ANEMPA Auxiliary Code Appendix

Cornell University

January 2008

Idan Beck

CPU Verilog Code

```
// mips like cpu
`include "ndmaref.h"
// Dont Write on this line for some reason

`include "IF.v"
`include "ID.v"
`include "NLayer.v"
`include "RF.v"
`include "ALU.v"
`include "NetCNTL.v"
`include "QC.v"
`include "net_mem_ctrl.v"
`include "in_out.v"

module CPU
(
    CLK, MRST,
    // Instruction Memory
    oIMemAddr, iMemIn,
    oIWAddr, oIWrite, oIData,
    // Data Memory
    dMemIn, dAddr, dWrite, WD,
    // Network
    nBusIn, nBusOut,
    sBusIn, sBusOut,
    eBusIn, eBusOut,
    wBusIn, wBusOut,
    // Peripheral Hook Up
    iPortA, iPortB, iPortC, iPortD,
    oPortE, oPortF, oPortG, oPortH,
    // Debuggins
    oSelfID,
    oDxFlag,
    oRxFlag,
    oInsBufCnt
);

input   wire            CLK;
input   wire            MRST;

// Instruction Memory
input   wire    [31:0]  iMemIn;
output  wire    [31:0]  oIMemAddr;
output  wire    [31:0]  oIWAddr;
output  wire    [31:0]  oIData;
output  wire            oIWrite;

assign oIMemAddr = PC;

// Data Memory
input   wire    [31:0]  dMemIn;
///output  reg     [31:0]  WD;
output  wire    [31:0]  WD;
output  wire    [31:0]  dAddr;
output  wire            dWrite;

// Network Layer
input   wire    [31:0]  nBusIn;
input   wire    [31:0]  wBusIn;
input   wire    [31:0]  sBusIn;
input   wire    [31:0]  eBusIn;
output  wire    [31:0]  nBusOut;
output  wire    [31:0]  sBusOut;
output  wire    [31:0]  eBusOut;
output  wire    [31:0]  wBusOut;

// IO Ports
input   wire    [31:0]  iPortA;
input   wire    [31:0]  iPortB;
input   wire    [31:0]  iPortC;
input   wire    [31:0]  iPortD;
output  wire    [31:0]  oPortE;
output  wire    [31:0]  oPortF;
output  wire    [31:0]  oPortG;
output  wire    [31:0]  oPortH;

// Debuggins
output  wire    [7:0]   oSelfID;
assign oSelfID  = wSelfID;
output  wire            oDxFlag;
assign oDxFlag = dxFlag;
output  wire            oRxFlag;
```

```
assign oRxFlag = rxFlag;
output  wire    [1:0]   oInsBufCnt;

// Program Counter
reg     [31:0]  PC = 0;
wire    [31:0]  pcInc;
wire    [31:0]  nextPC;
wire            takeBr;
wire    [4:0]   pcSel;
wire    [25:0]  pcJump;
wire    [31:0]  pcJmp;
wire    [31:0]  pcBranch;
wire    [15:0]  pcOffset;

// inputs for rf
wire    [4:0]   RS_reg;
wire    [4:0]   RT_reg;
wire    [4:0]   RD_reg;
wire            rWrite;

wire    [31:0]  RS_bus;
wire    [31:0]  RT_bus;

// ID Vars
wire            wBreak;

// IN OUT
wire    [4:0]   wIOAddr;
wire    [31:0]  wIOWD;
wire    [31:0]  wIOData;
wire            wROWrite;

// ALU variables
wire    [31:0]  ALU_bus;
wire    [31:0]  Imm_bus;
wire    [5:0]   Op_bus;
wire            ImmUse;

wire    [31:0]  Ins;

// Network Layer
wire            dxFlag;
wire            dxFlag_r;
//wire            dxFlag_rr;
wire    [7:0]   dest_ID;
wire    [7:0]   send_data;
wire            rxFlag;
wire            wNetMsgComplete;

wire    [31:0]  wNIP;

wire            wInsLoad;
wire    [31:0]  wNetIns;
wire            wNetInsValid;

wire    [7:0]   self_id;
wire            idFlag;
wire    [7:0]   wSelfID;

wire            OutGoingEmpty = (nBusOut == 0) && (sBusOut == 0) &&
                                (eBusOut == 0) && (wBusOut == 0);

assign pcInc = 1;
assign nextPC = PC + pcInc;
assign pcJmp = {PC[31:26], pcJump};
assign pcBranch = (pcOffset[15]) ? PC - (~pcOffset + 1) : PC + pcOffset + 1;

always @(pcOffset) begin
    //$display("pcOffset:%d %b", pcOffset, pcOffset);
end

always @(pcSel) begin
    $display("pcJmp: %d pcSel:%d QC: %d", pcJmp, pcSel, QC_out);
end

// increment the clock accordingly
// PC is flopped on negedge because
// otherwise the QC result is not valid
always @(negedge CLK) begin
    if(MRST)
        begin
            PC <= 0;
```

```
        end
    else
        begin
            if(wBreak == 0 && !wInsLoad && !dxFlag_r &&
               (!wNetInsValid || (wNetInsValid && wNetIns[`op] == `JALNET)))   // This must be removed
                begin
                    case(pcSel)
                        `pc_sel_nex:        PC <= nextPC;
                        `pc_sel_jmp:        PC <= pcJmp;
                        `pc_sel_jmp_reg:    PC <= RS_bus;
                        `pc_sel_br:         PC <= pcBranch;
                        `pc_sel_net_jmp:
                            begin
                                $display("JALNET to PC:%d ID:%d", wNIP, wSelfID);
                                PC <= wNIP;
                            end
                       default:             PC <= nextPC;
                    endcase
                end
        end
end

QC qc (
    .CLK    (CLK),
    .MRST   (MRST),
    .Ins    (Ins),
    .A      (RS_bus),
    .B      (RT_bus),
    .QC_out (QC_out)
);


//always @(QC_out) $display("QC_Out: %d A: %d B: %d", QC_out, RS_bus, RT_bus);

// IF ************************************
// Instruction memory contained in IF
// 1 kb (4*256)
IF ifc (
    .CLK        (CLK),
    .MRST       (MRST),
    .PC         (PC),
    .iMemIn     (iMemIn),
    .Iout       (Ins),
    .takeBranch (takeBr),   // This is deprecated looks like
    .pcSelect   (pcSel),
    .pcJump     (pcJump),
    .pcOffset   (pcOffset),
    .QC         (QC_out),
    // Network
    .iNetIns        (wNetIns),
    .iNetInsValid   (wNetInsValid)
);

// Debugging Section
// ********************************************************************
always @(Ins) if(!MRST) $display("ID: %d PC:%d, Ins:%b", wSelfID, PC+1, Ins);
//always @(Op_bus)  if(!MRST) $display("PC:%d, Op_bus:%b", PC, Op_bus);
//always @(Imm_bus) if(!MRST) $display("PC:%d, Imm_bus:%d", PC, Imm_bus);
//always @(ImmUse) if(!MRST) $display("PC:%d, ImmUse:%d", PC, ImmUse);
//always @(RS_reg) if(!MRST) $display("PC:%d, RS:%d", PC, RS_reg);
//always @(RT_reg) if(!MRST) $display("PC:%d, RT:%d", PC, RT_reg);
//always @(RD_reg) if(!MRST) $display("PC:%d, RD:%d", PC, RD_reg);
//always @(WD) if(!MRST) $display("PC:%d, WD:%d", PC, WD);

//always @(RS_bus) if(!MRST) $display("PC:%d, RS_bus:%d", PC, RS_bus);

// ********************************************************************

// RD ***************************************
ID id (
    .CLK        (CLK),
    .MRST       (MRST),
    .PC         (PC),
    .Iin        (Ins),
    .OpOut      (Op_bus),
    .Imm        (Imm_bus),
    .uImm       (ImmUse),
    .RS         (RS_reg),
    .RT         (RT_reg),
    .RD         (RD_reg),
    .rW         (rWrite),
    .roW        (wROWrite),
```

```
    .dW         (dWrite),
    .oBreak     (wBreak),
    .InsLoad    (wInsLoad),
    // Network
    .iNetIns        (wNetIns),
    .iNetInsValid   (wNetInsValid)
);

always@(wInsLoad)   begin
    $display("wInsLoad:%b PC:%d", wInsLoad, PC);
end

RF rf
(
    .CLK        (CLK),
    .MRST       (MRST),
    .iReadReg1  (RS_reg),
    .iReadReg2  (RT_reg),
    .iWriteReg  (RD_reg),
    .iWriteData (WD),
    .iWrite     (rWrite),
    .oReadData1 (RS_bus),
    .oReadData2 (RT_bus),
    // Network Signals
    // **********************
    // From NLayer
    .iNetMsg(rxData),
    .iNetMsgWrite(rxFlag),
    // To NLayer
    .oNetMsgComplete(wNetMsgComplete),
    // To Memory
    .oNetIns(wNetIns),
    .oInsValid(wNetInsValid),
    // **********************
    .oInsBufCnt(oInsBufCnt)
);

assign wIOAddr = (Ins[`op] == `OUTI) ? RS_reg :
                 (Ins[`op] == `SPECIAL &&
                  Ins[`function] == `IN) ? RS_reg :
                 (Ins[`op] == `SPECIAL &&
                  Ins[`function] == `OUT) ? RD_reg : 5'hzzz;

assign wIOWD = (Ins[`op] == `OUTI) ? Imm_bus : RS_bus;

// IN OUT stuff
in_out io
(
    .iCLK(CLK), .iMRST(MRST),
    // Input
    .iPortA(iPortA), .iPortB(iPortB), .iPortC(iPortC), .iPortD(iPortD),
    // Output
    .oPortE(oPortE), .oPortF(oPortF), .oPortG(oPortG), .oPortH(oPortH),
    // CPU Side
    .iRegAddr(wIOAddr),
    .iRegWD(wIOWD),
    .iRegWrite(wROWrite),
    .oRegData(wIOData)
);

/*always@(Ins or RS_reg or RS_bus or wIOData) begin
    if(Ins[`op] == `OUTI)
        begin
            $display("OUTI RS_reg:%d RS_bus:%d RT_reg: %d RT_bus:%d", RS_reg, RS_bus, RT_reg, RT_bus);
            $display("OUTI wIOAddr:%d oRegData:%d iRegWrite:%b", wIOAddr, wIOData, wROWrite);
        end
end
*/

/*always@(rxData or rxFlag) begin
    $display("rxData:%b, rxFlag:%b", rxData, rxFlag);
end*/

// EX section
ALU alu (
    .CLK            (CLK),
    .RS_bus         (RS_bus),
    .RT_bus         (RT_bus),
    .Immediate      (Imm_bus),
    .useImmediate   (ImmUse),
    .Operation      (Op_bus),
    .ALUresult      (ALU_bus)
```

```
);

// MEM Debugging Section
/*
always @(Ins) begin
    if(Ins[`op] == `LW)
        begin
            $display("** LW: PC:%d, ALU_bus:%d WD: %d", PC, ALU_bus, WD);
        end
    if(Ins[`op] == `SW)
        begin
            $display("** SW: PC:%d, ALU_bus:%d WD: %d", PC, ALU_bus, WD);
        end
end
*/

// WB Section  (no data memory currently)
// WD needs to be RT_bus for sw, dMemIn for lw, and ALU_bus for others

// Loads
assign WD = (Ins[`op] == `LW)   ?   dMemIn  :
            (Ins[`op] == `LH)   ?   {{16{dMemIn[15]}}, dMemIn[15:0]} :
            (Ins[`op] == `LHU)  ?   {{16{1'b0}}, dMemIn[15:0]} :
            (Ins[`op] == `LB)   ?   {{24{dMemIn[7]}}, dMemIn[7:0]} :
            (Ins[`op] == `LBU)  ?   {{24{1'b0}}, dMemIn[7:0]} :
            // Stores
            (Ins[`op] == `SW)   ?   RT_bus  :
            (Ins[`op] == `SH)   ?   {{16{1'b0}}, RT_bus[15:0]} :
            (Ins[`op] == `SB)   ?   {{24{1'b0}}, RT_bus[7:0]} :
            // JALNET
            (wNetIns[`op] == `JALNET && wNetInsValid) ? PC :
            // JAL
            (Ins[`op] == `JAL)  ?   PC + 1  :
            // IO
            (Ins[`op] == `SPECIAL &&
            Ins[`function] == `IN)   ?   wIOData : ALU_bus;

/*always@(Ins[`op] or dMemIn or RT_bus or PC or wIOData or ALU_bus) begin
    case(Ins[`op])
        `LW:    WD <= dMemIn;
        `SW:    WD <= RT_bus;
        `JAL:   WD <= PC + 1;
        default: WD <= ALU_bus;
    endcase
end    */

assign dAddr = ALU_bus;

//always @(ALU_bus)  $display("PC:%d, ALU_bus:%d", PC, ALU_bus);

// The net_mem_ctrl is a glorified counter
// simply controls writing of memory instructions to
// the "dual port" instruction memory

net_mem_ctrl nmc
(
    .iCLK(CLK),
    .iMRST(MRST),
    .iNetIns(wNetIns),
    .iNetInsValid(wNetInsValid),
    .oWAddr(oIWAddr),
    .oWData(oIData),
    .oWrite(oIWrite),
    .oNIP(wNIP)
);

NetCNTL net_cnt(
    .CLK        (CLK),
    .Iin        (Ins),
    .Imm        (Imm_bus),
    .RT_in      (RT_bus),
    .RS_in      (RS_bus),
    .dxFlag     (dxFlag),
    .dxFlag_r   (dxFlag_r),
    .dest_id    (dest_ID),
    .send_data  (send_data),
    .self_id    (self_id),
    .idFlag     (idFlag)
);

//always @(dxFlag) $display("dxFlag: %d d_id:%d s_d:%d", dxFlag, dest_ID, send_data);
```

```
wire    [7:0]   rxData;

NLayer net_layer (
    .CLK                (CLK),
    .MRST               (MRST),
    // Receieve
    .rxFlag             (rxFlag),
    .rxData             (rxData),
    .iNetMsgComplete    (wNetMsgComplete),
    // Send
    .dxFlag             (dxFlag),
    .dxFlag_r           (dxFlag_r),
    //.dxFlag_rr          (dxFlag_rr),
    .dest_id            (dest_ID),
    .send_data          (send_data),
    // Bus Connect
    .nBusIn             (nBusIn),
    .sBusIn             (sBusIn),
    .eBusIn             (eBusIn),
    .wBusIn             (wBusIn),
    .nBusOut            (nBusOut),
    .sBusOut            (sBusOut),
    .eBusOut            (eBusOut),
    .wBusOut            (wBusOut),
    .idFlag             (idFlag),
    .id                 (self_id),
    .oSelfID            (wSelfID)
);

//always@(wBusIn) $display("!! ID: %d, wBusIn: %b", wSelfID, wBusIn);

//always @(idFlag) $display("idFlag: %d PC: %d", idFlag, PC);

// network debuggin
always @(dxFlag) begin
    if(!MRST && dxFlag == 1)
        begin
            $display("S_Msg id:%d d:%d", dest_ID, send_data);
            //$display("S_Msg_S id:%d d:%d", sBusOut[`dest_id], sBusOut[`msg_data]);
            //$display("S_MSG_S: %b", sBusOut);
        end
end

endmodule
```

```
// Instruction Fetch Module

module IF
(
    CLK, MRST,
    PC,
    iMemIn,
    QC,
    Iout,
    takeBranch,
    pcSelect,
    pcJump,
    pcOffset,
    // Network
    iNetIns,
    iNetInsValid
);

input                   CLK;
input                   MRST;
input   wire    [31:0]  PC;
input   wire    [31:0]  iMemIn;
input                   QC;

output  wire    [31:0]  Iout;
output  reg             takeBranch;
output  reg     [4:0]   pcSelect;
output  reg     [25:0]  pcJump;
output  wire    [15:0]  pcOffset;

// Network
input   wire    [31:0]  iNetIns;
input   wire            iNetInsValid;

integer i;

assign Iout = iMemIn;

assign pcOffset = Iout[`immediate];

// Loads memory from file
initial begin
    takeBranch = 0;
    pcSelect = 0;
    pcJump = 0;
end


// take branch
always @(Iout or iNetInsValid) begin
    if(iNetInsValid)
        begin
            if(iNetIns[`op] == `SPECIAL)
                begin
                    case(iNetIns[`function])
                        `NDJR:   takeBranch <= 1;
                        default: takeBranch <= 0;
                    endcase
                end
            else
                begin
                    case(iNetIns[`op])
                        `JALNET:
                            begin
                                takeBranch <= 1;
                            end

                        default: takeBranch <= 0;
                    endcase
                end
        end
    else if(Iout[`op] == `SPECIAL)
        begin
            case(Iout[`function])
                `JR:
                    begin
                        takeBranch <= 1;
                    end
                default: takeBranch <= 0;
            endcase
        end
    else
```

```
        begin
            case(Iout[`op])
                `J:         takeBranch <= 1;
                `JAL:       takeBranch <= 1;
                `BEQ:       begin
                                //takeBranch <= 1;
                                //$display("branch_E! QC: %d", QC);
                            end
                `BNE:       begin
                                takeBranch <= 1;
                                //$display("branch_N! QC: %d", QC);
                            end
                default:    takeBranch <= 0;
            endcase
        end
end

// pcSelect
always @(QC or Iout or iNetInsValid) begin
    if(iNetInsValid)
        begin
            if(iNetIns[`op] == `SPECIAL)
                begin
                    case(iNetIns[`function])
                        `NDJR:      pcSelect <= `pc_sel_jmp_reg;

                        default:    pcSelect <= `pc_sel_nex;
                    endcase
                end
            else
                begin
                    case(iNetIns[`op])
                        `JALNET:    pcSelect <= `pc_sel_net_jmp;

                        default:    pcSelect <= `pc_sel_nex;
                    endcase
                end
        end
    else if(Iout[`op] == `SPECIAL)
        begin
            case(Iout[`function])
                `JR:
                    begin
                        pcSelect <= `pc_sel_jmp_reg;
                        $display("JR!");
                    end

                default:
                    begin
                        pcSelect <= `pc_sel_nex;
                    end
            endcase
        end
    else
        begin
            casex({Iout[`op], QC})
                {`J, `dc}:
                    begin
                        pcSelect <= `pc_sel_jmp;
                    end
                {`JAL, `dc}:
                    begin
                        pcSelect <= `pc_sel_jmp;
                    end

                {`BEQ, `true}:      pcSelect <= `pc_sel_br;
                {`BNE, `false}:     pcSelect <= `pc_sel_br;
                {`BGTZ, `true}:
                    begin
                        pcSelect <= `pc_sel_br;
                    end

                {`BGEZ, `true}:
                    begin
                        pcSelect <= `pc_sel_br;
                    end

                {`BLTZ, `true}:
                    begin
                        pcSelect <= `pc_sel_br;
                    end
```

```
                {`BLEZ, `true}:
                    begin
                        pcSelect <= `pc_sel_br;
                    end

                default:
                    begin
                        pcSelect <= `pc_sel_nex;
                    end
            endcase
        end
end

// pcJump
always @(Iout or iNetInsValid) begin
    if(iNetInsValid)
        begin
            case(iNetIns[`op])
                `JALNET:
                    begin
                        // need this?
                    end

                `NDJR:
                    begin
                       // need this?
                    end
            endcase
        end
    else
        begin
            case(Iout[`op])
                `J:         pcJump <= Iout[`target];
                `JAL:       pcJump <= Iout[`target];
            endcase
        end
end

endmodule
```

```
// Instruction Decoder
`include "ndmaref.h"

module ID
(
    CLK, MRST, PC,
    Iin, OpOut, Imm, uImm,
    RS, RT, RD, rW, dW, roW,
    InsLoad,
    oBreak,
    // Network
    iNetIns,
    iNetInsValid

);
input   wire            CLK;
input   wire            MRST;
input   wire    [31:0]  PC;
input   wire    [31:0]  Iin;

output  reg     [5:0]   OpOut;
output  reg     [31:0]  Imm;
output  reg             uImm;           //use Immediate

output  reg     [4:0]   RS;
output  reg     [4:0]   RT;
output  reg     [4:0]   RD;

output  reg             InsLoad;
output  reg             rW;
output  reg             roW;
output  reg             dW;
output  reg             oBreak;// = 0;

// Network
input   wire    [31:0]  iNetIns;
input   wire            iNetInsValid;

always @(posedge CLK) begin
    if(MRST) begin
        //OpOut <= 0;
        //Imm <= 0;
        //uImm <= 0;
        //RS <= 0;
        //RT <= 0;
        //RD <= 0;
        //rW <= 0;
        //dW <= 0;
        //oBreak <= 0;
    end
end


always @(Iin or iNetInsValid) begin : REGISTERS
     if(iNetInsValid)
        begin
            RS <= iNetIns[`rs];
            RT <= iNetIns[`rt];
            casex ({iNetIns[`op], iNetIns[`function]})
                {`JALNET, `dc6}:
                    begin
                        RD <= `ra;
                    end
                /*{`SPECIAL, `NDJR}:
                    begin
                        // need?
