Project 4

Kinner Parikh

November 17, 2022

CMPEN 331 - 001

1 Code

```
‘timescale 1ns / 1ps
module PC(
    input [31:0] nextPc,
    input clk,
    output reg [31:0] pc
);
    initial pc = 32'd100;

    always @(posedge clk)
    begin
        pc = nextPc;
    end
endmodule

module InstructionMemory(
    input [31:0] pc,
    output reg [31:0] instOut
);
    reg [31:0] memory [63:0];

    initial begin
        memory[25] = {6'b100011, 5'b00001, 5'b00010, 16'h0000};
        memory[26] = {6'b100011, 5'b00001, 5'b00011, 16'h0004};
        memory[27] = {6'b100011, 5'b00001, 5'b00100, 16'h0008};
        memory[28] = {6'b100011, 5'b00001, 5'b00101, 16'h000c};
    end

    always @(*)
        instOut <= memory[pc[7:2]];
endmodule

module PCAdder(
    input [31:0] pc,
    output reg [31:0] nextPc
);
    always @(*)
        nextPc <= pc + 4;
endmodule

module IFIDPipelineReg(
    input [31:0]instOut,
    input clk,
    output reg [31:0] dinstOut
);
    always @(posedge clk)
        dinstOut <= instOut;
endmodule

module ControlUnit(
    input [5:0] op, func,
    output reg wreg, m2reg, wmem, aluimm, regrt,
    output reg [3:0] aluc
);
    always @(*)
    begin
        case(op)
            ’b000000: //r-type
             begin
                wreg   = ’b1;
                m2reg  = ’b0;
                wmem   = ’b0;
                aluimm = ’b0;
                regrt  = ’b0;
                case(func)
                    ’b100000: //add
                    begin
                        aluc = ’b0010;
                    end
                    ’b100010: //sub
                    begin
                        aluc = ’b0110;
                    end
                endcase
             end
             ’b100011: //lw
             begin
                wreg   = ’b1;
                m2reg  = ’b1;
                wmem   = ’b0;
                aluc   = ’b0010;
                aluimm = ’b1;
                regrt  = ’b1;
             end
         endcase
    end
endmodule

module RegrtMux(
    input [4:0] rt, rd,
    input regrt,
    output reg [4:0] destReg
);
    always @(*)
    begin
        if (regrt == 0)
            destReg = rd;
        else
            destReg = rt;
    end
endmodule


module RegFile(
    input [4:0] rs, rt,
    output reg [31:0] qa, qb
);
    reg [31:0] registers[0:31];
    initial begin
        registers[0] = 0;
        registers[1] = 0;
    end

    always @(*)
    begin
        qa = registers[rs];
        qb = registers[rt];
    end
endmodule

module ImmExtender(
    input [15:0] imm,
    output reg [31:0] imm32
);
    always @(*)
```

```
        imm32 = {{16{imm[15]}}, imm};
