A Verilog Behavioral Simulation (Continued)

- The register transfer operation is the mechanism that implements the pipelining.
- There are two register transfer operations.

```
module enreg( clk, en, d, q);
   parameter width=1;
   input clk, en;
   input [width-1:0] d;
   output [width-1:0] q;
   reg [width-1:0] q;

   initial q = 0;

   always @(posedge clk)
      if ( en) q = d;
endmodule
```

```
module enreg_with_clear( clk,
en, clr, d, q);
   parameter width=1;
   input clk, en, clr;
   input [width-1:0] d;
   output [width-1:0] q;
   reg [width-1:0] q;

   initial q = 0;

   always @(posedge clk)
      if ( en)
         if ( clr)
            q = 0;
         else
            q = d;
endmodule
```


© 1996 LS All Rights Reserved

The mips module

```
module mips;
  wire clk;

  clock clock_0( clk);
  defparam clock_0.period=11;
  defparam clock_0.cycles=100;

  ifetch ifetch();
  id id();
  ex ex();
  mem mem();
     <fill pipe regs>
```

```
module clock( clk);
   parameter period = 11;
   parameter cycles = 45;
   output clk;
   reg clk;
   initial begin
      clk = 0;
      #(period)
      repeat (cycles) begin
         clk = 1;
         #(period/2)
         clk = 0;
         #(period-period/2) ;
      end
      #(period)
      $finish;
   end
endmodule
```


© 1996 LS All Rights Reserved

Filling ifetch Pipeline Reg

```
   enreg #(32) wreg_0( clk, !id.Stall | mem.Branch & mem.Zero,
                       mem.nextpc, ifetch.pc);

   enreg #(32) if_id_wreg_0( clk, !id.Stall, ifetch.pc4, id.pc4);
   enreg #(32) if_id_wreg_1( clk, !id.Stall, ifetch.ir, id.ir);
//
// id.IFflush means interpret id.ir as a nop
//
   enreg #(1)  if_id_wreg_2( clk, 1'b1, id.flush, id.IFflush);
```

```
tmpindex = (mips.ex.RegDst == 0) ? mips.ex.Rt : mips.ex.Rd;
Stall = mips.ex.RegWrite & tmpindex != 0 &
           ( tmpindex == Rs | tmpindex == Rt & Op != 6'b100011 )
      | mips.mem.RegWrite & mips.mem.windex != 0 &
           ( mips.mem.windex == Rs | mips.mem.windex == Rt
                  & Op != 6'b100011 );
```

In English: There is a data hazard if the instruction in the `ex` stage or the `mem` stage that will be writing to an operand register needed by this instruction, that is Rs or Rt, except that if this is a lw than Rt is the destination of the memory anyway.

From the test program ...

```
add $4, $0, $0   mem
add $5, $0, $0   ex
slt $8, $3, $4   id
```


© 1996 LS All Rights Reserved

Filling from id Pipeline Reg

```
   enreg #(32) id_ex_wreg_0( clk, 1'b1, id.pc4, ex.pc4);
   enreg #(32) id_ex_wreg_1( clk, 1'b1, id.rdata1, ex.rdata1);
   enreg #(32) id_ex_wreg_2( clk, 1'b1, id.rdata2, ex.rdata2);
   enreg #(32) id_ex_wreg_3( clk, 1'b1, id.offset, ex.offset);
   enreg #(5)  id_ex_wreg_4( clk, 1'b1, id.Rt, ex.Rt);
   enreg #(5)  id_ex_wreg_5( clk, 1'b1, id.Rd, ex.Rd);
//
// On the next edge, insert a nop into the EX stage if there is a data hazard,
//    a mispredicted branch is in the MEM stage, or
//    a mispredicted branch was in the MEM stage in the last cycle
//
   enreg_with_clear #(9) id_ex_wreg_6( clk, 1'b1,
                           id.IFflush | id.flush | id.Stall,
      {id.ALUOp,id.RegDst,id.ALUSrc,
       id.Branch,id.MemRead,id.MemWrite,
       id.MemToReg,id.RegWrite},
      {ex.ALUOp,ex.RegDst,ex.ALUSrc,
       ex.Branch,ex.MemRead,ex.MemWrite,
       ex.MemToReg,ex.RegWrite});
```

From id stage ...

```
always @(mips.mem.Branch or mips.mem.Zero)
   flush = mips.mem.Branch & mips.mem.Zero;
```


© 1996 LS All Rights Reserved

Filling from ex Pipeline Reg

```
    enreg #(32) ex_mem_wreg_0( clk, 1'b1, ex.branchpc, mem.branchpc);
    enreg #(32) ex_mem_wreg_1( clk, 1'b1, ex.result, mem.result);
    enreg #(32) ex_mem_wreg_2( clk, 1'b1, ex.rdata2, mem.wdata);
    enreg #(1) ex_mem_wreg_3( clk, 1'b1, ex.Zero, mem.Zero);
    enreg #(5) ex_mem_wreg_4( clk, 1'b1, ex.windex, mem.windex);
//
// On the next edge, insert a nop into the MEM stage if there is
//    a mispredicted branch is in the MEM stage.
//
    enreg_with_clear #(5) ex_mem_wreg_5( clk, 1'b1, id.flush,
       {ex.Branch,ex.MemRead,ex.MemWrite,
        ex.MemToReg,ex.RegWrite},
       {mem.Branch,mem.MemRead,mem.MemWrite,
        mem.MemToReg,mem.RegWrite});
```


© 1996 LS All Rights Reserved