                    end*/
            endcase
        end
    else
        begin
            // RS and RT always from same place (aihgs)
            RS <= Iin[`rs];
            RT <= Iin[`rt];

            // RD reg
            casex ({Iin[`op], Iin[`function]})
                {`SPECIAL, `ADD}:   RD <= Iin[`rd];
                {`SPECIAL, `ADDU}:  RD <= Iin[`rd];
                {`SPECIAL, `SUB}:   RD <= Iin[`rd];
                {`SPECIAL, `SUBU}:  RD <= Iin[`rd];
                {`SPECIAL, `AND}:   RD <= Iin[`rd];
```

```
                {`SPECIAL, `OR}:    RD <= Iin[`rd];
                {`SPECIAL, `XOR}:   RD <= Iin[`rd];
                {`SPECIAL, `NOR}:   RD <= Iin[`rd];
                {`SPECIAL, `SLT}:   RD <= Iin[`rd];
                {`SPECIAL, `SLTU}:  RD <= Iin[`rd];
                {`SPECIAL, `SLL}:   RD <= Iin[`rd];
                {`SPECIAL, `SRL}:   RD <= Iin[`rd];
                {`SPECIAL, `SRA}:   RD <= Iin[`rd];
                // Multiply Divide Rem(mod)
                {`SPECIAL, `MULT}:  RD <= Iin[`rd];
                {`SPECIAL, `MULTU}: RD <= Iin[`rd];
                {`SPECIAL, `DIV}:   RD <= Iin[`rd];
                {`SPECIAL, `DIVU}:  RD <= Iin[`rd];
                {`SPECIAL, `MOD}:   RD <= Iin[`rd];
                // IN OUT
                {`SPECIAL, `IN}:    RD <= Iin[`rd];
                {`SPECIAL, `OUT}:   RD <= Iin[`rd];
                // Immediate Instructions
                {`ADDI, `dc6}:      RD <= Iin[`rt];
                {`ADDIU, `dc6}:     RD <= Iin[`rt];
                {`SLTI, `dc6}:      RD <= Iin[`rt];
                {`SLTIU, `dc6}:     RD <= Iin[`rt];
                {`ANDI,  `dc6}:     RD <= Iin[`rt];
                {`ORI,   `dc6}:     RD <= Iin[`rt];
                {`XORI,  `dc6}:     RD <= Iin[`rt];
                {`LUI,   `dc6}:     RD <= Iin[`rt];
                // JAL
                {`JAL, `dc6}:       RD <= `ra;  // return address
                // Memory Instructions
                {`LW,  `dc6}:       RD <= Iin[`rt];
                {`LH,  `dc6}:       RD <= Iin[`rt];
                {`LHU, `dc6}:       RD <= Iin[`rt];
                {`LB,  `dc6}:       RD <= Iin[`rt];
                {`LBU, `dc6}:       RD <= Iin[`rt];
                // Network Instructions (CPU Side)
                {`SMSG,  `dc6}:     RD <= Iin[`rt];
                {`SID,   `dc6}:     RD <= Iin[`rt];
                //{`BCST,  `dc6}:
            endcase
        end
end

// Load Delay
always @(posedge CLK)
begin
    case(Iin[`op])
        `LW:
            begin
                if(InsLoad != 1) InsLoad <= 1;
                else InsLoad <= 0;
            end
        `LH:
            begin
                if(InsLoad != 1) InsLoad <= 1;
                else InsLoad <= 0;
            end
        `LHU:
            begin
                if(InsLoad != 1) InsLoad <= 1;
                else InsLoad <= 0;
            end
        `LB:
            begin
                if(InsLoad != 1) InsLoad <= 1;
                else InsLoad <= 0;
            end
        `LBU:
            begin
                if(InsLoad != 1) InsLoad <= 1;
                else InsLoad <= 0;
            end

        default:    InsLoad <= 0;
    endcase
end

always @(Iin)
begin : OP_CODE
    // OP Code
    casex ({Iin[`op], Iin[`function]})
        {`SPECIAL, `ADD}:   OpOut <= `alu_add;
        {`SPECIAL, `ADDU}:  OpOut <= `alu_add;
```

```
        {`SPECIAL, `SUB}:   OpOut <= `alu_sub;
        {`SPECIAL, `SUBU}:  OpOut <= `alu_sub;
        {`SPECIAL, `AND}:   OpOut <= `alu_and;
        {`SPECIAL, `OR}:    OpOut <= `alu_or;
        {`SPECIAL, `XOR}:   OpOut <= `alu_xor;
        {`SPECIAL, `NOR}:   OpOut <= `alu_nor;
        {`SPECIAL, `SLT}:   OpOut <= `alu_slt;
        {`SPECIAL, `SLTU}:  OpOut <= `alu_sltu;
        {`SPECIAL, `SLL}:   OpOut <= `alu_sll;
        {`SPECIAL, `SRL}:   OpOut <= `alu_srl;
        {`SPECIAL, `SRA}:   OpOut <= `alu_sra;
        // Mult Div Mod
        {`SPECIAL, `MULT}:  OpOut <= `alu_mult;
        {`SPECIAL, `MULTU}: OpOut <= `alu_multu;
        {`SPECIAL, `DIV}:   OpOut <= `alu_div;
        {`SPECIAL, `DIVU}:  OpOut <= `alu_divu;
        {`SPECIAL, `MOD}:   OpOut <= `alu_mod;
        // IN OUT
        {`SPECIAL, `IN}:    OpOut <= `alu_add;
        {`SPECIAL, `OUT}:   OpOut <= `alu_add;
        // Memory Ops
        // Loads
        {`LW, `dc6}:        OpOut <= `alu_add;
        {`LH, `dc6}:        OpOut <= `alu_add;
        {`LHU,`dc6}:        OpOut <= `alu_add;
        {`LB, `dc6}:        OpOut <= `alu_add;
        {`LBU,`dc6}:        OpOut <= `alu_add;
        // Stores
        {`SW, `dc6}:        OpOut <= `alu_add;
        {`SH, `dc6}:        OpOut <= `alu_add;
        {`SB, `dc6}:        OpOut <= `alu_add;
        // Immediate Instructions
        {`ADDI, `dc6}:      OpOut <= `alu_add;
        {`ADDIU, `dc6}:     OpOut <= `alu_add;
        {`SLTI, `dc6}:      OpOut <= `alu_slt;
        {`SLTIU, `dc6}:     OpOut <= `alu_sltu;
        {`ANDI,  `dc6}:     OpOut <= `alu_and;
        {`ORI,   `dc6}:     OpOut <= `alu_or;
        {`XORI,  `dc6}:     OpOut <= `alu_xor;
        {`LUI,   `dc6}:     OpOut <= `alu_or;
        // JAL
        {`JAL,   `dc6}:     OpOut <= `alu_add;
        // network instructions
        {`SMSG,  `dc6}:     OpOut <= `alu_add;
        {`SID,   `dc6}:     OpOut <= `alu_add;
        {`BCST,  `dc6}:     OpOut <= `alu_add;
        {`SMSGR, `dc6}:     OpOut <= `alu_add;
        {`BCSTR, `dc6}:     OpOut <= `alu_add;

    endcase

    // Use Immediate
    uImm <= (   (Iin[`op] == `ADDI)  || (Iin[`op] == `ADDIU)  ||
                (Iin[`op] == `SLTI)  || (Iin[`op] == `SLTIU)  ||
                (Iin[`op] == `ANDI)  || (Iin[`op] == `ORI)    ||
                (Iin[`op] == `XORI)  || (Iin[`op] == `LUI)    ||
                // Network Instructions
                (Iin[`op] == `SMSG)  || (Iin[`op] == `SID)    ||
                (Iin[`op] == `BCST)  ||
                // Mem Ops
                // Loads
                (Iin[`op] == `LW)    ||
                (Iin[`op] == `LH)    || (Iin[`op] == `LHU)    ||
                (Iin[`op] == `LB)    || (Iin[`op] == `LBU)    ||
                // Stores
                (Iin[`op] == `SW)    || (Iin[`op] == `SH)     ||
                (Iin[`op] == `SB)    ||
                // Special Ops
                ((Iin[`op] == `SPECIAL) && ((Iin[`function] == `SLL) ||
                                            (Iin[`function] == `SRL) ||
                                            (Iin[`function] == `SRA) )) );


    // Choose The Proper Immediate
    casex(Iin[`op])
        `ADDI:  Imm <= {{16{Iin[15]}}, {Iin[`immediate]}};
        `ADDIU: Imm <= {{16{1'b0}}, {Iin[`immediate]}};
        `SLTI:  Imm <= {{16{Iin[15]}}, {Iin[`immediate]}};
        `SLTIU: Imm <= {{16{Iin[15]}}, {Iin[`immediate]}};

        `ANDI:  Imm <= {16'b0, {Iin[`immediate]}};
        `ORI:   Imm <= {16'b0, {Iin[`immediate]}};
```

```
        `XORI:  Imm <= {16'b0, {Iin[`immediate]}};

        // Shift Ops
        `SLL:   Imm <= {27'b0, {Iin[`sa]}};
        `SRL:   Imm <= {27'b0, {Iin[`sa]}};
        `SRA:   Imm <= {27'b0, {Iin[`sa]}};

        // IN OUT
        `OUTI:  Imm <= {{16{Iin[15]}}, {Iin[`immediate]}};

        // Memory Ops
        // Loads
        `LW:    Imm <= {16'b0, Iin[`immediate]};
        `LH:    Imm <= {16'b0, Iin[`immediate]};
        `LHU:   Imm <= {16'b0, Iin[`immediate]};
        `LB:    Imm <= {16'b0, Iin[`immediate]};
        `LBU:   Imm <= {16'b0, Iin[`immediate]};
        // Stores
        `SW:    Imm <= {16'h0, Iin[`immediate]};
        `SH:    Imm <= {16'h0, Iin[`immediate]};
        `SB:    Imm <= {16'h0, Iin[`immediate]};

        `LUI:   Imm <= {{Iin[`immediate]}, 16'b0};

        // Since the message data can only be 8 bits currently we
        // only use the lower 16 bits
        `SMSG:  Imm <= {16'b0, {Iin[`immediate]}};
        `SID:   Imm <= {16'b0, {Iin[`immediate]}};
        `BCST:  Imm <= {16'b0, {Iin[`immediate]}};
        `SMSGR: Imm <= {16'b0, {Iin[`immediate]}};
        `BCSTR: Imm <= {16'b0, {Iin[`immediate]}};

    endcase
end

// Data Write
always @(Iin)
begin : DATA_WRITE
    if(Iin == 0)
        begin
            dW <= 0;
        end
    else if(Iin[`op] == `SPECIAL)
        begin
            dW <= 0;
        end
    else
        begin
            case(Iin[`op])
                `SW:    dW <= 1;
                `SH:    dW <= 1;
                `SB:    dW <= 1;

                default: dW <= 0;
            endcase
        end
end // end DATA_WRITE

// Write to Register at negative edge of clock (registerWrite)
always @(Iin or iNetInsValid)
begin : REG_WRITE
    if(iNetInsValid)
        begin
            case(iNetIns[`op])
                `JALNET:    rW <= 1;
                default:    rW <= 0;
            endcase
        end
    else
        begin
            // reg Write
            if(Iin == 0)
                begin
                    //if(InsLoad) rW <= 1;
                    //else rW <= 0;
                    rW <= 0;
                end
            else if(Iin[`op] == `SPECIAL)
                begin
                    case (Iin[`function])
                        `ADD:   rW <= 1;
                        `ADDU:  rW <= 1;
```

```
                        `SUB:   rW <= 1;
                        `SUBU:  rW <= 1;
                        `AND:   rW <= 1;
                        `OR:    rW <= 1;
                        `XOR:   rW <= 1;
                        `NOR:   rW <= 1;
                        `SLT:   rW <= 1;
                        `SLTU:  rW <= 1;
                        `SLL:   rW <= 1;
                        `SRL:   rW <= 1;
                        `SRA:   rW <= 1;
                        // mult div mod
                        `MULT:  rW <= 1;
                        `MULTU: rW <= 1;
                        `DIV:   rW <= 1;
                        `DIVU:  rW <= 1;
                        `MOD:   rW <= 1;
                        // IN OUT
                        `IN:    rW <= 1;
                        `OUT:   rW <= 0;
                        default:            rW <= 0;
                    endcase
                end
            else
                begin
                    case (Iin[`op])
                        // immediate instructions
                        `ADDI:      rW <= 1;
                        `ADDIU:     rW <= 1;
                        `SLTI:      rW <= 1;
                        `SLTIU:     rW <= 1;
                        `ANDI:      rW <= 1;
                        `ORI:       rW <= 1;
                        `XORI:      rW <= 1;
                        `LUI:       rW <= 1;
                        // JAL
                        `JAL:       rW <= 1;
                        // Memory Ops
                        // Loads
                        `LW:        rW <= 1;
                        `LH:        rW <= 1;
                        `LHU:       rW <= 1;
                        `LB:        rW <= 1;
                        `LBU:       rW <= 1;
                        // Stores
                        `SW:        rW <= 0;
                        `SH:        rW <= 0;
                        `SB:        rW <= 0;
                        // IN OUT
                        `OUTI:      rW <= 0;
                        // network instructions
                        `SMSG:      rW <= 0;
                        `SID:       rW <= 0;
                        `BCST:      rW <= 0;
                        `SMSGR:     rW <= 0;
                        `BCSTR:     rW <= 0;
                        default:    rW <= 0;
                    endcase
                end
        end
end

// IN OUT Register Write
always@(Iin)
begin : INOUT_WRITE
    if(Iin == 0)
        begin
            roW <= 0;
        end
    else if(Iin[`op] == `SPECIAL)
        begin
            case (Iin[`function])
                `OUT:       roW <= 1;
                default:    roW <= 0;
            endcase
        end
    else
        begin
            case (Iin[`op])
                `OUTI:      roW <= 1;
                default:    roW <= 0;
            endcase
```

```
        end
end

//always @(rW) $display("rW change!");


// break instruction
always @(Iin)
begin : BREAK_KILL
    if(MRST)
        begin
            oBreak <= 0;
        end
    else
        begin
            casex({Iin[`op], Iin[`function]})
                {`SPECIAL, `BREAK}:
                    begin
                        $display("Break Encountered");
                        // Need to halt PC some how
                        oBreak <= 1;
                        $stop;
                    end
                default: oBreak <= 0;
            endcase
        end
end


endmodule
```

```
// this module simply comapres two 32 bit
// inputs for equality then sets

// Add functionality for less than
//                       less than equal
//                       greater than
//                       greater than equal
module QC
(
    CLK, MRST,
    Ins, A, B,
    QC_out
);

input   wire            CLK;
input   wire            MRST;
input   wire    [31:0]  Ins;
input   wire    [31:0]  A;
input   wire    [31:0]  B;

output  wire            QC_out;

assign QC_out = (Ins[`op] == `BGTZ) ? (A[31] != 1) :
                (Ins[`op] == `BGEZ) ? (A[31] != 1 || A == 0) :
                (Ins[`op] == `BLTZ) ? (A[31] == 1) :
                (Ins[`op] == `BLEZ) ? (A[31] == 1 || A == 0) : (A == B);

always@(A or B or Ins) begin
    //if(Ins[`op] == `BGTZ) $display("BGTZ A:%b B:%b QC_out:%b", A, B, QC_out);
end

endmodule
```

```
// Register File
// 32 General Purpose Registers

module RF
(
    CLK, MRST,
    // Register File
    iReadReg1,
    iReadReg2,
    iWriteReg,
    iWriteData,
    iWrite,
    oReadData1,
    oReadData2,

    // Network RF
    // **************
    // From NLayer
    iNetMsg,
    iNetMsgWrite,
    // To Nlayer
    oNetMsgComplete,
    // To Memory
    oNetIns,
    oInsValid,
    // **************

    // debugging
    oInsBufCnt
);

input   wire            CLK;
input   wire            MRST;

// Register File
input   wire    [4:0]   iReadReg1;
input   wire    [4:0]   iReadReg2;
input   wire    [4:0]   iWriteReg;
input   wire    [31:0]  iWriteData;
input   wire            iWrite;
output  wire    [31:0]  oReadData1;
output  wire    [31:0]  oReadData2;

// Network RF
// ************************************
// From NLayer
input   wire    [7:0]   iNetMsg;
input   wire            iNetMsgWrite;
// To NLayer
output  reg             oNetMsgComplete;

// To Memory
output  wire    [31:0]  oNetIns;
output  reg             oInsValid;
// ************************************

output  wire    [1:0]   oInsBufCnt;

assign oInsBufCnt = rInsBufCnt;


integer i;

// 32 multipurpose registers
reg     [31:0]  registerFile    [0:31];

// Network Purpose Registers
reg     [7:0]   rInsBuf     [0:3];
reg     [1:0]   rInsBufCnt;

assign oNetIns = {rInsBuf[3], rInsBuf[2], rInsBuf[1], rInsBuf[0]};

assign oReadData1 = registerFile[iReadReg1];
assign oReadData2 = registerFile[iReadReg2];

// We write to the register file on the negative edge of the clock
// This means all of the stages must complete in that time.
always @(negedge CLK) begin : WRITE_DATA
    if(MRST)
        begin
            for(i = 0; i < 32; i = i + 1)
                begin
```

```
                    registerFile[i] <= 0;
                end
        end
    else
        begin
            // Register File
            if(iWrite)
                begin
                    if(iWriteReg != 0)
                        begin
                            registerFile[iWriteReg] <= iWriteData;
                            $display("RF: Register $%d value: %h", iWriteReg, registerFile[iWriteReg]);
                        end
                    else
                        begin
                            $display("Invalid Attempted Write on $0");
                        end
                end
        end
end


// Network Instruction Buffer

//always@(iNetMsgWrite or oInsValid) begin
always@(negedge CLK) begin
    if(MRST)
        begin
            rInsBufCnt <= 3;
            oInsValid <= 0;
            oNetMsgComplete <= 0;

            for(i = 0; i < 4; i = i + 1)
                begin
                    rInsBuf[i] <= 0;
                end

        end
    else
        begin
            if(iNetMsgWrite && !oNetMsgComplete)
                begin
                    rInsBuf[rInsBufCnt] <= iNetMsg;
                    $display("!!! NETRF wrote:%b to location:%d", iNetMsg, rInsBufCnt);

                    oNetMsgComplete <= 1;

                    if(rInsBufCnt != 0)
                        begin
                            rInsBufCnt <= rInsBufCnt - 1;
                        end
                    else
                        begin
                            rInsBufCnt <= 3;
                            oInsValid <= 1;
                        end
                end
            else if(!iNetMsgWrite)
                begin
                    // Reset Complete
                    oNetMsgComplete <= 0;
                    if(oInsValid)
                        begin
                            $display("Ins Valid:%b", oNetIns);
                            for(i = 0; i < 4; i = i + 1) rInsBuf[i] <= 8'hzz;
                        end
                    oInsValid <= 0;
                end
        end
end

endmodule
```

```
// Input output ports stay within CPU conventions
// of 32 bit width

module in_out
(
    iCLK, iMRST,
    // Input
    iPortA, iPortB, iPortC, iPortD,
    // Output
    oPortE, oPortF, oPortG, oPortH,
    // CPU Side
    iRegAddr,
    iRegWD,
    iRegWrite,
    oRegData,
);

input   wire            iCLK;
input   wire            iMRST;

// Input
input   wire    [31:0]  iPortA; // Addressed as $1
input   wire    [31:0]  iPortB; // Addressed as $2
input   wire    [31:0]  iPortC; // Addressed as $3
input   wire    [31:0]  iPortD; // Addressed as $4

// Output
output  reg     [31:0]  oPortE; // Addressed as $5
output  reg     [31:0]  oPortF; // Addressed as $6
output  reg     [31:0]  oPortG; // Addressed as $7
output  reg     [31:0]  oPortH; // Addressed as $8

input   wire    [4:0]   iRegAddr;
input   wire    [31:0]  iRegWD;
input   wire            iRegWrite;
output  wire    [31:0]  oRegData;

// Output Data Mux
assign oRegData = (iRegWrite) ? 32'hzzzzzzzz :
                  (iRegAddr == `r1) ? iPortA :
                  (iRegAddr == `r2) ? iPortB :
                  (iRegAddr == `r3) ? iPortC :
                  (iRegAddr == `r4) ? iPortD :
                  (iRegAddr == `r5) ? oPortE :
                  (iRegAddr == `r6) ? oPortF :
                  (iRegAddr == `r7) ? oPortG :
                  (iRegAddr == `r8) ? oPortH : 32'hxxxxxxxx;

// Use Negative Edge just like RF
always@(negedge iCLK) begin
    if(iMRST)
        begin
            oPortE <= 0;
            oPortF <= 0;
            oPortG <= 0;
            oPortH <= 0;
        end
    else
        begin
            if(iRegWrite)
                begin
                    if(iRegAddr >= `r5 && iRegAddr <= `r8)
                        begin
                            case(iRegAddr)
                                `r5: oPortE <= iRegWD;
                                `r6: oPortF <= iRegWD;
                                `r7: oPortG <= iRegWD;
                                `r8: oPortH <= iRegWD;
                            endcase
                            $display("** INOUT wrote %h to %d", iRegWD, iRegAddr);
                        end
                    else
                        begin
                            $display("INOUT attempted illegal write to r%d", iRegAddr);
                        end

                end
        end
end


endmodule
```

```
`include "ndmaref.h"

// Takes in two values and operates on them (32 bits)
// opCode is 6 bits
// ALUresult is 32 bits

module ALU
(
    CLK,
    RS_bus,
    RT_bus,
    Immediate,
    useImmediate,
    Operation,
    ALUresult
);

input   wire            CLK;
input   wire    [31:0]  RS_bus;
input   wire    [31:0]  RT_bus;
input   wire    [5:0]   Operation;
input   wire    [31:0]  Immediate;
input   wire            useImmediate;
output  wire    [31:0]  ALUresult;

wire    [31:0]  Toperand;
wire    [4:0]   ShiftAmt;

// Multiply Result
reg     [63:0]  MultResult;

assign Toperand = (useImmediate) ? Immediate : RT_bus;
assign ShiftAmt = (useImmediate) ? Immediate[10:6] : RS_bus[4:0];

// Combinational Based ALU
// Faster sometimes
assign ALUresult = (Operation == `alu_add) ? RS_bus + Toperand :
                   (Operation == `alu_sub) ? RS_bus - Toperand :
                   (Operation == `alu_or)  ? RS_bus | Toperand :
                   (Operation == `alu_and) ? RS_bus & Toperand :
                   (Operation == `alu_xor) ? RS_bus ^ Toperand :
                   (Operation == `alu_nor) ? ~(RS_bus | Toperand) :
                   (Operation == `alu_slt) ? (RS_bus < Toperand) :
                   (Operation == `alu_sltu) ? (RS_bus < Toperand) :  // Need to fix?