endmodule

module IDEXEPipelineReg(
    input wreg, m2reg, wmem, aluimm, clk,
    input [3:0] aluc,
    input [4:0] destReg,
    input [31:0] qa, qb, imm32,

    output reg ewreg, em2reg, ewmem, ealuimm,
    output reg [3:0] ealuc,
    output reg [4:0] edestReg,
    output reg [31:0] eqa, eqb, eimm32
);
    always @(posedge clk)
    begin
        ewreg    = wreg;
        em2reg   = m2reg;
        ewmem    = wmem;
        ealuc    = aluc;
        ealuimm  = aluimm;
        edestReg = destReg;
        eqa      = qa;
        eqb      = qb;
        eimm32   = imm32;
    end
endmodule

module ALUMux(
    input [31:0] eqb, eimm32,
    input ealuimm,
    output reg [31:0] b
);
    always @(*)
    begin
        if (ealuimm == 0)
            b <= eqb;
        else
            b <= eimm32;
    end
endmodule
```

```
module ALU(
    input [31:0] eqa, b,
    input [3:0] ealuc,

    output reg [31:0] r
);
    always @(*)
    begin
        case(ealuc)
            'b0010:
                r <= eqa + b;
            'b0110:
                r <= eqa - b;
        endcase
    end
endmodule

module EXEMEMPipelineReg(
    input clk, ewreg, em2reg, ewmem,
    input [4:0] edestReg,
    input [31:0] r, eqb,

    output reg mwreg, mm2reg, mwmem,
    output reg [4:0] mdestReg,
    output reg [31:0] mr, mqb
);
    always @(posedge clk)
    begin
        mwreg    <= ewreg;
        mm2reg   <= em2reg;
        mwmem    <= ewmem;
        mdestReg <= edestReg;
        mr       <= r;
        mqb      <= eqb;
    end
endmodule

module DataMemory(
    input clk, mwmem,
    input [31:0] mr, mqb,

    output reg [31:0] mdo
);
    reg [31:0] memory[0:31];

    initial begin
        memory[0] = 'hA00000AA;
        memory[1] = 'h10000011;
        memory[2] = 'h20000022;
        memory[3] = 'h30000033;
        memory[4] = 'h40000044;
        memory[5] = 'h50000055;
        memory[6] = 'h60000066;
        memory[7] = 'h70000077;
        memory[8] = 'h80000088;
        memory[9] = 'h90000099;
     end

    //reading from memory
    always @(*)
        mdo <= memory[mr[7:2]];

    // writing to memory
    always @(negedge clk)
    begin
        if (mwmem == 1)
        begin
            memory[mr[7:2]] <= mqb;
        end
    end
endmodule

module MEMWBPipelineReg (
    input clk, mwreg, mm2reg,
    input [4:0] mdestReg,
    input [31:0] mr, mdo,

    output reg wwreg, wm2reg,
    output reg [4:0] wdestReg,
    output reg [31:0] wr, wdo
);
    always @(posedge clk)
    begin
        wwreg    <= mwreg;
        wm2reg   <= mm2reg;
        wdestReg <= mdestReg;
        wr       <= mr;
        wdo      <= mdo;
    end
endmodule
```

