# 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 pipeline registers (later slides)>
```

# The `mips` module

```
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
```

# The Pipeline

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

# Setting the Pipeline Registers

```
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
```

# Getting the next PC & Filling the IF/ID 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 );
```

From id stage ...

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

# Filling the ID/EX 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 );
```

# Filling the ID/EX Pipeline Reg

```
//
// 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;
```

# Filling the EX/ MEM 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 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} );
```

# Initializing

```
Initial begin
        $monitor( "%0d pc = %h", $time, ifetch.pc );
        $readmemb( "prog.bin", ifetch.imem_0.a );
        $readmemb( "data.bin", mem.dmem_0.a );
end
```