                   (Operation == `alu_sll) ? RT_bus << Toperand :
                   (Operation == `alu_srl) ? RT_bus >> Toperand :
                   (Operation == `alu_sra) ? RT_bus >>> Toperand : 32'hzzzzzzzz;
                   /*(Operation == `alu_mult) ? RS_bus * Toperand :
                   (Operation == `alu_multu) ? RS_bus * Toperand :
                   (Operation == `alu_div) ? RS_bus / Toperand :
                   (Operation == `alu_divu) ? RS_bus / Toperand :
                   (Operation == `alu_mod) ? RS_bus % Toperand : 32'hzzzzzzzz; */

endmodule
```

```
module mem
(
    CLK, MRST,
    // Instruction Memory
    iAddr, Iout, iIData, iIWrite, iIWAddr,
    // Data Memory
    dAddr, dWrite, Din, Dout
);

input   wire            CLK;
input   wire            MRST;

// Instruction Memory
input   wire    [31:0]  iAddr;
output  wire    [31:0]  Iout;
input   wire    [31:0]  iIWAddr;
input   wire    [31:0]  iIData;
input   wire            iIWrite;

// Data Memory
input   wire    [31:0]  dAddr;
input   wire            dWrite;
input   wire    [31:0]  Din;
output  wire    [31:0]  Dout;

integer i;

// default memfile
parameter pIMEM_FILE = "mem.mem";

// these will eventually be replaced with real
// FPGA memories using M4K blocks
reg [31:0] imem [255:0];
reg [31:0] dmem [255:0];

initial begin

end

assign Iout = imem[iAddr];
assign Dout = (dWrite) ? 32'hzzzzzzzz : dmem[dAddr];

//always @(dAddr or dWrite or Din) begin
always @(negedge CLK) begin
    if(MRST)
        begin
                for(i = 0; i < 256; i = i + 1)
                begin
                    dmem[i] = 0;
                end
                $readmemb(pIMEM_FILE, imem);
        end
    else
        begin
            if(iIWrite)
                begin
                    imem[iIWAddr] <= iIData;
                    $display("Instruction Mem: Wrote: %d at %d", iIData, iIWAddr);
                end
            // Data Memory
            if(dWrite)
                begin
                    dmem[dAddr] <= Din;
                    $display("Data Mem: Wrote: %d at %d", Din, dAddr);
                end
        end
end

endmodule
```

```
// This is the network hardware controller.
// This is simpler to do this way rather than complicate the operation
// of the ALU and surrounding hardware

// Mainly drives the send and ID commands

`include "ndmaref.h"

module NetCNTL
(
    CLK, Iin, Imm, RT_in, RS_in,
    dxFlag,
    dxFlag_r,
    dest_id, send_data,
    self_id, idFlag
);

input   wire            CLK;
input   wire    [31:0]  Iin;
input   wire    [31:0]  Imm;
input   wire    [31:0]  RT_in;
input   wire    [31:0]  RS_in;

output  reg             dxFlag;
input   wire            dxFlag_r;
output  reg     [7:0]   dest_id;
output  reg     [7:0]   send_data;

output  wire    [7:0]   self_id;
output  reg             idFlag = 0;

assign self_id   = Imm[7:0];

// Split up and MUX RS_in and RT_in
// RS
wire    [7:0]   RS_in_3 = RS_in[31:24];
wire    [7:0]   RS_in_2 = RS_in[23:16];
wire    [7:0]   RS_in_1 = RS_in[15:8];
wire    [7:0]   RS_in_0 = RS_in[7:0];
// RT
wire    [7:0]   RT_in_3 = RT_in[31:24];
wire    [7:0]   RT_in_2 = RT_in[23:16];
wire    [7:0]   RT_in_1 = RT_in[15:8];
wire    [7:0]   RT_in_0 = RT_in[7:0];

// self id
always @(Iin) begin
        case ({Iin[`op]})
            {`SID}:
                begin
                    idFlag <= 1;
                    $display("set id: %d", self_id);
                end

            default:    idFlag <= 0;
        endcase
end

// dxFlag
always @(Iin) begin
    case ({Iin[`op]})
            {`SMSG}:
                begin
                    if(!dxFlag_r)
                        begin
                            dxFlag <= 1;
                        end
                    else
                        begin
                            dxFlag <= 0;
                        end
                    send_data <= Imm[7:0];
                    dest_id <= RT_in[7:0];
                end

            {`BCST}:
                begin
                    if(!dxFlag_r)
                        begin
                            dxFlag <= 1;
                        end
                    else
```

```
                        begin
                            dxFlag <= 0;
                        end
                    send_data <= Imm[7:0];
                    dest_id <= 8'hFF;           // Special ID for BCAST
                end

            {`SMSGR}:
                begin
                    if(!dxFlag_r)
                        begin
                            dxFlag <= 1;
                        end
                    else
                        begin
                            dxFlag <= 0;
                        end
                    //$display("--> SMSGR imm:%d RS_in:%h RT_in:%h", Imm[7:0], RS_in, RT_in);
                    case(Imm[7:0])
                        8'd3:   send_data <= RS_in_3;
                        8'd2:   send_data <= RS_in_2;
                        8'd1:   send_data <= RS_in_1;
                        8'd0:   send_data <= RS_in_0;
                        default:    send_data <= 0;     // this shouldn't happen
                    endcase
                    dest_id <= RT_in[7:0];
                end

            {`BCSTR}:
                begin
                    if(!dxFlag_r)
                        begin
                            dxFlag <= 1;
                        end
                    else
                        begin
                            dxFlag <= 0;
                        end
                    //$display("--> BCSTR imm:%d RS_in:%h RT_in:%h", Imm[7:0], RS_in, RT_in);
                    case(Imm[7:0])
                        8'd3:   send_data <= RS_in_3;
                        8'd2:   send_data <= RS_in_2;
                        8'd1:   send_data <= RS_in_1;
                        8'd0:   send_data <= RS_in_0;
                        default:    send_data <= 0;     // this shouldn't happen
                    endcase
                    dest_id <= 8'hFF;           // Special ID for BCAST
                end

            default:
                begin
                    dxFlag <= 0;
                    // likely optional but nice for debugging
                    send_data <= 0;
                    dest_id <= 0;
                end
        endcase
end

endmodule
```

```
// This is the network routing layer,  it basically acts transparently to
// the processor taking in an input message and dispatching or setting a
// flag to the processor that a message is waiting in the stack

`include "ndmaref.h"

module NLayer
(
    CLK, MRST,
    // ID
    idFlag, id, oSelfID,
    // Receive --> NET RF
    rxFlag, rxData,
    // NET RF Handshake signal
    iNetMsgComplete,
    // Send
    dxFlag, dest_id, send_data, dxFlag_r, dxFlag_rr,
    // Network Connect
    nBusIn, sBusIn, eBusIn, wBusIn,
    nBusOut, sBusOut, eBusOut, wBusOut
);

input   wire            MRST;
input   wire            CLK;
// ID
input   wire            idFlag;
input   wire    [7:0]   id;
output  wire    [7:0]   oSelfID;

// Recieve --> NET RF
output  wire            rxFlag;
output  wire    [7:0]   rxData;
// Complete flag from NET RF
input   wire            iNetMsgComplete;

// if we want to send a message
// assert the dxFlag when all the fields are valid
input   wire            dxFlag;
output  reg             dxFlag_r;
output  reg             dxFlag_rr;

input   wire    [7:0]   dest_id;
input   wire    [7:0]   send_data;

// We have 4 directions (this is technically scalable)
/*****************************************************
* Message is 32 bits:
*  8 bit Destination ID
*  8 bit Message
*  8 bit Origination ID
*  4 bit Age
*  2 bit origin dispatch direction
*  2 bit last taken direction
*****************************************************/
input   wire    [31:0]  nBusIn;
input   wire    [31:0]  sBusIn;
input   wire    [31:0]  eBusIn;
input   wire    [31:0]  wBusIn;

output  wire    [31:0]  nBusOut;
output  wire    [31:0]  sBusOut;
output  wire    [31:0]  eBusOut;
output  wire    [31:0]  wBusOut;

// This obtains the value if the connect is "on"
// if not the output busses are assigned to the wire values.
reg     [31:0]  nBus_R;
reg     [31:0]  sBus_R;
reg     [31:0]  eBus_R;
reg     [31:0]  wBus_R;

// Adaptive Network Counters
reg     [7:0]   n_cnt;
reg     [7:0]   s_cnt;
reg     [7:0]   e_cnt;
reg     [7:0]   w_cnt;

// Adaptive Network Control
wire            n_Thru;
wire            s_Thru;
wire            w_Thru;
wire            e_Thru;
```

```
// Receieve
reg             rxnFlag;
reg             rxsFlag;
reg             rxeFlag;
reg             rxwFlag;

reg     [7:0]   rxnData;
reg     [7:0]   rxsData;
reg     [7:0]   rxeData;
reg     [7:0]   rxwData;

// This is our 8 bit IDs
// Initialize to FF for debugging
reg     [7:0]   self_id = 8'hFF;

assign oSelfID = self_id;

// Adaptive Logic
assign n_Thru = (n_cnt > `OptVal);
assign s_Thru = (s_cnt > `OptVal);
assign w_Thru = (w_cnt > `OptVal);
assign e_Thru = (e_cnt > `OptVal);

// Register Thru Debugging
always @(n_Thru or s_Thru or w_Thru or e_Thru) begin
    if(n_Thru) $display("* North thru on id: %d cut", self_id);
    if(s_Thru) $display("* South thru on id: %d cut", self_id);
    if(e_Thru) $display("* East thru on id: %d cut", self_id);
    if(w_Thru) $display("* West thru on id: %d cut", self_id);
end


// this is our thru mux where the connection
// is enabled based on the x_Thru control signal
assign nBusOut = (s_Thru) ? sBusIn : nBus_R;
assign sBusOut = (n_Thru) ? nBusIn : sBus_R;
assign eBusOut = (w_Thru) ? wBusIn : eBus_R;
assign wBusOut = (e_Thru) ? eBusIn : wBus_R;

assign rxFlag = ( rxnFlag | rxsFlag | rxeFlag | rxwFlag );

assign rxData = (rxnFlag) ? rxnData :
                (rxsFlag) ? rxsData :
                (rxeFlag) ? rxeData :
                (rxwFlag) ? rxwData : 8'hzz;

// Set ID
always @(idFlag) begin
    if(MRST)
        begin
            self_id <= 8'hFF;
        end
    else if(idFlag && self_id == 8'hFF && id != 0 && id != 8'hFF )
        begin
            self_id <= id;
        end
end

always @(posedge CLK) begin
    if(MRST) begin
        dxFlag_r <= 0;
        dxFlag_rr <= 0;
    end

    // Send Flags
    /*if(dxFlag)
        begin
            if(!dxFlag_r)
                begin
                    $display("dxflag did:%d sd:%d sid:%d", dest_id, send_data, self_id);
                end
            dxFlag_r <= 1;
        end
    else
        begin
            dxFlag_r <= 0;
        end  */
    dxFlag_r <= dxFlag;
    dxFlag_rr <= dxFlag_r;
end
```

```
// The way message passing occurs on this net is wave-like
// So if a message comes in from the horizontal directions it is propogated in all directions
// except for the one in which it came.  If it incomes from the vertical direction it continues
// to propogate in that direction.

always @(nBusIn or sBusIn or eBusIn or wBusIn or dxFlag or dxFlag_r or dxFlag_rr or iNetMsgComplete) begin
    if(MRST)
        begin
            n_cnt <= 0;
            s_cnt <= 0;
            e_cnt <= 0;
            w_cnt <= 0;
            rxnFlag <= 0;
            rxsFlag <= 0;
            rxeFlag <= 0;
            rxwFlag <= 0;
            rxnData <= 0;
            rxsData <= 0;
            rxeData <= 0;
            rxwData <= 0;
        end
    else
        begin
            //$display("** ID:%d, dxFlag:%d, dxFlag_r:%d **", self_id, dxFlag, dxFlag_r);

            // Send
            // ************************************************************
            if(dxFlag_r)
                begin
                    if(!dxFlag_rr)
                        begin
                            nBus_R <= {dest_id, send_data, self_id, 4'b0, `north, `north};
                            sBus_R <= {dest_id, send_data, self_id, 4'b0, `south, `south};
                            eBus_R <= {dest_id, send_data, self_id, 4'b0, `east, `east};
                            wBus_R <= {dest_id, send_data, self_id, 4'b0, `west, `west};
                        end
                    else
                        begin
                            nBus_R <= 0;
                            sBus_R <= 0;
                            eBus_R <= 0;
                            wBus_R <= 0;
                        end
                end
            else
                begin
                    nBus_R <= 0;
                    sBus_R <= 0;
                    eBus_R <= 0;
                    wBus_R <= 0;
                end


            // Receive North
            if(nBusIn[`dest_id] == self_id && !n_Thru && nBusIn != 0)
                begin
                    $display("MR_N d_id:%d o_id:%d msg:%d", nBusIn[`dest_id], nBusIn[`origin_id], nBusIn[`msg_data
                        ]);
                    $stop;

                    rxnFlag <= 1;
                    rxnData <= nBusIn[`msg_data];

                    if(n_cnt != 0)
                        begin
                            n_cnt = n_cnt - 8'd1;
                        end
                end
            else if(nBusIn[`dest_id] == 8'hFF)      // Receive Broadcast
                begin
                    $display("MR_N BROADCAST o_id:%d msg:%d", nBusIn[`origin_id], nBusIn[`msg_data]);
                    //$stop;

                    rxnFlag <= 1;
                    rxnData <= nBusIn[`msg_data];

                    // Broadcast automatically resets register thru networks
                    n_cnt <= 0;

                    case(nBusIn[`o_dir])
                        // this should never happen
                        `north: $display("north bus error! n_id: %d", self_id);
```

```
                    // Pass to south bus regardless of last direction
                    `south: sBus_R <= nBusIn;
                    `east:  sBus_R <= nBusIn;
                    `west:  sBus_R <= nBusIn;
                endcase
            end
        else if(nBusIn != 0)
            begin
                // Never Receive on wrong ID
                rxnFlag <= 0;
                rxnData <= 0;

                //n_cnt = n_cnt + 8'd1;

                // Debug Display
                $display("Pass @%d msg N-> dest: %d", self_id, nBusIn[`dest_id]);
                $display("From id %d", nBusIn[`origin_id]);
                $display("Data: %d", nBusIn[`msg_data]);

                case(nBusIn[`o_dir])
                    // this should never happen
                    `north: $display("north bus error! n_id: %d", self_id);
                    // Pass to south bus regardless of last direction
                    `south: sBus_R <= nBusIn;
                    `east:  sBus_R <= nBusIn;
                    `west:  sBus_R <= nBusIn;
                endcase
            end
        else
            begin
                if(rxnFlag && iNetMsgComplete)
                    begin
                        rxnFlag <= 0;
                        rxnData <= 0;
                    end
            end

        // Receive South
        if(sBusIn[`dest_id] == self_id && !s_Thru && sBusIn != 0)
            begin
                $display("MR_S d_id:%d o_id:%d msg:%d", sBusIn[`dest_id], sBusIn[`origin_id], sBusIn[`msg_data
                    ]);
                $stop;

                rxsFlag <= 1;
                rxsData <= sBusIn[`msg_data];

                if(s_cnt != 0)
                    begin
                        s_cnt = s_cnt - 1;
                    end
            end
        else if(sBusIn[`dest_id] == 8'hFF)      // Receive Broadcast
            begin
                $display("MR_S BROADCAST o_id:%d msg:%d", sBusIn[`origin_id], sBusIn[`msg_data]);
                //$stop;

                rxsFlag <= 1;
                rxsData <= sBusIn[`msg_data];

                // Broadcast automatically resets register thru networks
                s_cnt <= 0;

                case(sBusIn[`o_dir])
                    // this should never happen
                    `south: $display("south bus error! n_id: %d", self_id);
                    // Pass to north bus regardless of last direction
                    `north: nBus_R <= sBusIn;
                    `east:  nBus_R <= sBusIn;
                    `west:  nBus_R <= sBusIn;
                endcase
            end
        else if(sBusIn != 0)
            begin
                // Never Receive on wrong ID
                rxsFlag <= 0;
                rxsData <= 0;

                //s_cnt = s_cnt + 1;

                // Debug Display
                $display("Pass @%d msg S-> dest: %d", self_id, sBusIn[`dest_id]);
```

```
                $display("From id %d", sBusIn[`origin_id]);
                $display("Data: %d", sBusIn[`msg_data]);

                case(sBusIn[`o_dir])
                    // this should never happen
                    `south: $display("south bus error! n_id: %d", self_id);
                    // Pass to north bus regardless of last direction
                    `north: nBus_R <= sBusIn;
                    `east:  nBus_R <= sBusIn;
                    `west:  nBus_R <= sBusIn;
                endcase
            end
        else
            begin
                if(rxsFlag && iNetMsgComplete)
                    begin
                        rxsFlag <= 0;
                        rxsData <= 0;
                    end
            end


        // Receive East
        if(eBusIn[`dest_id] == self_id && !e_Thru && eBusIn != 0)
            begin
                $display("MR_E d_id:%d o_id:%d msg:%d", eBusIn[`dest_id], eBusIn[`origin_id], eBusIn[`msg_data
                    ]);
                $stop;

                rxeFlag <= 1;
                rxeData <= eBusIn[`msg_data];

                if(e_cnt != 0)
                    begin
                        e_cnt = e_cnt - 1;
                    end
            end
        else if(eBusIn[`dest_id] == 8'hFF)      // Receive Broadcast
            begin
                $display("MR_E BROADCAST o_id:%d msg:%d", eBusIn[`origin_id], eBusIn[`msg_data]);
                //$stop;

                rxeFlag <= 1;
                rxeData <= eBusIn[`msg_data];

                // Broadcast automatically resets register thru networks
                e_cnt <= 0;

                // Pass on as well
                case(eBusIn[`o_dir])
                    // these should never happen
                    `north: $display("east bus error! n_id: %d", self_id);
                    `south: $display("east bus error! n_id: %d", self_id);
                    `east:  $display("east bus error! n_id: %d", self_id);

                    // All directions except west
                    `west:
                        begin
                            sBus_R <= eBusIn;
                            nBus_R <= eBusIn;
                            wBus_R <= eBusIn;
                        end
                 endcase
            end
        else if(eBusIn != 0)
            begin
                // Never Receive on wrong ID
                rxeFlag <= 0;
                rxeData <= 0;

                //e_cnt = e_cnt + 1;

                // Debug Display
                $display("Pass @%d msg E-> dest: %d", self_id, eBusIn[`dest_id]);
                $display("From id %d", eBusIn[`origin_id]);
                $display("Data: %d", eBusIn[`msg_data]);

                // Message Passing Algorithm
                case(eBusIn[`o_dir])
                        // these should never happen
                        `north: $display("east bus error! n_id: %d", self_id);
                        `south: $display("east bus error! n_id: %d", self_id);
```

```
                    `east:  $display("east bus error! n_id: %d", self_id);

                    // We pass it on in all directions (except east)
                    `west:
                        begin
                            sBus_R <= eBusIn;
                            nBus_R <= eBusIn;
                            wBus_R <= eBusIn;
                        end
                endcase
            end
        else
            begin
                if(rxeFlag && iNetMsgComplete)
                    begin
                        rxeFlag <= 0;
                        rxeData <= 0;
                    end
            end

        // Receive on West
        if(wBusIn[`dest_id] == self_id && !w_Thru && wBusIn != 0)
            begin
                $display("MR_W d_id:%d o_id:%d msg:%d", wBusIn[`dest_id], wBusIn[`origin_id], wBusIn[`msg_data
                    ]);
                //$stop;

                rxwFlag <= 1;
                rxwData <= wBusIn[`msg_data];

                if(w_cnt != 0)
                    begin
                        w_cnt = w_cnt - 1;
                    end
            end
        else if(wBusIn[`dest_id] == 8'hFF)      // Receive Broadcast
            begin
                $display("MR_W BROADCAST o_id:%d msg:%d", wBusIn[`origin_id], wBusIn[`msg_data]);
                //$stop;

                rxwFlag <= 1;
                rxwData <= wBusIn[`msg_data];