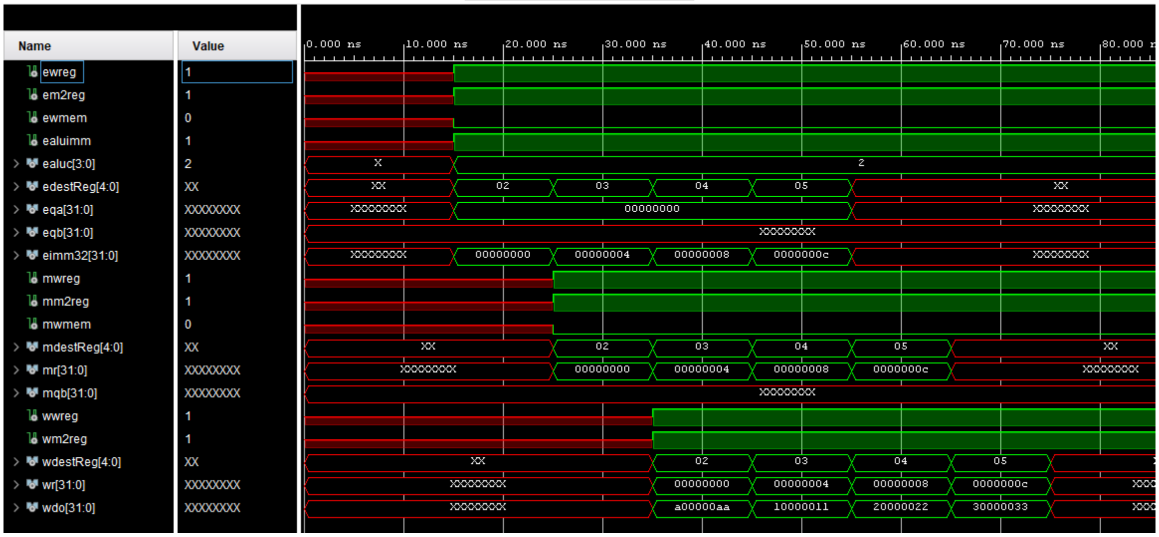
```
‘timescale 1ns / 1ps
module PC(
    input [31:0] nextPc,
    input clk,
    output reg [31:0] pc
);
    initial pc = 32’d100;

    always @(posedge clk)
    begin
        pc = nextPc;
    end
endmodule

module InstructionMemory(
    input [31:0] pc,
    output reg [31:0] instOut
);
    reg [31:0] memory [63:0];

    initial begin
        memory[25] = {6’b100011, 5’b00001, 5’b00010, 16’h0000};
        memory[26] = {6’b100011, 5’b00001, 5’b00011, 16’h0004};
        memory[27] = {6’b100011, 5’b00001, 5’b00100, 16’h0008};
        memory[28] = {6’b100011, 5’b00001, 5’b00101, 16’h000c};
    end

    always @(*)
        instOut <= memory[pc[7:2]];
endmodule

module PCAdder(
    input [31:0] pc,
    output reg [31:0] nextPc
);
    always @(*)
        nextPc <= pc + 4;
endmodule

module IFIDPipelineReg(
    input [31:0]instOut,
    input clk,
    output reg [31:0] dinstOut
);
    always @(posedge clk)
        dinstOut <= instOut;
endmodule

module ControlUnit(
    input [5:0] op, func,
    output reg wreg, m2reg, wmem, aluimm, regrt,
    output reg [3:0] aluc
);
    always @(*)
    begin
        case(op)
            'b000000: //r-type
             begin
                wreg   = 'b1;
                m2reg  = 'b0;
                wmem   = 'b0;
                aluimm = 'b0;
                regrt  = 'b0;
                case(func)
                    'b100000: //add
                    begin
                        aluc = 'b0010;
                    end
                    'b100010: //sub
                    begin
                        aluc = 'b0110;
                    end
                endcase
             end
             'b100011: //lw
             begin
                wreg   = 'b1;
                m2reg  = 'b1;
                wmem   = 'b0;
                aluc   = 'b0010;
                aluimm = 'b1;
                regrt  = 'b1;
             end
         endcase
    end
endmodule

module RegrtMux(
    input [4:0] rt, rd,
    input regrt,
    output reg [4:0] destReg
);
    always @(*)
    begin
        if (regrt == 0)
            destReg = rd;
        else
            destReg = rt;
    end
endmodule


module RegFile(
    input [4:0] rs, rt,
    output reg [31:0] qa, qb
);
    reg [31:0] registers[0:31];
    initial begin
        registers[0] = 0;
        registers[1] = 0;
    end

    always @(*)
    begin
        qa = registers[rs];
        qb = registers[rt];
    end
endmodule

module ImmExtender(
    input [15:0] imm,
    output reg [31:0] imm32
);
    always @(*)
        imm32 = {{16{imm[15]}}, imm};
endmodule
```

```
module IDEXEPipelineReg(
    input wreg, m2reg, wmem, aluimm, clk,
    input [3:0] aluc,
    input [4:0] destReg,
    input [31:0] qa, qb, imm32,

    output reg ewreg, em2reg, ewmem, ealuimm,
    output reg [3:0] ealuc,
    output reg [4:0] edestReg,
    output reg [31:0] eqa, eqb, eimm32
);
    always @(posedge clk)
    begin
        ewreg    = wreg;
        em2reg   = m2reg;
        ewmem    = wmem;
        ealuc    = aluc;
