Advanced Pipelining Concepts

- **Instruction level parallelism (ILP)**
  - Superscalar (more than one instruction issued per cycle)
  - Superpipelining (smaller pipeline cycles)

- **Other Techniques**
  - Out-of-order issue, out-of-order completion
  - Register renaming, multi-level branch history table

- **From ILP to TLP (thread-level parallelism)**
  - Simultaneous Multithreading
  - Multi-cores
Superscalar

- The major innovation of the superscalar architecture over the traditional pipelined processor is the multiple instruction issue advantage through dynamic instruction scheduling.
Putting it all together: Pentium-4 Pipeline

IBM PowerPC 970
Project Tips & Requirements

- Design your pipelined processor as 5-stages
  - Fetch, Decode, Execute, Memory, Writeback

- Output of every stage available at the negative edge of the CLK
  - Inputs to each stage are just wires

- Test each module individually

- Modifications to register file
  - Add up to 5 outputs ($1 to $5)
  - Use only these registers to test program

- Submit a professionally typed report with waveforms, code and instructions implemented
  - Due by December, 6th 2013 (20% of final grade)
Example: Instruction Decode

Verilog Code for ID Stage: INPUTS

```verilog
module ID(In_PC, In_IR, In_Rd, In_WriteData, In_RegWrite,
        Out_BranchPC, Out_PCSrc, Out_DataA, Out_DataB, Out_SE,
        Out_Funct, Out_Rs, Out_Rt, Out_Rd, Out_EXControl,
        Out_MEMControl, Out_WBControl, Clk);

  input [31:0] In_PC;  // Program counter
  input [31:0] In_IR;  // Instruction register
  input [31:0] In_WriteData;  // Address for writing
  input [31:0] In_RegWrite;  // Control line for writing into register
  input Clk;

  // The following two need not be registers...
  // and they are not assigned at the negative clock edge.

  output [31:0] Out_BranchPC;  // Branch target address or jump address depending on
                               // the instruction that is decoded.
  output Out_PCSrc;  // PCSrc control input to the multiplexor in the IF stage.
```

Verilog Code for ID Stage: OUTPUTS

```
output [31:0] Out_DataA;    // Read Register A
output [31:0] Out_DataB;    // Read Register B
output [31:0] Out_SE;       // Sign extended value
output [5:0] Out_Funct;     // 6 bit funct field

output [4:0] Out_Rs;        // Rs register address In.IR[25:21]: For use in forwarding logic.
output [4:0] Out Rt;         // Rt register address In.IR[20:16]:
output [4:0] Out.Rd;         // Rd register address In.IR[15:11]:


output [1:0] Out_MEMControl; // Control signals for MEM stage. [1] MemWrite, [0] MemRead;

output [1:0] Out_WBControl; // Control signals for WB stage. [1] MemToReg, [0] RegWrite
```

Verilog Code for ID Stage: REGISTERS

```
reg [31:0] Out_DataA;    // Read Register A
reg [31:0] Out_DataB;    // Read Register B
reg [31:0] Out_SE;       // Sign extended value
reg [5:0] Out_Funct;     // 6 bit funct field

reg [4:0] Out_Rs;        // Rs register address In.IR[25:21]: For use in forwarding logic.
reg [4:0] Out.Rt;         // Rt register address In.IR[20:16]:
reg [4:0] Out.Rd;         // Rd register address In.IR[15:11]:


reg [1:0] Out_MEMControl; // Control signals for MEM stage. [1] MemWrite, [0] MemRead;

reg [1:0] Out_WBControl; // Control signals for WB stage. [1] MemToReg, [0] RegWrite
```
Verilog Code for ID Stage: CODE

```verilog
wire [31:0] DataA, DataB, SE;
wire PCSrc, wire ALUSrc;
wire [1:0] ALUOp;
wire RegDst, MemWrite, MemRead, Branch, MemToReg, RegWrite;
wire [5:0] Funct;
wire [3:0] EXControl;
wire [1:0] MEMControl, WBControl;

RegisterFile RF(In_IR[25:21], DataA, In_IR[20:16], DataB, In_Rd, In_WriteData, In_RegWrite, Clk);

Control Ctrl(In_IR[31:26], In_IR[5:0], ALUSrc, ALUOp, RegDst, MemWrite, MemRead, Branch, MemToReg, RegWrite, Funct);
SignExtension SE3(In_IR[15:0], SE);

assign EXControl = {ALUSrc, ALUOp[1:0], RegDst};
assign MEMControl = {MemWrite, MemRead};
assign WBControl = {MemToReg, RegWrite};
assign Out_BranchPC = In_PC + (SE << 2);
// Branch target address computation
assign Out_PCSrc = (Branch == 1) & (DataA == DataB);
```

Verilog Code for ID Stage: @NEG

```verilog
// Assign the outputs at the negative edge of the clock.
// You do not want the outputs to settle after the
// positive edge.
always @(negedge Clk) begin
Out_DataA = DataA;
Out_DataB = DataB;

Out_SE = SE;
Out_RS[4:0] = In_IR[25:21];
Out_RT[4:0] = In_IR[20:16];
Out_RD[4:0] = In_IR[15:11];

Out_EXControl = EXControl;
Out_MEMControl = MEMControl;
Out_WBControl = WBControl;
Out_Funct = Funct;
end
endmodule
```