                // Broadcast automatically resets register thru networks
                w_cnt <= 0;

                // Pass on as well
                case(wBusIn[`o_dir])
                    // these should never happen
                    `north: $display("west bus error! n_id: %d", self_id);
                    `south: $display("west bus error! n_id: %d", self_id);
                    `west:  $display("west bus error! n_id: %d", self_id);

                    // All directions except west
                    `east:
                        begin
                            sBus_R <= wBusIn;
                            nBus_R <= wBusIn;
                            eBus_R <= wBusIn;
                        end
                 endcase

            end
        else if(wBusIn != 0)
            begin
                // Never Receive on wrong ID
                rxwFlag <= 0;
                rxwData <= 0;

                // w_cnt = w_cnt + 1;

                // Debug Display
                $display("Pass @%d msg W-> dest: %d", self_id, wBusIn[`dest_id]);
                $display("From id %d", wBusIn[`origin_id]);
                $display("Data: %d", wBusIn[`msg_data]);

                // Message Passing Algorithm
                case(wBusIn[`o_dir])
                        // these should never happen
                        `north: $display("west bus error! n_id: %d", self_id);
                        `south: $display("west bus error! n_id: %d", self_id);
                        `west:  $display("west bus error! n_id: %d", self_id);
```

```
                            // We pass it on in all directions (except west)
                            `east:
                                begin
                                    sBus_R <= wBusIn;
                                    nBus_R <= wBusIn;
                                    eBus_R <= wBusIn;
                                end
                        endcase
                    end
                else
                    begin
                        if(rxwFlag && iNetMsgComplete)
                            begin
                                $display("! msg reset W !");
                                rxwFlag <= 0;
                                rxwData <= 0;
                            end
                    end
        end     // END NOT MRST
end

endmodule
```

```
// The

module net_mem_ctrl
(
    iCLK, iMRST,
    iNetIns, iNetInsValid,
    oWAddr, oWData, oWrite,
    oNIP
);

input   wire            iCLK;
input   wire            iMRST;
input   wire    [31:0]  iNetIns;
input   wire            iNetInsValid;


output  reg     [31:0]  oWAddr;
output  reg     [31:0]  oWData;
output  reg             oWrite;

output  reg     [31:0]  oNIP;


always@(posedge iCLK) begin
    if(iMRST)
        begin
            oWAddr <= `NET_MEM_START;
            oNIP <= `NET_MEM_START;
            oWData <= 0;
        end
    else
        begin
            if(iNetInsValid)
                begin
                    // if iNetIns != reset / interrupt
                    case(iNetIns[`op])
                        `JALNET:
                            begin
                                // Jump to location and link

                            end

                        `NACK:
                            begin
                                // Send Back Something
                            end

                        `SNIP:
                            begin
                                // Set network instruction pointer
                                $display("Set Instruction MEM_PTR:%d", iNetIns[`target]);
                                oWAddr <= {{6{1'b0}},iNetIns[`target]};
                                oNIP <= {{6{1'b0}}, iNetIns[`target]};
                            end

                        `NDJR:
                            begin
                                // Network Driven Jump to Register

                            end

                        default:
                            begin
                                oWData <= iNetIns;
                                oWrite <= 1;
                            end
                    endcase
                end
            else
                begin
                    if(oWrite)
                        begin
                            oWrite <= 0;
                            oWAddr <= oWAddr + 1;
                        end
                end
        end
end

endmodule
```

```
module RS232BootLoader
(
    iCLK, iMRST,
    // Transmit
    oASCII,          // Transmit ASCII Value
    oValid_TX,       // Valid Transmit
    iComplete_TX,    // Transmit Complete
    // Recieve
    iASCII,          // Receive ASCII Value
    iBufferEmpty,    // Buffer Empty Value
    oValid_RX,       // Valid Receive
    iComplete_RX,     // Receive Complete
    oState,
    // Boot Loader Hookup
    oBootLoad,
    oPC,
    oIns,
    oWE_N
);

input   wire            iCLK;
input   wire            iMRST;

// Transmit
output  reg     [7:0]   oASCII;
output  reg             oValid_TX = 0;
input   wire            iComplete_TX;

// Receive
input   wire    [7:0]   iASCII;
input   wire            iBufferEmpty;
input   wire            iComplete_RX;
output  reg             oValid_RX = 0;

// Boot Loader Hook Up
output  reg             oBootLoad = 0;
output  wire    [31:0]  oPC;
output  wire    [31:0]  oIns;
output  reg             oWE_N = 1;

assign oPC = rPC;
assign oIns = rIns;

output  wire    [15:0]  oState;
assign oState = rState;

// Internal States
reg     [15:0]  rState = init;
reg     [7:0]   rASCII;

reg     [3:0]   rPC_cnt = 0;
reg     [31:0]  rPC = 0;
reg     [31:0]  rPC_out = 0;

reg     [3:0]   rIns_cnt = 0;
reg     [31:0]  rIns = 0;
reg     [31:0]  rIns_out = 0;


parameter   init = 0,
            // Boot Receive
            boot_pend = 1,
            boot_b = 2, boot_b_rx = 3,
            boot_bo = 4, boot_bo_rx = 5,
            boot_boo = 6, boot_boo_rx = 7,
            boot_boot = 8, boot_boot_rx = 9,
            // Ready Reply
            reply_r = 10, reply_r_tx = 11,
            reply_re = 12, reply_re_tx = 13,
            reply_rea = 14, reply_rea_tx = 15,
            reply_read = 16, reply_read_tx = 17,
            reply_ready = 18, reply_ready_tx = 19,
            // Get PC
            gPC = 20, gPC_rx = 21,
            // Respond PC
            dPC = 22, dPC_tx = 23,
            // Get Instruction
            gIns = 24, gIns_rx = 25,
            // Respond Instruction
            dIns = 26, dIns_tx = 27,
            // Write To Mem
            mem_set = 28, mem_done = 39,
```

```
            // Done
            Done = 30;

always@(posedge iCLK) begin
    if(iMRST)
        begin
            rState <= init;
            oValid_TX <= 0;
            oValid_RX <= 0;
            rPC_cnt <= 0;
            rPC <= 0;
            rIns_cnt <= 0;
            rIns <= 0;
            oWE_N <= 1;
            oBootLoad <= 0;
        end
    else
        begin
            case(rState)
                init:
                    begin
                        // Wait for input of "boot"
                        // "boot" == 0x62, 0x6F, 0x6F, 0x74
                        if(!iBufferEmpty && !iComplete_RX)
                            begin
                                oValid_RX <= 1;
                                rState <= boot_b_rx;
                            end
                    end

                boot_b_rx:
                    begin
                        if(iComplete_RX)
                            begin
                                oValid_RX <= 0;
                                if(iASCII == 8'h62)
                                    begin
                                        rASCII <= iASCII;
                                        rState <= boot_bo;
                                    end
                                else
                                    begin
                                        rState <= init;
                                    end
                            end
                    end

                boot_bo:
                    begin
                        if(!iBufferEmpty && !iComplete_RX)
                            begin
                                oValid_RX <= 1;
                                rState <= boot_bo_rx;
                            end
                    end

                boot_bo_rx:
                    begin
                        if(iComplete_RX)
                            begin
                                oValid_RX <= 0;
                                if(iASCII == 8'h6F)
                                    begin
                                        rASCII <= iASCII;
                                        rState <= boot_boo;
                                    end
                                else
                                    begin
                                        rState <= init;
                                    end
                            end
                    end

                boot_boo:
                    begin
                        if(!iBufferEmpty && !iComplete_RX)
                            begin
                                oValid_RX <= 1;
                                rState <= boot_boo_rx;
                            end
                    end
```

```
                boot_boo_rx:
                    begin
                        if(iComplete_RX)
                            begin
                                oValid_RX <= 0;
                                if(iASCII == 8'h6F)
                                    begin
                                        rASCII <= iASCII;
                                        rState <= boot_boot;
                                    end
                                else
                                    begin
                                        rState <= init;
                                    end
                            end
                    end

                boot_boot:
                    begin
                        if(!iBufferEmpty && !iComplete_RX)
                            begin
                                oValid_RX <= 1;
                                rState <= boot_boot_rx;
                            end
                    end

                boot_boot_rx:
                    begin
                        if(iComplete_RX)
                            begin
                                oValid_RX <= 0;
                                if(iASCII == 8'h74)
                                    begin
                                        rASCII <= iASCII;
                                        oBootLoad <= 1;
                                        rState <= reply_r;
                                    end
                                else
                                    begin
                                        rState <= init;
                                    end
                            end
                    end

                // Reply with "ready"
                // "ready" == 0x72, 0x65, 0x61, 0x64, 0x79
                reply_r:
                    begin
                        if(!iComplete_TX)
                            begin
                                oASCII <= 8'h72;
                                oValid_TX <= 1;
                                rState <= reply_r_tx;
                            end
                    end

                reply_r_tx:
                    begin
                        if(iComplete_TX)
                            begin
                                oValid_TX <= 0;
                                rState <= reply_re;
                            end
                    end

                reply_re:
                    begin
                        if(!iComplete_TX)
                            begin
                                oASCII <= 8'h65;
                                oValid_TX <= 1;
                                rState <= reply_re_tx;
                            end
                    end

                reply_re_tx:
                    begin
                        if(iComplete_TX)
                            begin
                                oValid_TX <= 0;
                                rState <= reply_rea;
                            end
```

```
        end

    reply_rea:
        begin
            if(!iComplete_TX)
                begin
                    oASCII <= 8'h61;
                    oValid_TX <= 1;
                    rState <= reply_rea_tx;
                end
        end

    reply_rea_tx:
        begin
            if(iComplete_TX)
                begin
                    oValid_TX <= 0;
                    rState <= reply_read;
                end
        end

    reply_read:
        begin
            if(!iComplete_TX)
                begin
                    oASCII <= 8'h64;
                    oValid_TX <= 1;
                    rState <= reply_read_tx;
                end
        end

    reply_read_tx:
        begin
            if(iComplete_TX)
                begin
                    oValid_TX <= 0;
                    rState <= reply_ready;
                end
        end

    reply_ready:
        begin
            if(!iComplete_TX)
                begin
                    oASCII <= 8'h79;
                    oValid_TX <= 1;
                    rState <= reply_ready_tx;
                end
        end

    reply_ready_tx:
        begin
            if(iComplete_TX)
                begin
                    oValid_TX <= 0;
                    rState <= gPC;
                end
        end

    gPC:
        begin
            if(!iBufferEmpty && !iComplete_RX)
                begin
                    oValid_RX <= 1;
                    rState <= gPC_rx;
                end
        end

    gPC_rx:
        begin
            if(iComplete_RX)
                begin
                    oValid_RX <= 0;
                    rASCII <= iASCII;
                    rPC <= {iASCII, rPC[31:8]};
                    if(rPC_cnt < 3)
                        begin
                            rPC_cnt <= rPC_cnt + 1;
                            rState <= gPC;
                        end
                    else
                        begin
```

```
                            rPC_cnt <= 0;
                            rPC_out <= {iASCII, rPC[31:8]};
                            rState <= dPC;
                        end
                end
        end

    dPC:
        begin
            if(!iComplete_TX)
                begin
                    oASCII <= rPC_out[7:0];
                    oValid_TX <= 1;
                    rState <= dPC_tx;
                end
        end

    dPC_tx:
        begin
            if(iComplete_TX)
                begin
                    oValid_TX <= 0;
                    if(rPC_cnt < 3)
                        begin
                            rPC_cnt <= rPC_cnt + 1;
                            rPC_out <= {8'h00, rPC_out[31:8]};
                            rState <= dPC;
                        end
                    else
                        begin
                            if(rPC == 32'hFFFFFFFF)
                                begin
                                    // Should not write to
                                    // PC = 0xFFFFFFFF
                                    // (this designates)
                                    // transmission over
                                    rState <= Done;
                                end
                            else
                                begin
                                    rState <= gIns;
                                end
                        end
                end
        end

    gIns:
        begin
            if(!iBufferEmpty && !iComplete_RX)
                begin
                    oValid_RX <= 1;
                    rState <= gIns_rx;
                end
        end

    gIns_rx:
        begin
            if(iComplete_RX)
                begin
                    oValid_RX <= 0;
                    rASCII <= iASCII;
                    rIns <= {iASCII, rIns[31:8]};
                    if(rIns_cnt < 3)
                        begin
                            rIns_cnt <= rIns_cnt + 1;
                            rState <= gIns;
                        end
                    else
                        begin
                            rIns_cnt <= 0;
                            rIns_out <= {iASCII, rIns[31:8]};
                            rState <= dIns;
                        end
                end
        end

    dIns:
        begin
            if(!iComplete_TX)
                begin
                    oASCII <= rIns_out[7:0];
                    oValid_TX <= 1;
```

```
                                rState <= dIns_tx;
                            end
                    end

                dIns_tx:
                    begin
                        if(iComplete_TX)
                            begin
                                oValid_TX <= 0;
                                if(rIns_cnt < 3)
                                    begin
                                        rIns_cnt <= rIns_cnt + 1;
                                        rIns_out <= {8'h00, rIns_out[31:8]};
                                        rState <= dIns;
                                    end
                                else
                                    begin
                                        rState <= mem_set;                                    end
                                    end
                    end

                mem_set:
                    begin
                        if(rPC_cnt == 3 && rIns_cnt == 3)
                            begin
                                oWE_N <= 0;
                                rState <= mem_done;
                            end
                    end

                mem_done:
                    begin
                        oWE_N <= 1;
                        rPC_cnt <= 0;
                        rIns_cnt <= 0;
                        //rIns <= 0;
                        //rPC <= 0;
                        rState <= gPC;
                    end

                Done:
                    begin
                        oWE_N <= 1;
                        rPC_cnt <= 0;
                        rIns_cnt <= 0;
                        oBootLoad <= 0;
                        rState <= init;
                    end
            endcase
        end
end


endmodule
```

```
`include "../../CPU/CPU.v"
`include "../../MEM/M4K/M4KMEM_DUALPORT.v"
`include "../../MEM/M4K/M4KMEM.v"
`include "../../IO/HexDecode/HexDecode.v"
`include "../../VGA/GPU/GPU.v"
`include "../../IO/RS232/RS232InputBuffer.v"
`include "../../IO/PS2/PS2InputBuffer.v"
`include "../../CPU/RS232BootLoader.v"
`include "ndma_pll.v"

module NDMA_TOP
    (
        ////////////////////    Clock Input     ////////////////////
        CLOCK_27,                       //  27 MHz
        CLOCK_50,                       //  50 MHz
        EXT_CLOCK,                      //  External Clock
        ////////////////////    Push Button     ////////////////////
        KEY,                            //  Pushbutton[3:0]
        ////////////////////    DPDT Switch     ////////////////////
        SW,                             //  Toggle Switch[17:0]
        ////////////////////    7-SEG Dispaly   ////////////////////
        HEX0,                           //  Seven Segment Digit 0
        HEX1,                           //  Seven Segment Digit 1
        HEX2,                           //  Seven Segment Digit 2
        HEX3,                           //  Seven Segment Digit 3
        HEX4,                           //  Seven Segment Digit 4
        HEX5,                           //  Seven Segment Digit 5
        HEX6,                           //  Seven Segment Digit 6
        HEX7,                           //  Seven Segment Digit 7
        //////////////////////    LED     //////////////////////
        LEDG,                           //  LED Green[8:0]
        LEDR,                           //  LED Red[17:0]
        //////////////////////    UART    //////////////////////
        UART_TXD,                       //  UART Transmitter
        UART_RXD,                       //  UART Receiver
        /*////////////////////  IRDA    //////////////////////
        IRDA_TXD,                       //  IRDA Transmitter
        IRDA_RXD,                       //  IRDA Receiver   */
        ////////////////////   SDRAM Interface     ////////////////
        DRAM_DQ,                        //  SDRAM Data bus 16 Bits
        DRAM_ADDR,                      //  SDRAM Address bus 12 Bits
        DRAM_LDQM,                      //  SDRAM Low-byte Data Mask
        DRAM_UDQM,                      //  SDRAM High-byte Data Mask