        ealuimm  = aluimm;
        edestReg = destReg;
        eqa      = qa;
        eqb      = qb;
        eimm32   = imm32;
    end
endmodule

module ALUMux(
    input [31:0] eqb, eimm32,
    input ealuimm,
    output reg [31:0] b
);
    always @(*)
    begin
        if (ealuimm == 0)
            b <= eqb;
        else
            b <= eimm32;
    end
endmodule

module ALU(
    input [31:0] eqa, b,
    input [3:0] ealuc,

    output reg [31:0] r
);
    always @(*)
    begin
        case(ealuc)
            'b0010:
                r <= eqa + b;
            'b0110:
                r <= eqa - b;
        endcase
    end
endmodule

module EXEMEMPipelineReg(
    input clk, ewreg, em2reg, ewmem,
    input [4:0] edestReg,
    input [31:0] r, eqb,

    output reg mwreg, mm2reg, mwmem,
    output reg [4:0] mdestReg,
    output reg [31:0] mr, mqb
);
    always @(posedge clk)
    begin
        mwreg    <= ewreg;
        mm2reg   <= em2reg;
        mwmem    <= ewmem;
        mdestReg <= edestReg;
        mr       <= r;
        mqb      <= eqb;
    end
endmodule

module DataMemory(
    input clk, mwmem,
    input [31:0] mr, mqb,

    output reg [31:0] mdo
);
    reg [31:0] memory[0:31];
    initial begin
        memory[0] = 'hA00000AA;
        memory[1] = 'h10000011;
        memory[2] = 'h20000022;
        memory[3] = 'h30000033;
        memory[4] = 'h40000044;
        memory[5] = 'h50000055;
        memory[6] = 'h60000066;
        memory[7] = 'h70000077;
        memory[8] = 'h80000088;
        memory[9] = 'h90000099;
     end

    //reading from memory
    always @(*)
        mdo <= memory[mr[7:2]];

    // writing to memory
    always @(negedge clk)
    begin
        if (mwmem == 1)
        begin
            memory[mr[7:2]] <= mqb;
        end
    end
endmodule

module MEMWBPipelineReg (
    input clk, mwreg, mm2reg,
    input [4:0] mdestReg,
    input [31:0] mr, mdo,

    output reg wwreg, wm2reg,
    output reg [4:0] wdestReg,
    output reg [31:0] wr, wdo
);
    always @(posedge clk)
    begin
        wwreg    <= mwreg;
        wm2reg   <= mm2reg;
        wdestReg <= mdestReg;
        wr        <= mr;
        wdo       <= mdo;
    end
endmodule
```

```
‘timescale 1ns / 1ps
module testbench();
    reg clk_tb;
    initial clk_tb = 1’b0;

    wire [31:0] pc, dinstOut;
    wire ewreg, em2reg, ewmem, ealuimm;
    wire [3:0] ealuc;
    wire [4:0] edestReg;
    wire [31:0] eqa, eqb, eimm32;

    wire mwreg, mm2reg, mwmem;
    wire [4:0] mdestReg;
    wire [31:0] mr, mqb;

    wire wwreg, wm2reg;
    wire [4:0] wdestReg;
    wire [31:0] wr, wdo;

    DataPath dp_tb (
        clk_tb,

        pc, dinstOut,
        ewreg, em2reg, ewmem, ealuimm,
        ealuc,
        edestReg,
        eqa, eqb, eimm32,

        mwreg, mm2reg, mwmem,
        mdestReg,
        mr, mqb,

        wwreg, wm2reg,
        wdestReg,
        wr, wdo
    );
    //DataPath dp_tb(clk_tb, pc, dinstOut, ewreg, em2reg, ewmem, ealuimm, ea

    always
    begin
        #5
        clk_tb = ~clk_tb;
    end
endmodule
```

