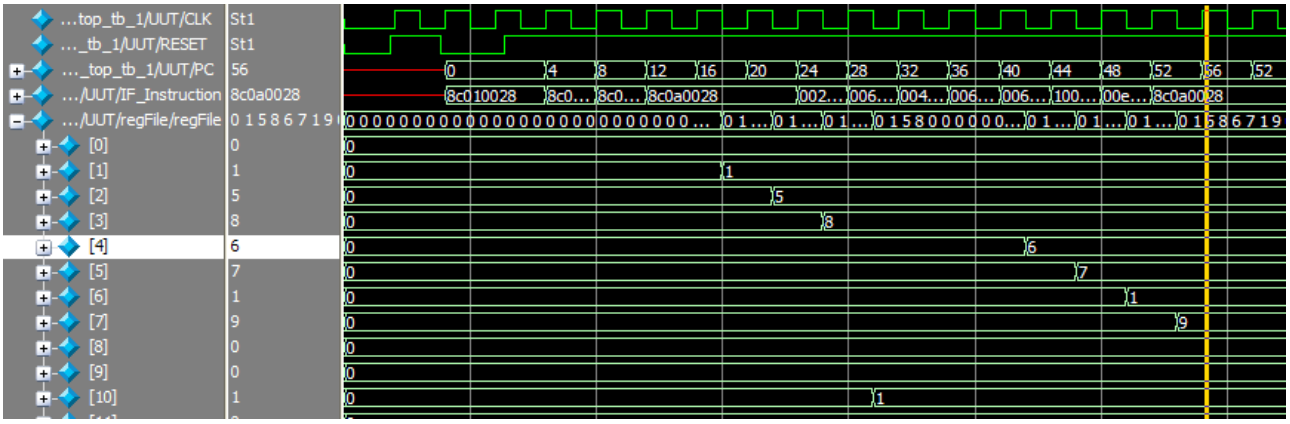
11. 将 testbench 添加至工程，如下图：



12. 双击 Sources 中出现的 test_tb_x.v 文件，可以查看具体的代码，添加图中标注的代码修改输入。这里修改的目的是，将 reset 信号的作用时间包含初始化和 PC 指令两部分，并且 PC 置零在初始化之后。如不需，也可不修改。

```
48      initial begin
49          // -------------  Current Time:  185ns
50          #185;
51          RESET = 1'b1;
52          // ------------------------------------
53          // -------------  Current Time:  385ns
54          #200;
55          RESET = 1'b0;
56          // ------------------------------------
57          // -------------  Current Time:  585ns
58          #250;
59          RESET = 1'b1;
60          // ------------------------------------
61      end
```

13. 运行仿真，可以看到仿真结果（操作小技巧：小键盘 -， +，快速缩放波形视野）。检查波形，如果有错，检查代码逻辑，重新仿真。



3.2 实验验证代码

计算每个寄存器的值并验证，画出流水时序图：

```
lw $1, 40($0)          ;$1=
lw $2, 44($0)          ;$2=
add $3, $1, $2         ;$3=
sub $4, $2, $1         ;$4=
or $5, $3, $4          ;$5=
slt $5, $5, $1         ;$5=
sw $3, 52($5)          ;
and $5, $3, $4         ;$5=
lw $6, 48($0)          ;$6=
beq $5, $6, end
add $5, $5, $5         ;$5=
end:
sw $5, 44($4)
```

数据

```
Address=40, Data=1
Address=44, Data=5
Address=48, Data=4
```