        DRAM_WE_N,                      //  SDRAM Write Enable
        DRAM_CAS_N,                     //  SDRAM Column Address Strobe
        DRAM_RAS_N,                     //  SDRAM Row Address Strobe
        DRAM_CS_N,                      //  SDRAM Chip Select
        DRAM_BA_0,                      //  SDRAM Bank Address 0
        DRAM_BA_1,                      //  SDRAM Bank Address 0
        DRAM_CLK,                       //  SDRAM Clock
        DRAM_CKE,                       //  SDRAM Clock Enable
        ////////////////////    Flash Interface     ////////////////
        FL_DQ,                          //  FLASH Data bus 8 Bits
        FL_ADDR,                        //  FLASH Address bus 22 Bits
        FL_WE_N,                        //  FLASH Write Enable
        FL_RST_N,                       //  FLASH Reset
        FL_OE_N,                        //  FLASH Output Enable
        FL_CE_N,                        //  FLASH Chip Enable
        ////////////////////    SRAM Interface      ////////////////
        SRAM_DQ,                        //  SRAM Data bus 16 Bits
        SRAM_ADDR,                      //  SRAM Address bus 18 Bits
        SRAM_UB_N,                      //  SRAM High-byte Data Mask
        SRAM_LB_N,                      //  SRAM Low-byte Data Mask
        SRAM_WE_N,                      //  SRAM Write Enable
        SRAM_CE_N,                      //  SRAM Chip Enable
        SRAM_OE_N,                      //  SRAM Output Enable
        ////////////////////    ISP1362 Interface   ////////////////
        OTG_DATA,                       //  ISP1362 Data bus 16 Bits
        OTG_ADDR,                       //  ISP1362 Address 2 Bits
        OTG_CS_N,                       //  ISP1362 Chip Select
        OTG_RD_N,                       //  ISP1362 Write
        OTG_WR_N,                       //  ISP1362 Read
        OTG_RST_N,                      //  ISP1362 Reset
        OTG_FSPEED,                     //  USB Full Speed, 0 = Enable, Z = Disable
        OTG_LSPEED,                     //  USB Low Speed,  0 = Enable, Z = Disable
        OTG_INT0,                       //  ISP1362 Interrupt 0
        OTG_INT1,                       //  ISP1362 Interrupt 1
        OTG_DREQ0,                      //  ISP1362 DMA Request 0
        OTG_DREQ1,                      //  ISP1362 DMA Request 1
        OTG_DACK0_N,                    //  ISP1362 DMA Acknowledge 0
        OTG_DACK1_N,                    //  ISP1362 DMA Acknowledge 1
        ////////////////////    LCD Module 16X2     ////////////////
```

```
        LCD_ON,                         //  LCD Power ON/OFF
        LCD_BLON,                       //  LCD Back Light ON/OFF
        LCD_RW,                         //  LCD Read/Write Select, 0 = Write, 1 = Read
        LCD_EN,                         //  LCD Enable
        LCD_RS,                         //  LCD Command/Data Select, 0 = Command, 1 = Data
        LCD_DATA,                       //  LCD Data bus 8 bits
        ////////////////////    SD_Card Interface   ////////////////
        SD_DAT,                         //  SD Card Data
        SD_DAT3,                        //  SD Card Data 3
        SD_CMD,                         //  SD Card Command Signal
        SD_CLK,                         //  SD Card Clock
        ////////////////////    USB JTAG link   ////////////////////
        TDI,                            // CPLD -> FPGA (data in)
        TCK,                            // CPLD -> FPGA (clk)
        TCS,                            // CPLD -> FPGA (CS)
        TDO,                            // FPGA -> CPLD (data out)
        ////////////////////    I2C     ////////////////////////////
        I2C_SDAT,                       //  I2C Data
        I2C_SCLK,                       //  I2C Clock
        ////////////////////    PS2     ////////////////////////////
        PS2_DAT,                        //  PS2 Data
        PS2_CLK,                        //  PS2 Clock
        ////////////////////    VGA     ////////////////////////////
        VGA_CLK,                        //  VGA Clock
        VGA_HS,                         //  VGA H_SYNC
        VGA_VS,                         //  VGA V_SYNC
        VGA_BLANK,                      //  VGA BLANK
        VGA_SYNC,                       //  VGA SYNC
        VGA_R,                          //  VGA Red[9:0]
        VGA_G,                          //  VGA Green[9:0]
        VGA_B,                          //  VGA Blue[9:0]
        ////////////    Ethernet Interface  ////////////////////////
        ENET_DATA,                      //  DM9000A DATA bus 16Bits
        ENET_CMD,                       //  DM9000A Command/Data Select, 0 = Command, 1 = Data
        ENET_CS_N,                      //  DM9000A Chip Select
        ENET_WR_N,                      //  DM9000A Write
        ENET_RD_N,                      //  DM9000A Read
        ENET_RST_N,                     //  DM9000A Reset
        ENET_INT,                       //  DM9000A Interrupt
        ENET_CLK,                       //  DM9000A Clock 25 MHz
        ////////////////    Audio CODEC     ////////////////////////
        AUD_ADCLRCK,                    //  Audio CODEC ADC LR Clock
        AUD_ADCDAT,                     //  Audio CODEC ADC Data
        AUD_DACLRCK,                    //  Audio CODEC DAC LR Clock
        AUD_DACDAT,                     //  Audio CODEC DAC Data
        AUD_BCLK,                       //  Audio CODEC Bit-Stream Clock
        AUD_XCK,                        //  Audio CODEC Chip Clock
        ////////////////    TV Decoder      ////////////////////////
        TD_DATA,                        //  TV Decoder Data bus 8 bits
        TD_HS,                          //  TV Decoder H_SYNC
        TD_VS,                          //  TV Decoder V_SYNC
        TD_RESET,                       //  TV Decoder Reset
        ////////////////////    GPIO    ////////////////////////////
        GPIO_0,                         //  GPIO Connection 0
        GPIO_1                          //  GPIO Connection 1
    );

//////////////////////    Clock Input     ////////////////////////
input           CLOCK_27;               //  27 MHz
input           CLOCK_50;               //  50 MHz
input           EXT_CLOCK;              //  External Clock
//////////////////////    Push Button     ////////////////////////
input   [3:0]   KEY;                    //  Pushbutton[3:0]
//////////////////////    DPDT Switch     ////////////////////////
input   [17:0]  SW;                     //  Toggle Switch[17:0]
//////////////////////    7-SEG Dispaly   ////////////////////////
output  [6:0]   HEX0;                   //  Seven Segment Digit 0
output  [6:0]   HEX1;                   //  Seven Segment Digit 1
output  [6:0]   HEX2;                   //  Seven Segment Digit 2
output  [6:0]   HEX3;                   //  Seven Segment Digit 3
output  [6:0]   HEX4;                   //  Seven Segment Digit 4
output  [6:0]   HEX5;                   //  Seven Segment Digit 5
output  [6:0]   HEX6;                   //  Seven Segment Digit 6
output  [6:0]   HEX7;                   //  Seven Segment Digit 7
////////////////////////////    LED     ////////////////////////////
output  [8:0]   LEDG;                   //  LED Green[8:0]
output  [17:0]  LEDR;                   //  LED Red[17:0]
////////////////////////////    UART    ////////////////////////////
output          UART_TXD;               //  UART Transmitter
input           UART_RXD;               //  UART Receiver
/*////////////////////////////  IRDA    ////////////////////////////
output          IRDA_TXD;               //  IRDA Transmitter
```

```
input           IRDA_RXD;               //  IRDA Receiver*/
//////////////////////     SDRAM Interface //////////////////////
inout   [15:0]  DRAM_DQ;                //  SDRAM Data bus 16 Bits
output  [11:0]  DRAM_ADDR;              //  SDRAM Address bus 12 Bits
output          DRAM_LDQM;              //  SDRAM Low-byte Data Mask
output          DRAM_UDQM;              //  SDRAM High-byte Data Mask
output          DRAM_WE_N;              //  SDRAM Write Enable
output          DRAM_CAS_N;             //  SDRAM Column Address Strobe
output          DRAM_RAS_N;             //  SDRAM Row Address Strobe
output          DRAM_CS_N;              //  SDRAM Chip Select
output          DRAM_BA_0;              //  SDRAM Bank Address 0
output          DRAM_BA_1;              //  SDRAM Bank Address 0
output          DRAM_CLK;               //  SDRAM Clock
output          DRAM_CKE;               //  SDRAM Clock Enable
//////////////////////    Flash Interface //////////////////////
inout   [7:0]   FL_DQ;                  //  FLASH Data bus 8 Bits
output  [21:0]  FL_ADDR;                //  FLASH Address bus 22 Bits
output          FL_WE_N;                //  FLASH Write Enable
output          FL_RST_N;               //  FLASH Reset
output          FL_OE_N;                //  FLASH Output Enable
output          FL_CE_N;                //  FLASH Chip Enable
//////////////////////    SRAM Interface  //////////////////////
inout   [15:0]  SRAM_DQ;                //  SRAM Data bus 16 Bits
output  [17:0]  SRAM_ADDR;              //  SRAM Address bus 18 Bits
output          SRAM_UB_N;              //  SRAM High-byte Data Mask
output          SRAM_LB_N;              //  SRAM Low-byte Data Mask
output          SRAM_WE_N;              //  SRAM Write Enable
output          SRAM_CE_N;              //  SRAM Chip Enable
output          SRAM_OE_N;              //  SRAM Output Enable
////////////////////    ISP1362 Interface   ////////////////////////
inout   [15:0]  OTG_DATA;               //  ISP1362 Data bus 16 Bits
output  [1:0]   OTG_ADDR;               //  ISP1362 Address 2 Bits
output          OTG_CS_N;               //  ISP1362 Chip Select
output          OTG_RD_N;               //  ISP1362 Write
output          OTG_WR_N;               //  ISP1362 Read
output          OTG_RST_N;              //  ISP1362 Reset
output          OTG_FSPEED;             //  USB Full Speed, 0 = Enable, Z = Disable
output          OTG_LSPEED;             //  USB Low Speed,  0 = Enable, Z = Disable
input           OTG_INT0;               //  ISP1362 Interrupt 0
input           OTG_INT1;               //  ISP1362 Interrupt 1
input           OTG_DREQ0;              //  ISP1362 DMA Request 0
input           OTG_DREQ1;              //  ISP1362 DMA Request 1
output          OTG_DACK0_N;            //  ISP1362 DMA Acknowledge 0
output          OTG_DACK1_N;            //  ISP1362 DMA Acknowledge 1
////////////////////    LCD Module 16X2 ////////////////////////////
inout   [7:0]   LCD_DATA;               //  LCD Data bus 8 bits
output          LCD_ON;                 //  LCD Power ON/OFF
output          LCD_BLON;               //  LCD Back Light ON/OFF
output          LCD_RW;                 //  LCD Read/Write Select, 0 = Write, 1 = Read
output          LCD_EN;                 //  LCD Enable
output          LCD_RS;                 //  LCD Command/Data Select, 0 = Command, 1 = Data
////////////////////    SD Card Interface   ////////////////////////
inout           SD_DAT;                 //  SD Card Data
inout           SD_DAT3;                //  SD Card Data 3
inout           SD_CMD;                 //  SD Card Command Signal
output          SD_CLK;                 //  SD Card Clock
//////////////////////    I2C     ////////////////////////////////
inout           I2C_SDAT;               //  I2C Data
output          I2C_SCLK;               //  I2C Clock
//////////////////////    PS2     ////////////////////////////////
input           PS2_DAT;                //  PS2 Data
input           PS2_CLK;                //  PS2 Clock
////////////////////    USB JTAG link   ////////////////////////////
input           TDI;                    // CPLD -> FPGA (data in)
input           TCK;                    // CPLD -> FPGA (clk)
input           TCS;                    // CPLD -> FPGA (CS)
output          TDO;                    // FPGA -> CPLD (data out)
//////////////////////    VGA         ////////////////////////////
output          VGA_CLK;                //  VGA Clock
output          VGA_HS;                 //  VGA H_SYNC
output          VGA_VS;                 //  VGA V_SYNC
output          VGA_BLANK;              //  VGA BLANK
output          VGA_SYNC;               //  VGA SYNC
output  [9:0]   VGA_R;                  //  VGA Red[9:0]
output  [9:0]   VGA_G;                  //  VGA Green[9:0]
output  [9:0]   VGA_B;                  //  VGA Blue[9:0]
////////////////    Ethernet Interface  ////////////////////////////
inout   [15:0]  ENET_DATA;              //  DM9000A DATA bus 16Bits
output          ENET_CMD;               //  DM9000A Command/Data Select, 0 = Command, 1 = Data
output          ENET_CS_N;              //  DM9000A Chip Select
output          ENET_WR_N;              //  DM9000A Write
output          ENET_RD_N;              //  DM9000A Read
```

```
output          ENET_RST_N;             //  DM9000A Reset
input           ENET_INT;               //  DM9000A Interrupt
output          ENET_CLK;               //  DM9000A Clock 25 MHz
////////////////////    Audio CODEC     ////////////////////////////
inout           AUD_ADCLRCK;            //  Audio CODEC ADC LR Clock
input           AUD_ADCDAT;             //  Audio CODEC ADC Data
inout           AUD_DACLRCK;            //  Audio CODEC DAC LR Clock
output          AUD_DACDAT;             //  Audio CODEC DAC Data
inout           AUD_BCLK;               //  Audio CODEC Bit-Stream Clock
output          AUD_XCK;                //  Audio CODEC Chip Clock
////////////////////    TV Devoder      ////////////////////////////
input   [7:0]   TD_DATA;                //  TV Decoder Data bus 8 bits
input           TD_HS;                  //  TV Decoder H_SYNC
input           TD_VS;                  //  TV Decoder V_SYNC
output          TD_RESET;               //  TV Decoder Reset
////////////////////////    GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0;                 //  GPIO Connection 0
inout   [35:0]  GPIO_1;                 //  GPIO Connection 1

//  All inout port turn to tri-state
assign  DRAM_DQ     =   16'hzzzz;
assign  FL_DQ       =   8'hzz;
//assign    SRAM_DQ     =   16'hzzzz;
assign  OTG_DATA    =   16'hzzzz;
assign  LCD_DATA    =   8'hzz;
assign  SD_DAT      =   1'bz;
assign  I2C_SDAT    =   1'bz;
assign  ENET_DATA   =   16'hzzzz;
//assign    AUD_ADCLRCK =   1'bz;
//assign    AUD_DACLRCK =   1'bz;
//assign    AUD_BCLK    =   1'bz;
//assign    GPIO_0      =   36'hzzzzzzzzz;
//assign    GPIO_1      =   36'hzzzzzzzzz;

// Master Control Signals
wire            wMRST = ~KEY[0];
wire            wMCLK;
reg             rMCLK = 0;
reg [3:0]       rMCLK_cnt = 0;

// Try to seperate the CPU clock from the VGA_PLL
always@(posedge CLOCK_27) begin
    if(rMCLK_cnt < 9)
        begin
            rMCLK_cnt <= rMCLK_cnt + 1;
        end
    else
        begin
            rMCLK_cnt <= 0;
            rMCLK <= ~rMCLK;
        end
end

// This will allow switching between single stepped operation
// and full speed operation.
wire            wCLK_SWITCH = (SW[0]) ? rMCLK : ~KEY[1];

// GPU Interface
wire            wGPUComplete;
wire            wGPUValid;
wire    [15:0]  wGPUInstruction;

// PS2 Input Buffer
wire            wPS2BufferEmpty;
wire    [7:0]   wPS2ASCIIOut;
wire            wPS2Complete;
wire            wPS2Valid;

// START CPU 1
wire    [31:0]  PC1, Imem1, dAddr1, Din1, Dout1;
wire            dWrite1;
// CPU 1 Network
wire    [31:0]  nBI1, sBI1, wBI1, eBI1;
wire    [31:0]  nBO1, sBO1, wBO1, eBO1;
// IO
wire    [31:0]  wPortA1, wPortB1, wPortC1, wPortD1;
wire    [31:0]  wPortE1, wPortF1, wPortG1, wPortH1;

// Boot Loader
wire    [7:0]   wASCII_TX;
wire            wValid_TX;
wire            wComplete_TX;
```

```
wire    [7:0]   wASCII_RX;
wire            wBufferEmpty;
wire            wValid_RX;
wire            wComplete_RX;

assign LEDG[0] = wBufferEmpty;
assign LEDG[7] = wBootLoad;
//assign LEDR = wMem1Addr;

wire    [31:0]  wPC;
wire    [31:0]  wIns;
wire            wBootLoad;
wire            wWE_N;
wire    [31:0]  wMem1Addr = (wBootLoad) ? wPC : PC1;

wire            wBootCLK;

//ndma_pll ndmapll(CLOCK_50, wBootCLK);

// CPU 1 BootLoader
RS232InputBuffer rs232
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // RS-232 Connections
   `UART_INTERFACE_DE2,
    // Buffer Interface
    // Transmit
    .iASCII_TX(wASCII_TX),
    .iValid_TX(wValid_TX),
    .oComplete_TX(wComplete_TX),
    // Receive
    .iValid(wValid_RX),
    .oBufferEmpty(wBufferEmpty),
    .oASCIIOut(wASCII_RX),
    .oComplete(wComplete_RX)
);

RS232BootLoader bootload
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // Transmit
    .oASCII(wASCII_TX),             // Transmit ASCII Value
    .oValid_TX(wValid_TX),          // Valid Transmit
    .iComplete_TX(wComplete_TX),    // Transmit Complete
    // Recieve
    .iASCII(wASCII_RX),             // Receive ASCII Value
    .iBufferEmpty(wBufferEmpty),    // Buffer Empty Value
    .oValid_RX(wValid_RX),          // Valid Receive
    .iComplete_RX(wComplete_RX),    // Receive Complete
    //.oState(LEDR),
    .oBootLoad(wBootLoad),
    .oPC(wPC),
    .oIns(wIns),
    .oWE_N(wWE_N)
);

// Instruction Memory CPU 1
M4KMEM m4kmem_ins1(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(wWE_N), .iAddr(wMem1Addr),
    .iWriteData(wIns), .oReadData(Imem1)
);
// m4kmem parameters
defparam m4kmem_ins1.pRAM_ADDR_WIDTH = 9,
         m4kmem_ins1.pRAM_UNIT_COUNT = 512,
         m4kmem_ins1.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins1.pINIT_FILE = "memory.mif";
// end m4kmem parameters

HexDecode hex0(Imem1[3:0], HEX0);
HexDecode hex1(Imem1[7:4], HEX1);
HexDecode hex2(Imem1[11:8], HEX2);
HexDecode hex3(Imem1[15:12], HEX3);
HexDecode hex4(Imem1[19:16], HEX4);
HexDecode hex5(Imem1[23:20], HEX5);
HexDecode hex6(Imem1[27:24], HEX6);
HexDecode hex7(Imem1[31:28], HEX7);

// Data Memory CPU 1
// negative edge triggered ?
M4KMEM m4kmem_data1(
```

```
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite1), .iAddr(dAddr1),
    .iWriteData(Din1), .oReadData(Dout1)
);
// m4kmem parameters
defparam m4kmem_data1.pRAM_ADDR_WIDTH = 8,
         m4kmem_data1.pRAM_UNIT_COUNT = 256,
         m4kmem_data1.pRAM_UNIT_WIDTH = 32,
         m4kmem_data1.pINIT_FILE = "data.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu1
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    .oIMemAddr  (PC1),
    .iMemIn     (Imem1),
    .dMemIn     (Dout1),
    .dAddr      (dAddr1),
    .dWrite     (dWrite1),
    .WD         (Din1),
    // Network Side (dont need for this test)
    .nBusIn     (nBI1), .nBusOut    (nBO1),
    .sBusIn     (nBO3), .sBusOut    (sBO1),
    .eBusIn     (wBO2), .eBusOut    (eBO1),
    .wBusIn     (nBI1), .wBusOut    (wBO1),
    // IO Side
    .iPortA(wPortA1),   // Input From GPU
    .oPortE(wPortE1),   // Output to GPU
    .iPortB(wPortB1),   // Input from PS2 Buffer
    .oPortF(wPortF1)    // Output to PS2 buffer
);
// END CPU 1

assign LEDR = PC1;

// PS2 Input Buffer Belongs to CPU1
// ********************************************
assign wPortB1 = {wPS2BufferEmpty, wPS2Complete, wPS2ASCIIOut};
assign wPS2Valid = wPortF1[0];
assign LEDG[6] = wPS2Valid;
assign LEDG[5] = wPS2BufferEmpty;

PS2InputBuffer ps2buf
(
    .iCLK(CLOCK_50), .iMRST(wMRST | wBootLoad),
    `PS2_INTERFACE_DE2,
    .iValid(wPS2Valid),
    .oBufferEmpty(wPS2BufferEmpty),
    .oASCIIOut(wPS2ASCIIOut),
    .oComplete(wPS2Complete)
);
// ********************************************

// GPU belongs to CPU1
// ********************************************
assign wPortA1 = {15'd0, wGPUComplete};
assign wGPUInstruction = wPortE1[15:0];
assign wGPUValid = wPortE1[16];

GPU gpu
(
    `VGA_CLOCKS_DE2,
    // User Data
    .iReset(wMRST | wBootLoad),
    .iValid(wGPUValid),
    .iIns(wGPUInstruction),
    .oComplete(wGPUComplete),
    .oVGA_CTRL_CLK(wVGA_CTRL_CLK),
    //.oAUDCtrlClk(wAUDCtrlClk),
    //.oDLY_RST(wDLY_RST),
    `VGA_INTERFACE_DE2,
    `VGA_TD_RESET_DE2,
    // SRAM Interface
    `SRAM_INTERFACE_DE2
);
// ********************************************

endmodule
```

```
`include "../../CPU/CPU.v"
`include "../../MEM/M4K/M4KMEM_DUALPORT.v"
`include "../../MEM/M4K/M4KMEM.v"
`include "../../IO/HexDecode/HexDecode.v"
`include "../../VGA/GPU/GPU.v"
`include "../../IO/RS232/RS232InputBuffer.v"
`include "../../IO/PS2/PS2InputBuffer.v"
`include "../../CPU/RS232BootLoader.v"
//`include "ndma_pll.v"

module NDMA2Core_TOP
    (
        ////////////////////    Clock Input     ////////////////////
        CLOCK_27,                       //  27 MHz
        CLOCK_50,                       //  50 MHz
        EXT_CLOCK,                      //  External Clock
        ////////////////////    Push Button     ////////////////////
        KEY,                            //  Pushbutton[3:0]