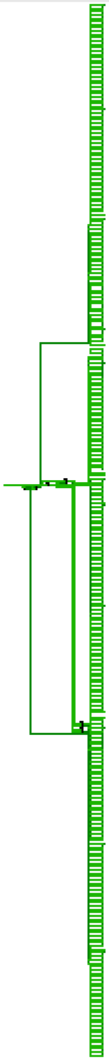
2 Images

Timing Waveform

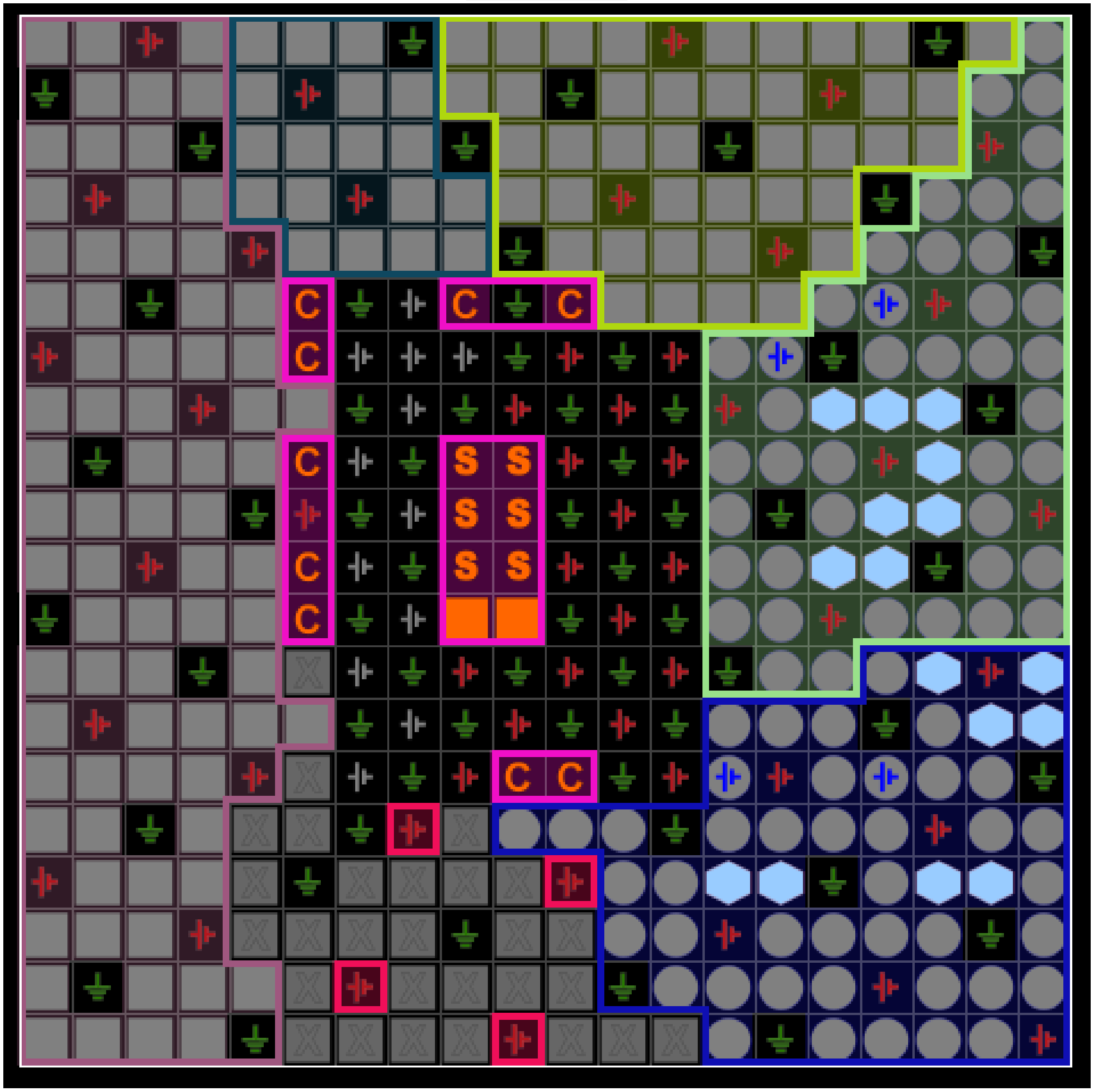


Name	Value
ewreg	1
em2reg	1
ewmem	0
ealuimm	1
ealuc[3:0]	2
edestReg[4:0]	XX
eqa[31:0]	XXXXXXXX
eqb[31:0]	XXXXXXXX
eimm32[31:0]	XXXXXXXX
mwreg	1
mm2reg	1
mwmem	0
mdestReg[4:0]	XX
mr[31:0]	XXXXXXXX
mqb[31:0]	XXXXXXXX
wwreg	1
wm2reg	1
wdestReg[4:0]	XX
wr[31:0]	XXXXXXXX
wdo[31:0]	XXXXXXXX