        ////////////////////    DPDT Switch     ////////////////////
        SW,                             //  Toggle Switch[17:0]
        ////////////////////    7-SEG Dispaly   ////////////////////
        HEX0,                           //  Seven Segment Digit 0
        HEX1,                           //  Seven Segment Digit 1
        HEX2,                           //  Seven Segment Digit 2
        HEX3,                           //  Seven Segment Digit 3
        HEX4,                           //  Seven Segment Digit 4
        HEX5,                           //  Seven Segment Digit 5
        HEX6,                           //  Seven Segment Digit 6
        HEX7,                           //  Seven Segment Digit 7
        //////////////////////////    LED     //////////////////////////
        LEDG,                           //  LED Green[8:0]
        LEDR,                           //  LED Red[17:0]
        //////////////////////////    UART    //////////////////////////
        UART_TXD,                       //  UART Transmitter
        UART_RXD,                       //  UART Receiver
        /*////////////////////////  IRDA    //////////////////////////
        IRDA_TXD,                       //  IRDA Transmitter
        IRDA_RXD,                       //  IRDA Receiver   */
        ////////////////////    SDRAM Interface     ////////////////
        DRAM_DQ,                        //  SDRAM Data bus 16 Bits
        DRAM_ADDR,                      //  SDRAM Address bus 12 Bits
        DRAM_LDQM,                      //  SDRAM Low-byte Data Mask
        DRAM_UDQM,                      //  SDRAM High-byte Data Mask
        DRAM_WE_N,                      //  SDRAM Write Enable
        DRAM_CAS_N,                     //  SDRAM Column Address Strobe
        DRAM_RAS_N,                     //  SDRAM Row Address Strobe
        DRAM_CS_N,                      //  SDRAM Chip Select
        DRAM_BA_0,                      //  SDRAM Bank Address 0
        DRAM_BA_1,                      //  SDRAM Bank Address 0
        DRAM_CLK,                       //  SDRAM Clock
        DRAM_CKE,                       //  SDRAM Clock Enable
        ////////////////////    Flash Interface     ////////////////
        FL_DQ,                          //  FLASH Data bus 8 Bits
        FL_ADDR,                        //  FLASH Address bus 22 Bits
        FL_WE_N,                        //  FLASH Write Enable
        FL_RST_N,                       //  FLASH Reset
        FL_OE_N,                        //  FLASH Output Enable
        FL_CE_N,                        //  FLASH Chip Enable
        ////////////////////    SRAM Interface      ////////////////
        SRAM_DQ,                        //  SRAM Data bus 16 Bits
        SRAM_ADDR,                      //  SRAM Address bus 18 Bits
        SRAM_UB_N,                      //  SRAM High-byte Data Mask
        SRAM_LB_N,                      //  SRAM Low-byte Data Mask
        SRAM_WE_N,                      //  SRAM Write Enable
        SRAM_CE_N,                      //  SRAM Chip Enable
        SRAM_OE_N,                      //  SRAM Output Enable
        ////////////////////    ISP1362 Interface   ////////////////
        OTG_DATA,                       //  ISP1362 Data bus 16 Bits
        OTG_ADDR,                       //  ISP1362 Address 2 Bits
        OTG_CS_N,                       //  ISP1362 Chip Select
        OTG_RD_N,                       //  ISP1362 Write
        OTG_WR_N,                       //  ISP1362 Read
        OTG_RST_N,                      //  ISP1362 Reset
        OTG_FSPEED,                     //  USB Full Speed, 0 = Enable, Z = Disable
        OTG_LSPEED,                     //  USB Low Speed,  0 = Enable, Z = Disable
        OTG_INT0,                       //  ISP1362 Interrupt 0
        OTG_INT1,                       //  ISP1362 Interrupt 1
        OTG_DREQ0,                      //  ISP1362 DMA Request 0
        OTG_DREQ1,                      //  ISP1362 DMA Request 1
        OTG_DACK0_N,                    //  ISP1362 DMA Acknowledge 0
        OTG_DACK1_N,                    //  ISP1362 DMA Acknowledge 1
        ////////////////////    LCD Module 16X2     ////////////////
```

```
        LCD_ON,                         //  LCD Power ON/OFF
        LCD_BLON,                       //  LCD Back Light ON/OFF
        LCD_RW,                         //  LCD Read/Write Select, 0 = Write, 1 = Read
        LCD_EN,                         //  LCD Enable
        LCD_RS,                         //  LCD Command/Data Select, 0 = Command, 1 = Data
        LCD_DATA,                       //  LCD Data bus 8 bits
        ////////////////////    SD_Card Interface   ////////////////
        SD_DAT,                         //  SD Card Data
        SD_DAT3,                        //  SD Card Data 3
        SD_CMD,                         //  SD Card Command Signal
        SD_CLK,                         //  SD Card Clock
        ////////////////////    USB JTAG link   ////////////////////
        TDI,                            // CPLD -> FPGA (data in)
        TCK,                            // CPLD -> FPGA (clk)
        TCS,                            // CPLD -> FPGA (CS)
        TDO,                            // FPGA -> CPLD (data out)
        ////////////////////    I2C     ////////////////////////////
        I2C_SDAT,                       //  I2C Data
        I2C_SCLK,                       //  I2C Clock
        ////////////////////    PS2     ////////////////////////////
        PS2_DAT,                        //  PS2 Data
        PS2_CLK,                        //  PS2 Clock
        ////////////////////    VGA     ////////////////////////////
        VGA_CLK,                        //  VGA Clock
        VGA_HS,                         //  VGA H_SYNC
        VGA_VS,                         //  VGA V_SYNC
        VGA_BLANK,                      //  VGA BLANK
        VGA_SYNC,                       //  VGA SYNC
        VGA_R,                          //  VGA Red[9:0]
        VGA_G,                          //  VGA Green[9:0]
        VGA_B,                          //  VGA Blue[9:0]
        ////////////    Ethernet Interface  ////////////////////////
        ENET_DATA,                      //  DM9000A DATA bus 16Bits
        ENET_CMD,                       //  DM9000A Command/Data Select, 0 = Command, 1 = Data
        ENET_CS_N,                      //  DM9000A Chip Select
        ENET_WR_N,                      //  DM9000A Write
        ENET_RD_N,                      //  DM9000A Read
        ENET_RST_N,                     //  DM9000A Reset
        ENET_INT,                       //  DM9000A Interrupt
        ENET_CLK,                       //  DM9000A Clock 25 MHz
        ////////////////    Audio CODEC     ////////////////////////
        AUD_ADCLRCK,                    //  Audio CODEC ADC LR Clock
        AUD_ADCDAT,                     //  Audio CODEC ADC Data
        AUD_DACLRCK,                    //  Audio CODEC DAC LR Clock
        AUD_DACDAT,                     //  Audio CODEC DAC Data
        AUD_BCLK,                       //  Audio CODEC Bit-Stream Clock
        AUD_XCK,                        //  Audio CODEC Chip Clock
        ////////////////    TV Decoder      ////////////////////////
        TD_DATA,                        //  TV Decoder Data bus 8 bits
        TD_HS,                          //  TV Decoder H_SYNC
        TD_VS,                          //  TV Decoder V_SYNC
        TD_RESET,                       //  TV Decoder Reset
        ////////////////////    GPIO    ////////////////////////////
        GPIO_0,                         //  GPIO Connection 0
        GPIO_1                          //  GPIO Connection 1
    );

//////////////////////    Clock Input     ////////////////////////
input           CLOCK_27;               //  27 MHz
input           CLOCK_50;               //  50 MHz
input           EXT_CLOCK;              //  External Clock
//////////////////////    Push Button     ////////////////////////
input   [3:0]   KEY;                    //  Pushbutton[3:0]
//////////////////////    DPDT Switch     ////////////////////////
input   [17:0]  SW;                     //  Toggle Switch[17:0]
//////////////////////    7-SEG Dispaly   ////////////////////////
output  [6:0]   HEX0;                   //  Seven Segment Digit 0
output  [6:0]   HEX1;                   //  Seven Segment Digit 1
output  [6:0]   HEX2;                   //  Seven Segment Digit 2
output  [6:0]   HEX3;                   //  Seven Segment Digit 3
output  [6:0]   HEX4;                   //  Seven Segment Digit 4
output  [6:0]   HEX5;                   //  Seven Segment Digit 5
output  [6:0]   HEX6;                   //  Seven Segment Digit 6
output  [6:0]   HEX7;                   //  Seven Segment Digit 7
////////////////////////////    LED     ////////////////////////////
output  [8:0]   LEDG;                   //  LED Green[8:0]
output  [17:0]  LEDR;                   //  LED Red[17:0]
////////////////////////////    UART    ////////////////////////////
output          UART_TXD;               //  UART Transmitter
input           UART_RXD;               //  UART Receiver
/*////////////////////////////  IRDA    ////////////////////////////
output          IRDA_TXD;               //  IRDA Transmitter
```

```
input           IRDA_RXD;               //  IRDA Receiver*/
//////////////////////    SDRAM Interface //////////////////////
inout   [15:0]  DRAM_DQ;                //  SDRAM Data bus 16 Bits
output  [11:0]  DRAM_ADDR;              //  SDRAM Address bus 12 Bits
output          DRAM_LDQM;              //  SDRAM Low-byte Data Mask
output          DRAM_UDQM;              //  SDRAM High-byte Data Mask
output          DRAM_WE_N;              //  SDRAM Write Enable
output          DRAM_CAS_N;             //  SDRAM Column Address Strobe
output          DRAM_RAS_N;             //  SDRAM Row Address Strobe
output          DRAM_CS_N;              //  SDRAM Chip Select
output          DRAM_BA_0;              //  SDRAM Bank Address 0
output          DRAM_BA_1;              //  SDRAM Bank Address 0
output          DRAM_CLK;               //  SDRAM Clock
output          DRAM_CKE;               //  SDRAM Clock Enable
//////////////////////    Flash Interface //////////////////////
inout   [7:0]   FL_DQ;                  //  FLASH Data bus 8 Bits
output  [21:0]  FL_ADDR;                //  FLASH Address bus 22 Bits
output          FL_WE_N;                //  FLASH Write Enable
output          FL_RST_N;               //  FLASH Reset
output          FL_OE_N;                //  FLASH Output Enable
output          FL_CE_N;                //  FLASH Chip Enable
//////////////////////    SRAM Interface  //////////////////////
inout   [15:0]  SRAM_DQ;                //  SRAM Data bus 16 Bits
output  [17:0]  SRAM_ADDR;              //  SRAM Address bus 18 Bits
output          SRAM_UB_N;              //  SRAM High-byte Data Mask
output          SRAM_LB_N;              //  SRAM Low-byte Data Mask
output          SRAM_WE_N;              //  SRAM Write Enable
output          SRAM_CE_N;              //  SRAM Chip Enable
output          SRAM_OE_N;              //  SRAM Output Enable
////////////////////    ISP1362 Interface   ////////////////////////
inout   [15:0]  OTG_DATA;               //  ISP1362 Data bus 16 Bits
output  [1:0]   OTG_ADDR;               //  ISP1362 Address 2 Bits
output          OTG_CS_N;               //  ISP1362 Chip Select
output          OTG_RD_N;               //  ISP1362 Write
output          OTG_WR_N;               //  ISP1362 Read
output          OTG_RST_N;              //  ISP1362 Reset
output          OTG_FSPEED;             //  USB Full Speed, 0 = Enable, Z = Disable
output          OTG_LSPEED;             //  USB Low Speed,  0 = Enable, Z = Disable
input           OTG_INT0;               //  ISP1362 Interrupt 0
input           OTG_INT1;               //  ISP1362 Interrupt 1
input           OTG_DREQ0;              //  ISP1362 DMA Request 0
input           OTG_DREQ1;              //  ISP1362 DMA Request 1
output          OTG_DACK0_N;            //  ISP1362 DMA Acknowledge 0
output          OTG_DACK1_N;            //  ISP1362 DMA Acknowledge 1
////////////////////    LCD Module 16X2 ////////////////////////////
inout   [7:0]   LCD_DATA;               //  LCD Data bus 8 bits
output          LCD_ON;                 //  LCD Power ON/OFF
output          LCD_BLON;               //  LCD Back Light ON/OFF
output          LCD_RW;                 //  LCD Read/Write Select, 0 = Write, 1 = Read
output          LCD_EN;                 //  LCD Enable
output          LCD_RS;                 //  LCD Command/Data Select, 0 = Command, 1 = Data
////////////////////    SD Card Interface   ////////////////////////
inout           SD_DAT;                 //  SD Card Data
inout           SD_DAT3;                //  SD Card Data 3
inout           SD_CMD;                 //  SD Card Command Signal
output          SD_CLK;                 //  SD Card Clock
//////////////////////    I2C     ////////////////////////////////
inout           I2C_SDAT;               //  I2C Data
output          I2C_SCLK;               //  I2C Clock
//////////////////////    PS2     ////////////////////////////////
input           PS2_DAT;                //  PS2 Data
input           PS2_CLK;                //  PS2 Clock
////////////////////    USB JTAG link   ////////////////////////////
input           TDI;                    // CPLD -> FPGA (data in)
input           TCK;                    // CPLD -> FPGA (clk)
input           TCS;                    // CPLD -> FPGA (CS)
output          TDO;                    // FPGA -> CPLD (data out)
//////////////////////    VGA         ////////////////////////////
output          VGA_CLK;                //  VGA Clock
output          VGA_HS;                 //  VGA H_SYNC
output          VGA_VS;                 //  VGA V_SYNC
output          VGA_BLANK;              //  VGA BLANK
output          VGA_SYNC;               //  VGA SYNC
output  [9:0]   VGA_R;                  //  VGA Red[9:0]
output  [9:0]   VGA_G;                  //  VGA Green[9:0]
output  [9:0]   VGA_B;                  //  VGA Blue[9:0]
////////////////    Ethernet Interface  ////////////////////////////
inout   [15:0]  ENET_DATA;              //  DM9000A DATA bus 16Bits
output          ENET_CMD;               //  DM9000A Command/Data Select, 0 = Command, 1 = Data
output          ENET_CS_N;              //  DM9000A Chip Select
output          ENET_WR_N;              //  DM9000A Write
output          ENET_RD_N;              //  DM9000A Read
```

```
output          ENET_RST_N;             //  DM9000A Reset
input           ENET_INT;               //  DM9000A Interrupt
output          ENET_CLK;               //  DM9000A Clock 25 MHz
////////////////////    Audio CODEC     ////////////////////////////
inout           AUD_ADCLRCK;            //  Audio CODEC ADC LR Clock
input           AUD_ADCDAT;             //  Audio CODEC ADC Data
inout           AUD_DACLRCK;            //  Audio CODEC DAC LR Clock
output          AUD_DACDAT;             //  Audio CODEC DAC Data
inout           AUD_BCLK;               //  Audio CODEC Bit-Stream Clock
output          AUD_XCK;                //  Audio CODEC Chip Clock
////////////////////    TV Devoder      ////////////////////////////
input   [7:0]   TD_DATA;                //  TV Decoder Data bus 8 bits
input           TD_HS;                  //  TV Decoder H_SYNC
input           TD_VS;                  //  TV Decoder V_SYNC
output          TD_RESET;               //  TV Decoder Reset
////////////////////////    GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0;                 //  GPIO Connection 0
inout   [35:0]  GPIO_1;                 //  GPIO Connection 1

//  All inout port turn to tri-state
assign  DRAM_DQ     =   16'hzzzz;
assign  FL_DQ       =   8'hzz;
//assign    SRAM_DQ     =   16'hzzzz;
assign  OTG_DATA    =   16'hzzzz;
assign  LCD_DATA    =   8'hzz;
assign  SD_DAT      =   1'bz;
assign  I2C_SDAT    =   1'bz;
assign  ENET_DATA   =   16'hzzzz;
//assign    AUD_ADCLRCK =   1'bz;
//assign    AUD_DACLRCK =   1'bz;
//assign    AUD_BCLK    =   1'bz;
//assign    GPIO_0      =   36'hzzzzzzzzz;
//assign    GPIO_1      =   36'hzzzzzzzzz;

// Master Control Signals
wire            wMRST = ~KEY[0];
wire            wMCLK;
reg             rMCLK = 0;
reg [3:0]       rMCLK_cnt = 0;

// Try to seperate the CPU clock from the VGA_PLL
always@(posedge CLOCK_27) begin
    if(rMCLK_cnt < 9)
        begin
            rMCLK_cnt <= rMCLK_cnt + 1;
        end
    else
        begin
            rMCLK_cnt <= 0;
            rMCLK <= ~rMCLK;
        end
end

// This will allow switching between single stepped operation
// and full speed operation.
wire            wCLK_SWITCH = (SW[0]) ? rMCLK : ~KEY[1];

// GPU Interface
wire            wGPUComplete;
wire            wGPUValid;
wire    [15:0]  wGPUInstruction;

// PS2 Input Buffer
wire            wPS2BufferEmpty;
wire    [7:0]   wPS2ASCIIOut;
wire            wPS2Complete;
wire            wPS2Valid;


// Boot Loader
wire    [7:0]   wASCII_TX;
wire            wValid_TX;
wire            wComplete_TX;

wire    [7:0]   wASCII_RX;
wire            wBufferEmpty;
wire            wValid_RX;
wire            wComplete_RX;

//assign LEDG[0] = wBufferEmpty;

//assign LEDR = wMem1Addr;
```

```
wire    [31:0]  wPC;
wire    [31:0]  wIns;
wire            wBootLoad;
wire            wWE_N;

// If bootloading drive the instruction memory with boot loader
wire    [31:0]  wMem1Write = (wBootLoad) ? wWE_N : ~wIWriteEnable1;
wire    [31:0]  wMem1Addr = (wBootLoad) ? wPC : wIWriteAddr1;
wire    [31:0]  wMem1Data = (wBootLoad) ? wIns : wIWriteData1;

wire    [31:0]  wMem2Addr = PC2;

wire            wBootCLK;

//ndma_pll ndmapll(CLOCK_50, wBootCLK);

// CPU 1 BootLoader
RS232InputBuffer rs232
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // RS-232 Connections
   `UART_INTERFACE_DE2,
    // Buffer Interface
    // Transmit
    .iASCII_TX(wASCII_TX),
    .iValid_TX(wValid_TX),
    .oComplete_TX(wComplete_TX),
    // Receive
    .iValid(wValid_RX),
    .oBufferEmpty(wBufferEmpty),
    .oASCIIOut(wASCII_RX),
    .oComplete(wComplete_RX)
);

RS232BootLoader bootload
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // Transmit
    .oASCII(wASCII_TX),             // Transmit ASCII Value
    .oValid_TX(wValid_TX),          // Valid Transmit
    .iComplete_TX(wComplete_TX),    // Transmit Complete
    // Recieve
    .iASCII(wASCII_RX),             // Receive ASCII Value
    .iBufferEmpty(wBufferEmpty),    // Buffer Empty Value
    .oValid_RX(wValid_RX),          // Valid Receive
    .iComplete_RX(wComplete_RX),    // Receive Complete
    //.oState(LEDR),
    .oBootLoad(wBootLoad),
    .oPC(wPC),
    .oIns(wIns),
    .oWE_N(wWE_N)
);

// Debugging
// ****************************************************************

wire    [31:0]  HexInsOut = (SW[17]) ? Imem2 : Imem1;

wire    [31:0]  HexOut = (SW[16]) ? HexCPU1EastOut : HexInsOut;

wire    [31:0]  HexCPU1EastOut = eBO1;

wire    [31:0]  HexCPU2WestOut = wBO2;

HexDecode hex0(HexOut[3:0], HEX0);
HexDecode hex1(HexOut[7:4], HEX1);
HexDecode hex2(HexOut[11:8], HEX2);
HexDecode hex3(HexOut[15:12], HEX3);
HexDecode hex4(HexOut[19:16], HEX4);
HexDecode hex5(HexOut[23:20], HEX5);
HexDecode hex6(HexOut[27:24], HEX6);
HexDecode hex7(HexOut[31:28], HEX7);

assign LEDR = (SW[17]) ? PC2 : PC1;

//assign LEDG = (SW[17]) ? wSelfID2 : wSelfID1;
assign LEDG[7] = wDxFlag1;
assign LEDG[6] = wRxFlag2;


assign LEDG[5] = wInsBufCnt[1];
```

```
assign LEDG[4] = wInsBufCnt[0];

assign LEDG[3] = wSelfID1[1];
assign LEDG[2] = wSelfID1[0];

assign LEDG[1] = wSelfID2[1];
assign LEDG[0] = wSelfID2[0];

// ********************************************************************


// CPU 1
// ********************************************************************

wire    [31:0]  PC1, Imem1, dAddr1, Din1, Dout1;
wire            dWrite1;
// CPU 1 Network
wire    [31:0]  nBI1, sBI1, wBI1, eBI1;
wire    [31:0]  nBO1, sBO1, wBO1, eBO1;
// IO
wire    [31:0]  wPortA1, wPortB1, wPortC1, wPortD1;
wire    [31:0]  wPortE1, wPortF1, wPortG1, wPortH1;

// Network Memory
wire    [31:0]  wIWriteAddr1;
wire    [31:0]  wIWriteData1;
wire    [31:0]  wIWriteEnable1;

// Debuggins
wire    [7:0]   wSelfID1;
wire            wDxFlag1;

// Instruction Memory CPU 1
M4KMEM_DUALPORT m4kmem_ins1(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(wMem1Write),
    .iWriteAddr(wMem1Addr),
    .iWriteData(wMem1Data),
    // Read Side
    .iReadAddr(PC1),
    .oReadData(Imem1)
);
// m4kmem parameters
defparam m4kmem_ins1.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins1.pRAM_UNIT_COUNT = 1024,
         m4kmem_ins1.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins1.pINIT_FILE = "memory1.mif";
// end m4kmem parameters



// Data Memory CPU 1
// negative edge triggered ?
M4KMEM m4kmem_data1(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite1), .iAddr(dAddr1),
    .iWriteData(Din1), .oReadData(Dout1)
);
// m4kmem parameters
defparam m4kmem_data1.pRAM_ADDR_WIDTH = 10,
         m4kmem_data1.pRAM_UNIT_COUNT = 1024,
         m4kmem_data1.pRAM_UNIT_WIDTH = 32,
         m4kmem_data1.pINIT_FILE = "data1.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu1
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    .oIMemAddr  (PC1),
    .iMemIn     (Imem1),
    // Network Memory
    .oIWAddr    (wIWriteAddr1),
    .oIWrite    (wIWriteEnable1),
    .oIData     (wIWriteData1),
    // Data Memory
    .dMemIn     (Dout1),
    .dAddr      (dAddr1),
    .dWrite     (dWrite1),
    .WD         (Din1),
```

```
    // Network Side
    .nBusIn     (nBI1), .nBusOut    (nBO1),
    .sBusIn     (nBO3), .sBusOut    (sBO1),
    .eBusIn     (wBO2), .eBusOut    (eBO1),
    .wBusIn     (nBI1), .wBusOut    (wBO1),
    // IO Side
    .iPortA(wPortA1),   // Input From GPU
    .oPortE(wPortE1),   // Output to GPU
    .iPortB(wPortB1),   // Input from PS2 Buffer
    .oPortF(wPortF1),   // Output to PS2 buffer
    // dubbings
    .oSelfID(wSelfID1),
    .oDxFlag(wDxFlag1)
);



// ***********************************************************************


// CPU 2
// ***********************************************************************
// Note that CPU 2 is initialized with the following code:
// sid 2
// nop
// nop
// j 1

wire    [31:0]  PC2, Imem2, dAddr2, Din2, Dout2;
wire            dWrite2;
// CPU 2 Network
wire    [31:0]  nBI2, sBI2, wBI2, eBI2;
wire    [31:0]  nBO2, sBO2, wBO2, eBO2;
// IO
wire    [31:0]  wPortA2, wPortB2, wPortC2, wPortD2;
wire    [31:0]  wPortE2, wPortF2, wPortG2, wPortH2;
// Network Connections
wire    [31:0]  wIWriteAddr2;
wire    [31:0]  wIWriteData2;
wire    [31:0]  wIWriteEnable2;

wire    [7:0]   wSelfID2;
wire            wRxFlag2;
wire    [1:0]   wInsBufCnt;

// Instruction Memory CPU 2
M4KMEM_DUALPORT m4kmem_ins2(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable2),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr2),
    .iWriteData(wIWriteData2),
    // Read Side
    .iReadAddr(wMem2Addr),
    .oReadData(Imem2)
);
// m4kmem parameters
defparam m4kmem_ins2.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins2.pRAM_UNIT_COUNT = 1024,
         m4kmem_ins2.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins2.pINIT_FILE = "memory2.mif";
// end m4kmem parameters


// Data Memory CPU 2
M4KMEM m4kmem_data2(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite2), .iAddr(dAddr2),
    .iWriteData(Din2), .oReadData(Dout2)
);
// m4kmem parameters
defparam m4kmem_data2.pRAM_ADDR_WIDTH = 10,
         m4kmem_data2.pRAM_UNIT_COUNT = 1024,
         m4kmem_data2.pRAM_UNIT_WIDTH = 32,
         m4kmem_data2.pINIT_FILE = "data2.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu2
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
```

```
    // Instruction Memory
    .oIMemAddr  (PC2),
    .iMemIn     (Imem2),
    // Network Memory
    .oIWAddr    (wIWriteAddr2),
    .oIWrite    (wIWriteEnable2),
    .oIData     (wIWriteData2),
    // Data Memory
    .dMemIn     (Dout2),
    .dAddr      (dAddr2),
    .dWrite     (dWrite2),
    .WD         (Din2),
    // Network Side
    .nBusIn     (nBI2), .nBusOut    (nBO2),
    .sBusIn     (nBO4), .sBusOut    (sBO2),
    .eBusIn     (eBI2), .eBusOut    (eBO2),
    .wBusIn     (eBO1), .wBusOut    (wBO2),
    // IO Side
    .iPortA(wPortA2),   // Input From GPU
    .oPortE(wPortE2),   // Output to GPU
    .iPortB(wPortB2),   // Input from PS2 Buffer
    .oPortF(wPortF2),   // Output to PS2 buffer
    // SELF ID
    .oSelfID(wSelfID2),
    .oRxFlag(wRxFlag2),
    .oInsBufCnt(wInsBufCnt)
);
// ********************************************************************


// ********************************************************************
// SYSTEM RESOURCES
// ********************************************************************

// PS2 Input Buffer Belongs to CPU1
// ********************************************
assign wPortB1 = {wPS2BufferEmpty, wPS2Complete, wPS2ASCIIOut};
assign wPS2Valid = wPortF1[0];
//assign LEDG[6] = wPS2Valid;
//assign LEDG[5] = wPS2BufferEmpty;

PS2InputBuffer ps2buf
(
    .iCLK(CLOCK_50), .iMRST(wMRST | wBootLoad),
    `PS2_INTERFACE_DE2,
    .iValid(wPS2Valid),
    .oBufferEmpty(wPS2BufferEmpty),
    .oASCIIOut(wPS2ASCIIOut),
    .oComplete(wPS2Complete)
);
// ********************************************

// GPU belongs to CPU1
// ********************************************
assign wPortA1 = {15'd0, wGPUComplete};
assign wGPUInstruction = wPortE1[15:0];
assign wGPUValid = wPortE1[16];

GPU gpu
(
    `VGA_CLOCKS_DE2,
    // User Data
    .iReset(wMRST | wBootLoad),
    .iValid(wGPUValid),
    .iIns(wGPUInstruction),
    .oComplete(wGPUComplete),
    .oVGA_CTRL_CLK(wVGA_CTRL_CLK),
    //.oAUDCtrlClk(wAUDCtrlClk),
    //.oDLY_RST(wDLY_RST),
    `VGA_INTERFACE_DE2,
    `VGA_TD_RESET_DE2,
    // SRAM Interface
    `SRAM_INTERFACE_DE2
);
// ********************************************

endmodule
```

```
`include "../../CPU/CPU.v"
`include "../../MEM/M4K/M4KMEM_DUALPORT.v"
`include "../../MEM/M4K/M4KMEM.v"
`include "../../IO/HexDecode/HexDecode.v"
`include "../../VGA/GPU/GPU.v"
`include "../../IO/RS232/RS232InputBuffer.v"
`include "../../IO/PS2/PS2InputBuffer.v"
`include "../../CPU/RS232BootLoader.v"

module NDMA4Core_TOP
    (
        ////////////////////    Clock Input     ////////////////////
        CLOCK_27,                       //  27 MHz
        CLOCK_50,                       //  50 MHz
        EXT_CLOCK,                      //  External Clock
        ////////////////////    Push Button     ////////////////////
        KEY,                            //  Pushbutton[3:0]
        ////////////////////    DPDT Switch     ////////////////////
        SW,                             //  Toggle Switch[17:0]
        ////////////////////    7-SEG Dispaly   ////////////////////
        HEX0,                           //  Seven Segment Digit 0
        HEX1,                           //  Seven Segment Digit 1
        HEX2,                           //  Seven Segment Digit 2
        HEX3,                           //  Seven Segment Digit 3
        HEX4,                           //  Seven Segment Digit 4
        HEX5,                           //  Seven Segment Digit 5
        HEX6,                           //  Seven Segment Digit 6
        HEX7,                           //  Seven Segment Digit 7
        //////////////////////    LED     //////////////////////
        LEDG,                           //  LED Green[8:0]
        LEDR,                           //  LED Red[17:0]
        //////////////////////    UART    //////////////////////
        UART_TXD,                       //  UART Transmitter
        UART_RXD,                       //  UART Receiver
        /*////////////////////////  IRDA    ////////////////////////
        IRDA_TXD,                       //  IRDA Transmitter
        IRDA_RXD,                       //  IRDA Receiver   */
        ////////////////////   SDRAM Interface     ////////////////
        DRAM_DQ,                        //  SDRAM Data bus 16 Bits
        DRAM_ADDR,                      //  SDRAM Address bus 12 Bits
        DRAM_LDQM,                      //  SDRAM Low-byte Data Mask
        DRAM_UDQM,                      //  SDRAM High-byte Data Mask
        DRAM_WE_N,                      //  SDRAM Write Enable
        DRAM_CAS_N,                     //  SDRAM Column Address Strobe
        DRAM_RAS_N,                     //  SDRAM Row Address Strobe
        DRAM_CS_N,                      //  SDRAM Chip Select
        DRAM_BA_0,                      //  SDRAM Bank Address 0
        DRAM_BA_1,                      //  SDRAM Bank Address 0
        DRAM_CLK,                       //  SDRAM Clock
        DRAM_CKE,                       //  SDRAM Clock Enable
        ////////////////////    Flash Interface     ////////////////
        FL_DQ,                          //  FLASH Data bus 8 Bits
        FL_ADDR,                        //  FLASH Address bus 22 Bits
        FL_WE_N,                        //  FLASH Write Enable
        FL_RST_N,                       //  FLASH Reset
        FL_OE_N,                        //  FLASH Output Enable
        FL_CE_N,                        //  FLASH Chip Enable
        ////////////////////    SRAM Interface      ////////////////
        SRAM_DQ,                        //  SRAM Data bus 16 Bits
        SRAM_ADDR,                      //  SRAM Address bus 18 Bits
        SRAM_UB_N,                      //  SRAM High-byte Data Mask
        SRAM_LB_N,                      //  SRAM Low-byte Data Mask
        SRAM_WE_N,                      //  SRAM Write Enable
        SRAM_CE_N,                      //  SRAM Chip Enable
        SRAM_OE_N,                      //  SRAM Output Enable
        ////////////////////    ISP1362 Interface   ////////////////
        OTG_DATA,                       //  ISP1362 Data bus 16 Bits
        OTG_ADDR,                       //  ISP1362 Address 2 Bits
        OTG_CS_N,                       //  ISP1362 Chip Select
        OTG_RD_N,                       //  ISP1362 Write
        OTG_WR_N,                       //  ISP1362 Read
        OTG_RST_N,                      //  ISP1362 Reset
        OTG_FSPEED,                     //  USB Full Speed, 0 = Enable, Z = Disable
        OTG_LSPEED,                     //  USB Low Speed,  0 = Enable, Z = Disable
        OTG_INT0,                       //  ISP1362 Interrupt 0
        OTG_INT1,                       //  ISP1362 Interrupt 1
        OTG_DREQ0,                      //  ISP1362 DMA Request 0
        OTG_DREQ1,                      //  ISP1362 DMA Request 1
        OTG_DACK0_N,                    //  ISP1362 DMA Acknowledge 0
        OTG_DACK1_N,                    //  ISP1362 DMA Acknowledge 1
        ////////////////////    LCD Module 16X2     ////////////////
        LCD_ON,                         //  LCD Power ON/OFF
```

```
        LCD_BLON,                       //  LCD Back Light ON/OFF
        LCD_RW,                         //  LCD Read/Write Select, 0 = Write, 1 = Read
        LCD_EN,                         //  LCD Enable
        LCD_RS,                         //  LCD Command/Data Select, 0 = Command, 1 = Data
        LCD_DATA,                       //  LCD Data bus 8 bits
        ////////////////////    SD_Card Interface   ////////////////
        SD_DAT,                         //  SD Card Data
        SD_DAT3,                        //  SD Card Data 3
        SD_CMD,                         //  SD Card Command Signal
        SD_CLK,                         //  SD Card Clock
        ////////////////////    USB JTAG link   ////////////////////
        TDI,                            // CPLD -> FPGA (data in)
        TCK,                            // CPLD -> FPGA (clk)
        TCS,                            // CPLD -> FPGA (CS)
        TDO,                            // FPGA -> CPLD (data out)
        ////////////////////    I2C     ////////////////////////////
        I2C_SDAT,                       //  I2C Data
        I2C_SCLK,                       //  I2C Clock
        ////////////////////    PS2     ////////////////////////////
        PS2_DAT,                        //  PS2 Data
        PS2_CLK,                        //  PS2 Clock
        ////////////////////    VGA     ////////////////////////////
        VGA_CLK,                        //  VGA Clock
        VGA_HS,                         //  VGA H_SYNC
        VGA_VS,                         //  VGA V_SYNC
        VGA_BLANK,                      //  VGA BLANK
        VGA_SYNC,                       //  VGA SYNC
        VGA_R,                          //  VGA Red[9:0]
        VGA_G,                          //  VGA Green[9:0]
        VGA_B,                          //  VGA Blue[9:0]
        ////////////    Ethernet Interface  ///////////////////////
        ENET_DATA,                      //  DM9000A DATA bus 16Bits
        ENET_CMD,                       //  DM9000A Command/Data Select, 0 = Command, 1 = Data
        ENET_CS_N,                      //  DM9000A Chip Select
        ENET_WR_N,                      //  DM9000A Write
        ENET_RD_N,                      //  DM9000A Read
        ENET_RST_N,                     //  DM9000A Reset
        ENET_INT,                       //  DM9000A Interrupt
        ENET_CLK,                       //  DM9000A Clock 25 MHz
        ////////////////    Audio CODEC     ////////////////////////
        AUD_ADCLRCK,                    //  Audio CODEC ADC LR Clock
        AUD_ADCDAT,                     //  Audio CODEC ADC Data
        AUD_DACLRCK,                    //  Audio CODEC DAC LR Clock
        AUD_DACDAT,                     //  Audio CODEC DAC Data
        AUD_BCLK,                       //  Audio CODEC Bit-Stream Clock
        AUD_XCK,                        //  Audio CODEC Chip Clock
        ////////////////    TV Decoder      ////////////////////////
        TD_DATA,                        //  TV Decoder Data bus 8 bits
        TD_HS,                          //  TV Decoder H_SYNC
        TD_VS,                          //  TV Decoder V_SYNC
        TD_RESET,                       //  TV Decoder Reset
        ////////////////////    GPIO    ////////////////////////////
        GPIO_0,                         //  GPIO Connection 0
        GPIO_1                          //  GPIO Connection 1
    );

//////////////////////    Clock Input     ////////////////////////
input           CLOCK_27;               //  27 MHz
input           CLOCK_50;               //  50 MHz
input           EXT_CLOCK;              //  External Clock
//////////////////////    Push Button     ////////////////////////
input   [3:0]   KEY;                    //  Pushbutton[3:0]
//////////////////////    DPDT Switch     ////////////////////////
input   [17:0]  SW;                     //  Toggle Switch[17:0]
//////////////////////    7-SEG Dispaly   ////////////////////////
output  [6:0]   HEX0;                   //  Seven Segment Digit 0
output  [6:0]   HEX1;                   //  Seven Segment Digit 1
output  [6:0]   HEX2;                   //  Seven Segment Digit 2
output  [6:0]   HEX3;                   //  Seven Segment Digit 3
output  [6:0]   HEX4;                   //  Seven Segment Digit 4
output  [6:0]   HEX5;                   //  Seven Segment Digit 5
output  [6:0]   HEX6;                   //  Seven Segment Digit 6
output  [6:0]   HEX7;                   //  Seven Segment Digit 7
////////////////////////////    LED     ////////////////////////////
output  [8:0]   LEDG;                   //  LED Green[8:0]
output  [17:0]  LEDR;                   //  LED Red[17:0]
////////////////////////////    UART    ////////////////////////////
output          UART_TXD;               //  UART Transmitter
input           UART_RXD;               //  UART Receiver
/*////////////////////////////  IRDA    ////////////////////////////
output          IRDA_TXD;               //  IRDA Transmitter
input           IRDA_RXD;               //  IRDA Receiver*/
```

```
//////////////////////     SDRAM Interface ////////////////////////
inout   [15:0]  DRAM_DQ;                //  SDRAM Data bus 16 Bits
output  [11:0]  DRAM_ADDR;              //  SDRAM Address bus 12 Bits
output          DRAM_LDQM;              //  SDRAM Low-byte Data Mask
output          DRAM_UDQM;              //  SDRAM High-byte Data Mask
output          DRAM_WE_N;              //  SDRAM Write Enable
output          DRAM_CAS_N;             //  SDRAM Column Address Strobe
output          DRAM_RAS_N;             //  SDRAM Row Address Strobe
output          DRAM_CS_N;              //  SDRAM Chip Select
output          DRAM_BA_0;              //  SDRAM Bank Address 0
output          DRAM_BA_1;              //  SDRAM Bank Address 0
output          DRAM_CLK;               //  SDRAM Clock
output          DRAM_CKE;               //  SDRAM Clock Enable
////////////////////////    Flash Interface ////////////////////////
inout   [7:0]   FL_DQ;                  //  FLASH Data bus 8 Bits
output  [21:0]  FL_ADDR;                //  FLASH Address bus 22 Bits
output          FL_WE_N;                //  FLASH Write Enable
output          FL_RST_N;               //  FLASH Reset
output          FL_OE_N;                //  FLASH Output Enable
output          FL_CE_N;                //  FLASH Chip Enable
//////////////////////    SRAM Interface  ////////////////////////
inout   [15:0]  SRAM_DQ;                //  SRAM Data bus 16 Bits
output  [17:0]  SRAM_ADDR;              //  SRAM Address bus 18 Bits
output          SRAM_UB_N;              //  SRAM High-byte Data Mask
output          SRAM_LB_N;              //  SRAM Low-byte Data Mask
output          SRAM_WE_N;              //  SRAM Write Enable
output          SRAM_CE_N;              //  SRAM Chip Enable
output          SRAM_OE_N;              //  SRAM Output Enable
////////////////////    ISP1362 Interface   ////////////////////////
inout   [15:0]  OTG_DATA;               //  ISP1362 Data bus 16 Bits
output  [1:0]   OTG_ADDR;               //  ISP1362 Address 2 Bits
output          OTG_CS_N;               //  ISP1362 Chip Select
output          OTG_RD_N;               //  ISP1362 Write
output          OTG_WR_N;               //  ISP1362 Read
output          OTG_RST_N;              //  ISP1362 Reset
output          OTG_FSPEED;             //  USB Full Speed, 0 = Enable, Z = Disable
output          OTG_LSPEED;             //  USB Low Speed,  0 = Enable, Z = Disable
input           OTG_INT0;               //  ISP1362 Interrupt 0
input           OTG_INT1;               //  ISP1362 Interrupt 1
input           OTG_DREQ0;              //  ISP1362 DMA Request 0
input           OTG_DREQ1;              //  ISP1362 DMA Request 1
output          OTG_DACK0_N;            //  ISP1362 DMA Acknowledge 0
output          OTG_DACK1_N;            //  ISP1362 DMA Acknowledge 1
////////////////////    LCD Module 16X2 ////////////////////////////
inout   [7:0]   LCD_DATA;               //  LCD Data bus 8 bits
output          LCD_ON;                 //  LCD Power ON/OFF
output          LCD_BLON;               //  LCD Back Light ON/OFF
output          LCD_RW;                 //  LCD Read/Write Select, 0 = Write, 1 = Read
output          LCD_EN;                 //  LCD Enable
output          LCD_RS;                 //  LCD Command/Data Select, 0 = Command, 1 = Data
////////////////////    SD Card Interface   ////////////////////////
inout           SD_DAT;                 //  SD Card Data
inout           SD_DAT3;                //  SD Card Data 3
inout           SD_CMD;                 //  SD Card Command Signal
output          SD_CLK;                 //  SD Card Clock
////////////////////////    I2C     ////////////////////////////////
inout           I2C_SDAT;               //  I2C Data
output          I2C_SCLK;               //  I2C Clock
////////////////////////    PS2     ////////////////////////////////
input           PS2_DAT;                //  PS2 Data
input           PS2_CLK;                //  PS2 Clock
////////////////////    USB JTAG link   ////////////////////////////
input           TDI;                    // CPLD -> FPGA (data in)
input           TCK;                    // CPLD -> FPGA (clk)
input           TCS;                    // CPLD -> FPGA (CS)
output          TDO;                    // FPGA -> CPLD (data out)
////////////////////////    VGA         ////////////////////////////
output          VGA_CLK;                //  VGA Clock
output          VGA_HS;                 //  VGA H_SYNC
output          VGA_VS;                 //  VGA V_SYNC
output          VGA_BLANK;              //  VGA BLANK
output          VGA_SYNC;               //  VGA SYNC
output  [9:0]   VGA_R;                  //  VGA Red[9:0]
output  [9:0]   VGA_G;                  //  VGA Green[9:0]
output  [9:0]   VGA_B;                  //  VGA Blue[9:0]
////////////////    Ethernet Interface  ////////////////////////////
inout   [15:0]  ENET_DATA;              //  DM9000A DATA bus 16Bits
output          ENET_CMD;               //  DM9000A Command/Data Select, 0 = Command, 1 = Data
output          ENET_CS_N;              //  DM9000A Chip Select
output          ENET_WR_N;              //  DM9000A Write
output          ENET_RD_N;              //  DM9000A Read
output          ENET_RST_N;             //  DM9000A Reset
```

```
input           ENET_INT;               //  DM9000A Interrupt
output          ENET_CLK;               //  DM9000A Clock 25 MHz
//////////////////    Audio CODEC     ////////////////////////////
inout           AUD_ADCLRCK;            //  Audio CODEC ADC LR Clock
input           AUD_ADCDAT;             //  Audio CODEC ADC Data
inout           AUD_DACLRCK;            //  Audio CODEC DAC LR Clock
output          AUD_DACDAT;             //  Audio CODEC DAC Data
inout           AUD_BCLK;               //  Audio CODEC Bit-Stream Clock
output          AUD_XCK;                //  Audio CODEC Chip Clock
//////////////////    TV Devoder      ////////////////////////////
input   [7:0]   TD_DATA;                //  TV Decoder Data bus 8 bits
input           TD_HS;                  //  TV Decoder H_SYNC
input           TD_VS;                  //  TV Decoder V_SYNC
output          TD_RESET;               //  TV Decoder Reset
//////////////////////    GPIO    ////////////////////////////////
inout   [35:0]  GPIO_0;                 //  GPIO Connection 0
inout   [35:0]  GPIO_1;                 //  GPIO Connection 1

//  All inout port turn to tri-state
assign  DRAM_DQ     =   16'hzzzz;
assign  FL_DQ       =   8'hzz;
//assign    SRAM_DQ     =   16'hzzzz;
assign  OTG_DATA    =   16'hzzzz;
assign  LCD_DATA    =   8'hzz;
assign  SD_DAT      =   1'bz;
assign  I2C_SDAT    =   1'bz;
assign  ENET_DATA   =   16'hzzzz;
//assign    AUD_ADCLRCK =   1'bz;
//assign    AUD_DACLRCK =   1'bz;
//assign    AUD_BCLK    =   1'bz;
//assign    GPIO_0      =   36'hzzzzzzzzz;
//assign    GPIO_1      =   36'hzzzzzzzzz;

// Master Control Signals
wire            wMRST = ~KEY[0];
wire            wMCLK;
reg             rMCLK = 0;
reg [3:0]       rMCLK_cnt = 0;

// Try to seperate the CPU clock from the VGA_PLL
always@(posedge CLOCK_27) begin
    if(rMCLK_cnt < 9)
        begin
            rMCLK_cnt <= rMCLK_cnt + 1;
        end
    else
        begin
            rMCLK_cnt <= 0;
            rMCLK <= ~rMCLK;
        end
end

// This will allow switching between single stepped operation
// and full speed operation.
wire            wCLK_SWITCH = (SW[0]) ? rMCLK : ~KEY[1];

// GPU Interface
wire            wGPUComplete;
wire            wGPUValid;
wire    [15:0]  wGPUInstruction;

// PS2 Input Buffer
wire            wPS2BufferEmpty;
wire    [7:0]   wPS2ASCIIOut;
wire            wPS2Complete;
wire            wPS2Valid;


// Boot Loader
wire    [7:0]   wASCII_TX;
wire            wValid_TX;
wire            wComplete_TX;

wire    [7:0]   wASCII_RX;
wire            wBufferEmpty;
wire            wValid_RX;
wire            wComplete_RX;

//assign LEDG[0] = wBufferEmpty;

//assign LEDR = wMem1Addr;
```

```
wire    [31:0]  wPC;
wire    [31:0]  wIns;
wire            wBootLoad;
wire            wWE_N;

// If bootloading drive the instruction memory with boot loader
wire    [31:0]  wMem1Write = (wBootLoad) ? wWE_N : ~wIWriteEnable1;
wire    [31:0]  wMem1Addr = (wBootLoad) ? wPC : wIWriteAddr1;
wire    [31:0]  wMem1Data = (wBootLoad) ? wIns : wIWriteData1;

wire    [31:0]  wMem2Addr = PC2;
wire    [31:0]  wMem3Addr = PC3;
wire    [31:0]  wMem4Addr = PC4;
wire    [31:0]  wMem5Addr = PC5;
wire    [31:0]  wMem6Addr = PC6;

wire            wBootCLK;

//ndma_pll ndmapll(CLOCK_50, wBootCLK);

// CPU 1 BootLoader
RS232InputBuffer rs232
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // RS-232 Connections
   `UART_INTERFACE_DE2,
    // Buffer Interface
    // Transmit
    .iASCII_TX(wASCII_TX),
    .iValid_TX(wValid_TX),
    .oComplete_TX(wComplete_TX),
    // Receive
    .iValid(wValid_RX),
    .oBufferEmpty(wBufferEmpty),
    .oASCIIOut(wASCII_RX),
    .oComplete(wComplete_RX)
);

RS232BootLoader bootload
(
    .iCLK(CLOCK_50), .iMRST(wMRST),
    // Transmit
    .oASCII(wASCII_TX),             // Transmit ASCII Value
    .oValid_TX(wValid_TX),          // Valid Transmit
    .iComplete_TX(wComplete_TX),    // Transmit Complete
    // Recieve
    .iASCII(wASCII_RX),             // Receive ASCII Value
    .iBufferEmpty(wBufferEmpty),    // Buffer Empty Value
    .oValid_RX(wValid_RX),          // Valid Receive
    .iComplete_RX(wComplete_RX),    // Receive Complete
    //.oState(LEDR),
    .oBootLoad(wBootLoad),
    .oPC(wPC),
    .oIns(wIns),
    .oWE_N(wWE_N)
);

// Debugging
// ****************************************************************

wire    [31:0]  HexInsOut = (SW[17]) ? Imem4 : Imem1;

wire    [31:0]  HexOut = (SW[16]) ? sBO5 : HexInsOut;

wire    [31:0]  HexCPU2SouthOut = sBO2;

wire    [31:0]  HexCPU2WestOut = wBO2;

HexDecode hex0(HexOut[3:0], HEX0);
HexDecode hex1(HexOut[7:4], HEX1);
HexDecode hex2(HexOut[11:8], HEX2);
HexDecode hex3(HexOut[15:12], HEX3);
HexDecode hex4(HexOut[19:16], HEX4);
HexDecode hex5(HexOut[23:20], HEX5);
HexDecode hex6(HexOut[27:24], HEX6);
HexDecode hex7(HexOut[31:28], HEX7);

assign LEDR = (SW[17]) ? PC4 : PC1;

//assign LEDG = (SW[17]) ? wSelfID2 : wSelfID1;
assign LEDG[7] = wDxFlag1;
assign LEDG[6] = wRxFlag4;
```

```
assign LEDG[5] = wInsBufCnt4[1];
assign LEDG[4] = wInsBufCnt4[0];

assign LEDG[3] = wSelfID4[2];
assign LEDG[2] = wSelfID4[1];
assign LEDG[1] = wSelfID4[0];

// assign LEDG[1] = wSelfID2[1];
//assign LEDG[0] = wSelfID2[0];

// ********************************************************************


// CPU 1
// ********************************************************************

wire    [31:0]  PC1, Imem1, dAddr1, Din1, Dout1;
wire            dWrite1;
// CPU 1 Network
wire    [31:0]  nBI1, sBI1, wBI1, eBI1;
wire    [31:0]  nBO1, sBO1, wBO1, eBO1;
// IO
wire    [31:0]  wPortA1, wPortB1, wPortC1, wPortD1;
wire    [31:0]  wPortE1, wPortF1, wPortG1, wPortH1;

// Network Memory
wire    [31:0]  wIWriteAddr1;
wire    [31:0]  wIWriteData1;
wire    [31:0]  wIWriteEnable1;

// Debuggins
wire    [7:0]   wSelfID1;
wire            wDxFlag1;

// Instruction Memory CPU 1
M4KMEM_DUALPORT m4kmem_ins1(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(wMem1Write),
    .iWriteAddr(wMem1Addr),
    .iWriteData(wMem1Data),
    // Read Side
    .iReadAddr(PC1),
    .oReadData(Imem1)
);
// m4kmem parameters
defparam m4kmem_ins1.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins1.pRAM_UNIT_COUNT = 1024,
         m4kmem_ins1.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins1.pINIT_FILE = "memory1.mif";
// end m4kmem parameters



// Data Memory CPU 1
// negative edge triggered ?