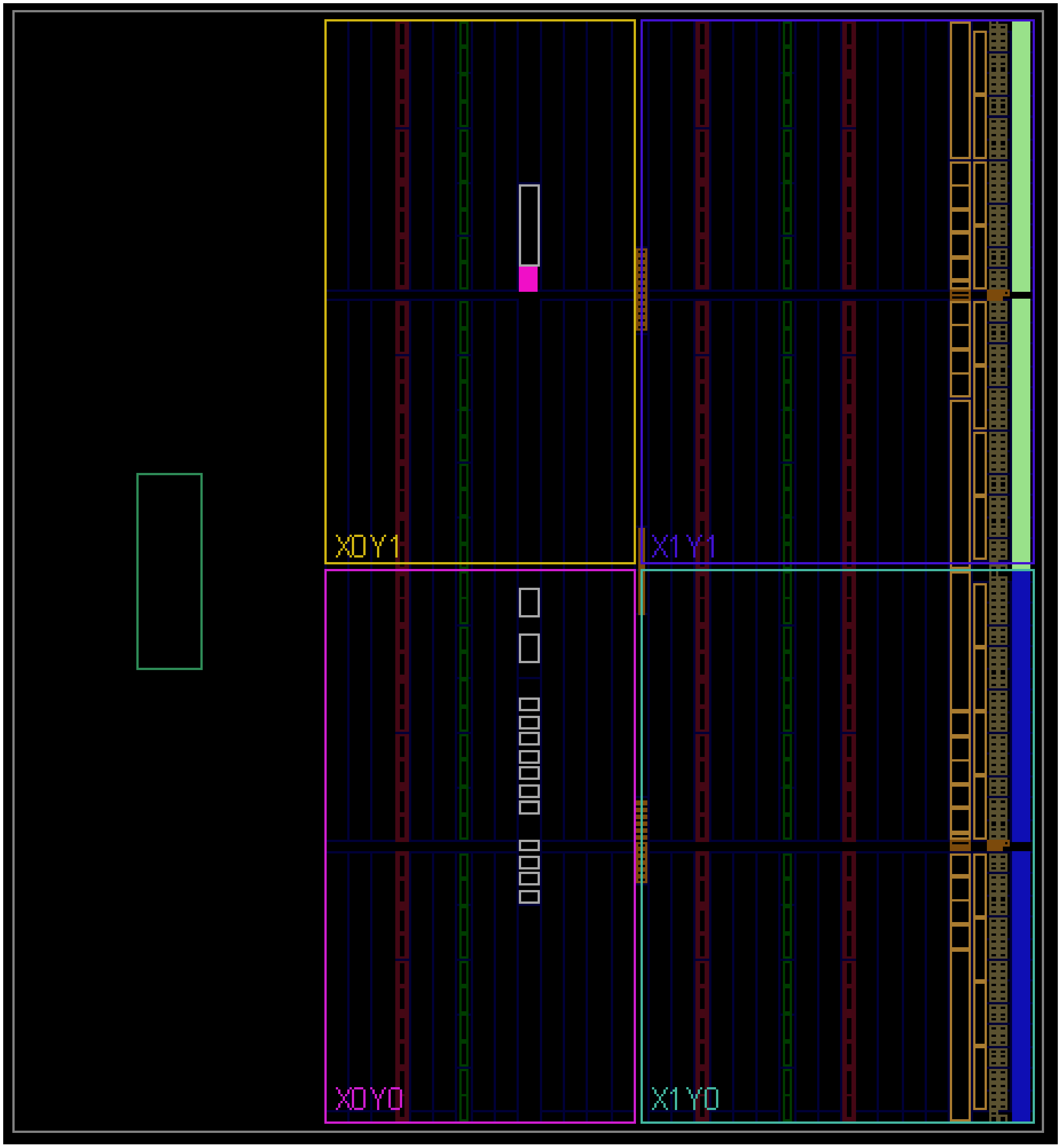
Schematic



I/O Planning



Floor Planning


X0Y1
X1Y1
X0Y0
X1Y0