M4KMEM m4kmem_data1(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite1), .iAddr(dAddr1),
    .iWriteData(Din1), .oReadData(Dout1)
);
// m4kmem parameters
defparam m4kmem_data1.pRAM_ADDR_WIDTH = 10,
         m4kmem_data1.pRAM_UNIT_COUNT = 1024,
         m4kmem_data1.pRAM_UNIT_WIDTH = 32,
         m4kmem_data1.pINIT_FILE = "data1.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu1
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    .oIMemAddr  (PC1),
    .iMemIn     (Imem1),
    // Network Memory
    .oIWAddr    (wIWriteAddr1),
    .oIWrite    (wIWriteEnable1),
    .oIData     (wIWriteData1),
    // Data Memory
    .dMemIn     (Dout1),
```

```
    .dAddr      (dAddr1),
    .dWrite     (dWrite1),
    .WD         (Din1),
    // Network Side
    .nBusIn     (nBI1), .nBusOut    (nBO1),
    .sBusIn     (nBO6), .sBusOut    (sBO1),
    .eBusIn     (wBO2), .eBusOut    (eBO1),
    .wBusIn     (nBI1), .wBusOut    (wBO1),
    // IO Side
    .iPortA(wPortA1),   // Input From GPU
    .oPortE(wPortE1),   // Output to GPU
    .iPortB(wPortB1),   // Input from PS2 Buffer
    .oPortF(wPortF1),   // Output to PS2 buffer
    // dubbings
    .oSelfID(wSelfID1),
    .oDxFlag(wDxFlag1)
);



// ********************************************************************

// CPU 2
// ********************************************************************
// Note that CPU 2 is initialized with the following code:
// sid 2
// nop
// nop
// j 1

wire    [31:0]  PC2, Imem2, dAddr2, Din2, Dout2;
wire            dWrite2;
// CPU 2 Network
wire    [31:0]  nBI2, sBI2, wBI2, eBI2;
wire    [31:0]  nBO2, sBO2, wBO2, eBO2;
// IO
wire    [31:0]  wPortA2, wPortB2, wPortC2, wPortD2;
wire    [31:0]  wPortE2, wPortF2, wPortG2, wPortH2;
// Network Connections
wire    [31:0]  wIWriteAddr2;
wire    [31:0]  wIWriteData2;
wire    [31:0]  wIWriteEnable2;

wire    [7:0]   wSelfID2;
wire            wRxFlag2;
wire    [1:0]   wInsBufCnt2;

// Instruction Memory CPU 2
M4KMEM_DUALPORT m4kmem_ins2(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable2),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr2),
    .iWriteData(wIWriteData2),
    // Read Side
    .iReadAddr(wMem2Addr),
    .oReadData(Imem2)
);
// m4kmem parameters
defparam m4kmem_ins2.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins2.pRAM_UNIT_COUNT = 1024,
         m4kmem_ins2.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins2.pINIT_FILE = "memory2.mif";
// end m4kmem parameters


// Data Memory CPU 2
M4KMEM m4kmem_data2(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite2), .iAddr(dAddr2),
    .iWriteData(Din2), .oReadData(Dout2)
);
// m4kmem parameters
defparam m4kmem_data2.pRAM_ADDR_WIDTH = 10,
         m4kmem_data2.pRAM_UNIT_COUNT = 1024,
         m4kmem_data2.pRAM_UNIT_WIDTH = 32,
         m4kmem_data2.pINIT_FILE = "data2.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu2
(
```

```
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    // Instruction Memory
    .oIMemAddr  (PC2),
    .iMemIn     (Imem2),
    // Network Memory
    .oIWAddr    (wIWriteAddr2),
    .oIWrite    (wIWriteEnable2),
    .oIData     (wIWriteData2),
    // Data Memory
    .dMemIn     (Dout2),
    .dAddr      (dAddr2),
    .dWrite     (dWrite2),
    .WD         (Din2),
    // Network Side
    .nBusIn     (nBI2), .nBusOut    (nBO2),
    .sBusIn     (nBO3), .sBusOut    (sBO2),
    .eBusIn     (eBO5), .eBusOut    (eBO2),
    .wBusIn     (eBO1), .wBusOut    (wBO2),
    // IO Side
    /*
    .iPortA(wPortA2),   // Input From GPU
    .oPortE(wPortE2),   // Output to GPU
    .iPortB(wPortB2),   // Input from PS2 Buffer
    .oPortF(wPortF2),   // Output to PS2 buffer
    */
    // SELF ID
    .oSelfID(wSelfID2),
    .oRxFlag(wRxFlag2),
    .oInsBufCnt(wInsBufCnt2)
);
// ********************************************************************

// CPU 3
// ********************************************************************
// Note that CPU 3 is initialized with the following code:
// sid 3
// nop
// nop
// j 1

wire    [31:0]  PC3, Imem3, dAddr3, Din3, Dout3;
wire            dWrite3;
// CPU 2 Network
wire    [31:0]  nBI3, sBI3, wBI3, eBI3;
wire    [31:0]  nBO3, sBO3, wBO3, eBO3;
// IO
wire    [31:0]  wPortA3, wPortB3, wPortC3, wPortD3;
wire    [31:0]  wPortE3, wPortF3, wPortG3, wPortH3;
// Network Connections
wire    [31:0]  wIWriteAddr3;
wire    [31:0]  wIWriteData3;
wire    [31:0]  wIWriteEnable3;

wire    [7:0]   wSelfID3;
wire            wRxFlag3;
wire    [1:0]   wInsBufCnt3;

// Instruction Memory CPU 3
M4KMEM_DUALPORT m4kmem_ins3(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable3),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr3),
    .iWriteData(wIWriteData3),
    // Read Side
    .iReadAddr(wMem3Addr),
    .oReadData(Imem3)
);
// m4kmem parameters
defparam m4kmem_ins3.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins3.pRAM_UNIT_COUNT = 1024,
         m4kmem_ins3.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins3.pINIT_FILE = "memory3.mif";
// end m4kmem parameters


// Data Memory CPU 3
M4KMEM m4kmem_data3(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite3), .iAddr(dAddr3),
    .iWriteData(Din3), .oReadData(Dout3)
```

```
);
// m4kmem parameters
defparam m4kmem_data3.pRAM_ADDR_WIDTH = 10,
         m4kmem_data3.pRAM_UNIT_COUNT = 1024,
         m4kmem_data3.pRAM_UNIT_WIDTH = 32,
         m4kmem_data3.pINIT_FILE = "data3.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu3
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    // Instruction Memory
    .oIMemAddr  (PC3),
    .iMemIn     (Imem3),
    // Network Memory
    .oIWAddr    (wIWriteAddr3),
    .oIWrite    (wIWriteEnable3),
    .oIData     (wIWriteData3),
    // Data Memory
    .dMemIn     (Dout3),
    .dAddr      (dAddr3),
    .dWrite     (dWrite3),
    .WD         (Din3),
    // Network Side
    .nBusIn     (sBO2), .nBusOut    (nBO3),
    .sBusIn     (sBI3), .sBusOut    (sBO3),
    .eBusIn     (wBO4), .eBusOut    (eBO3),
    .wBusIn     (eBO6), .wBusOut    (wBO3),
    // IO Side
    /*
    .iPortA(wPortA3),   // Input From GPU
    .oPortE(wPortE3),   // Output to GPU
    .iPortB(wPortB3),   // Input from PS2 Buffer
    .oPortF(wPortF3),   // Output to PS2 buffer
    */
    // SELF ID
    .oSelfID(wSelfID3),
    .oRxFlag(wRxFlag3),
    .oInsBufCnt(wInsBufCnt3)
);
// ********************************************************************

// CPU 4
// ********************************************************************
// Note that CPU 4 is initialized with the following code:
// sid 4
// nop
// nop
// j 1

wire    [31:0]  PC4, Imem4, dAddr4, Din4, Dout4;
wire            dWrite4;
// CPU 2 Network
wire    [31:0]  nBI4, sBI4, wBI4, eBI4;
wire    [31:0]  nBO4, sBO4, wBO4, eBO4;
// IO
wire    [31:0]  wPortA4, wPortB4, wPortC4, wPortD4;
wire    [31:0]  wPortE4, wPortF4, wPortG4, wPortH4;
// Network Connections
wire    [31:0]  wIWriteAddr4;
wire    [31:0]  wIWriteData4;
wire    [31:0]  wIWriteEnable4;

wire    [7:0]   wSelfID4;
wire            wRxFlag4;
wire    [1:0]   wInsBufCnt4;

// Instruction Memory CPU 4
M4KMEM_DUALPORT m4kmem_ins4(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable4),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr4),
    .iWriteData(wIWriteData4),
    // Read Side
    .iReadAddr(wMem4Addr),
    .oReadData(Imem4)
);
// m4kmem parameters
defparam m4kmem_ins4.pRAM_ADDR_WIDTH = 10,
```

```
        m4kmem_ins4.pRAM_UNIT_COUNT = 1024,
        m4kmem_ins4.pRAM_UNIT_WIDTH = 32,
        m4kmem_ins4.pINIT_FILE = "memory4.mif";
// end m4kmem parameters


// Data Memory CPU 4
M4KMEM m4kmem_data4(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite4), .iAddr(dAddr4),
    .iWriteData(Din4), .oReadData(Dout4)
);
// m4kmem parameters
defparam m4kmem_data4.pRAM_ADDR_WIDTH = 10,
        m4kmem_data4.pRAM_UNIT_COUNT = 1024,
        m4kmem_data4.pRAM_UNIT_WIDTH = 32,
        m4kmem_data4.pINIT_FILE = "data4.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu4
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    // Instruction Memory
    .oIMemAddr  (PC4),
    .iMemIn     (Imem4),
    // Network Memory
    .oIWAddr    (wIWriteAddr4),
    .oIWrite    (wIWriteEnable4),
    .oIData     (wIWriteData4),
    // Data Memory
    .dMemIn     (Dout4),
    .dAddr      (dAddr4),
    .dWrite     (dWrite4),
    .WD         (Din4),
    // Network Side
    .nBusIn     (sBO5), .nBusOut    (nBO4),
    .sBusIn     (sBI4), .sBusOut    (sBO4),
    .eBusIn     (eBI4), .eBusOut    (eBO4),
    .wBusIn     (eBO3), .wBusOut    (wBO4),
    // IO Side
    /*
    .iPortA(wPortA4),   // Input From GPU
    .oPortE(wPortE4),   // Output to GPU
    .iPortB(wPortB4),   // Input from PS2 Buffer
    .oPortF(wPortF4),   // Output to PS2 buffer
    */
    // SELF ID
    .oSelfID(wSelfID4),
    .oRxFlag(wRxFlag4),
    .oInsBufCnt(wInsBufCnt4)
);
// ********************************************************************

// CPU 5
// ********************************************************************
// Note that CPU 5 is initialized with the following code:
// sid 5
// nop
// nop
// j 1

wire    [31:0]  PC5, Imem5, dAddr5, Din5, Dout5;
wire            dWrite5;
// CPU 2 Network
wire    [31:0]  nBI5, sBI5, wBI5, eBI5;
wire    [31:0]  nBO5, sBO5, wBO5, eBO5;
// IO
wire    [31:0]  wPortA5, wPortB5, wPortC5, wPortD5;
wire    [31:0]  wPortE5, wPortF5, wPortG5, wPortH5;
// Network Connections
wire    [31:0]  wIWriteAddr5;
wire    [31:0]  wIWriteData5;
wire    [31:0]  wIWriteEnable5;

wire    [7:0]   wSelfID5;
wire            wRxFlag5;
wire    [1:0]   wInsBufCnt5;

// Instruction Memory CPU 5
M4KMEM_DUALPORT m4kmem_ins5(
```

```
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable5),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr5),
    .iWriteData(wIWriteData5),
    // Read Side
    .iReadAddr(wMem5Addr),
    .oReadData(Imem5)
);
// m4kmem parameters
defparam m4kmem_ins5.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins5.pRAM_UNIT_COUNT = 512,
         m4kmem_ins5.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins5.pINIT_FILE = "memory5.mif";
// end m4kmem parameters


// Data Memory CPU 5
M4KMEM m4kmem_data5(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite5), .iAddr(dAddr5),
    .iWriteData(Din5), .oReadData(Dout5)
);
// m4kmem parameters
defparam m4kmem_data5.pRAM_ADDR_WIDTH = 10,
         m4kmem_data5.pRAM_UNIT_COUNT = 512,
         m4kmem_data5.pRAM_UNIT_WIDTH = 32,
         m4kmem_data5.pINIT_FILE = "data5.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu5
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    // Instruction Memory
    .oIMemAddr  (PC5),
    .iMemIn     (Imem5),
    // Network Memory
    .oIWAddr    (wIWriteAddr5),
    .oIWrite    (wIWriteEnable5),
    .oIData     (wIWriteData5),
    // Data Memory
    .dMemIn     (Dout5),
    .dAddr      (dAddr5),
    .dWrite     (dWrite5),
    .WD         (Din5),
    // Network Side
    .nBusIn     (nBI5), .nBusOut    (nBO5),
    .sBusIn     (nBO4), .sBusOut    (sBO5),
    .eBusIn     (eBI5), .eBusOut    (eBO5),
    .wBusIn     (eBO2), .wBusOut    (wBO5),
    // IO Side
    /*
    .iPortA(wPortA5),   // Input From GPU
    .oPortE(wPortE5),   // Output to GPU
    .iPortB(wPortB5),   // Input from PS2 Buffer
    .oPortF(wPortF5),   // Output to PS2 buffer
    */
    // SELF ID
    .oSelfID(wSelfID5),
    .oRxFlag(wRxFlag5),
    .oInsBufCnt(wInsBufCnt5)
);
// ********************************************************************

// CPU 6
// ********************************************************************
// Note that CPU 6 is initialized with the following code:
// sid 6
// nop
// nop
// j 1

wire    [31:0]  PC6, Imem6, dAddr6, Din6, Dout6;
wire            dWrite6;
// CPU 2 Network
wire    [31:0]  nBI6, sBI6, wBI6, eBI6;
wire    [31:0]  nBO6, sBO6, wBO6, eBO6;
// IO
wire    [31:0]  wPortA6, wPortB6, wPortC6, wPortD6;
wire    [31:0]  wPortE6, wPortF6, wPortG6, wPortH6;
```

```
// Network Connections
wire    [31:0]  wIWriteAddr6;
wire    [31:0]  wIWriteData6;
wire    [31:0]  wIWriteEnable6;

wire    [7:0]   wSelfID6;
wire            wRxFlag6;
wire    [1:0]   wInsBufCnt6;

// Instruction Memory CPU 6
M4KMEM_DUALPORT m4kmem_ins6(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    // Write Side
    .iWE_N(~wIWriteEnable6),            // Write enable MUST be negated
    .iWriteAddr(wIWriteAddr6),
    .iWriteData(wIWriteData6),
    // Read Side
    .iReadAddr(wMem6Addr),
    .oReadData(Imem6)
);
// m4kmem parameters
defparam m4kmem_ins6.pRAM_ADDR_WIDTH = 10,
         m4kmem_ins6.pRAM_UNIT_COUNT = 512,
         m4kmem_ins6.pRAM_UNIT_WIDTH = 32,
         m4kmem_ins6.pINIT_FILE = "memory6.mif";
// end m4kmem parameters


// Data Memory CPU 6
M4KMEM m4kmem_data6(
    .iCLK(CLOCK_50), .iMRST(~wMRST),
    .iWE_N(~dWrite6), .iAddr(dAddr6),
    .iWriteData(Din6), .oReadData(Dout6)
);
// m4kmem parameters
defparam m4kmem_data6.pRAM_ADDR_WIDTH = 10,
         m4kmem_data6.pRAM_UNIT_COUNT = 512,
         m4kmem_data6.pRAM_UNIT_WIDTH = 32,
         m4kmem_data6.pINIT_FILE = "data6.mif";
// end m4kmem parameters

// NDMA CPU
CPU cpu6
(
    .CLK        (wCLK_SWITCH),
    .MRST       (wMRST | wBootLoad),        // wBootLoad resets CPU (PC -> 0)
    // Instruction Memory
    .oIMemAddr  (PC6),
    .iMemIn     (Imem6),
    // Network Memory
    .oIWAddr    (wIWriteAddr6),
    .oIWrite    (wIWriteEnable6),
    .oIData     (wIWriteData6),
    // Data Memory
    .dMemIn     (Dout6),
    .dAddr      (dAddr6),
    .dWrite     (dWrite6),
    .WD         (Din6),
    // Network Side
    .nBusIn     (sBO1), .nBusOut    (nBO6),
    .sBusIn     (sBI6), .sBusOut    (sBO6),
    .eBusIn     (wBO3), .eBusOut    (eBO6),
    .wBusIn     (wBI6), .wBusOut    (wBO6),
    // IO Side
    /*
    .iPortA(wPortA5),   // Input From GPU
    .oPortE(wPortE5),   // Output to GPU
    .iPortB(wPortB5),   // Input from PS2 Buffer
    .oPortF(wPortF5),   // Output to PS2 buffer
    */
    // SELF ID
    .oSelfID(wSelfID6),
    .oRxFlag(wRxFlag6),
    .oInsBufCnt(wInsBufCnt6)
);
// ********************************************************************


// ********************************************************************
// SYSTEM RESOURCES
// ********************************************************************
```

```
// PS2 Input Buffer Belongs to CPU1
// ********************************************
assign wPortB1 = {wPS2BufferEmpty, wPS2Complete, wPS2ASCIIOut};
assign wPS2Valid = wPortF1[0];
//assign LEDG[6] = wPS2Valid;
//assign LEDG[5] = wPS2BufferEmpty;

PS2InputBuffer ps2buf
(
    .iCLK(CLOCK_50), .iMRST(wMRST | wBootLoad),
    `PS2_INTERFACE_DE2,
    .iValid(wPS2Valid),
    .oBufferEmpty(wPS2BufferEmpty),
    .oASCIIOut(wPS2ASCIIOut),
    .oComplete(wPS2Complete)
);
// ********************************************

// GPU belongs to CPU1
// ********************************************
assign wPortA1 = {15'd0, wGPUComplete};
assign wGPUInstruction = wPortE1[15:0];
assign wGPUValid = wPortE1[16];

GPU gpu
(
    `VGA_CLOCKS_DE2,
    // User Data
    .iReset(wMRST | wBootLoad),
    .iValid(wGPUValid),
    .iIns(wGPUInstruction),
    .oComplete(wGPUComplete),
    .oVGA_CTRL_CLK(wVGA_CTRL_CLK),
    //.oAUDCtrlClk(wAUDCtrlClk),
    //.oDLY_RST(wDLY_RST),
    `VGA_INTERFACE_DE2,
    `VGA_TD_RESET_DE2,
    // SRAM Interface
    `SRAM_INTERFACE_DE2
);
// ********************************************

endmodule
```

```
/************************************************
* Network Driven Microprocessor Architecture Reference
************************************************/

`define TICK        #1

`define     true    1'b1
`define     false   1'b0

`define     dc      1'bx
`define     dc5     5'bxxxxx
`define     dc6     6'bxxxxxx

// Instruction fields of 32 bit instructions

`define     op          31:26       // 6 bit operation field
`define     function    5:0         // 6 bit function field
`define     rs          25:21       // 5 bit source register field
`define     rt          20:16       // 5 bit target register field
`define     rd          15:11       // 5 bit destination register field
`define     immediate   15:0        // 16 bit signed immediate
`define     target      25:0        // 26 bit indext shifted left (target address for jumps)
`define     sa          10:6        // 5 bit shift amount

// Instruction types

// I type
//  ___________________________________________
// | op * 6 | rs * 5 | rt * 5  | immediate * 16 |
// template
// opopoprssssrttttimmediateimmedia

// J type
//  ____________________
// | op * 6 | target * 26|
// template
// opopoptargettargettargettargetta

// R type
//  ___________________________________________________
// | op * 6 | rs * 5 | rt * 5 | rd * 5 | sa * 5 | function * 6 |
// template:
// opopoprssssrttttrddddsaaaafuncti

// Register names

`define r0      5'b00000
`define r1      5'b00001
`define r2      5'b00010
`define r3      5'b00011
`define r4      5'b00100
`define r5      5'b00101
`define r6      5'b00110
`define r7      5'b00111
`define r8      5'b01000
`define r9      5'b01001
`define r10     5'b01010
`define r11     5'b01011
`define r12     5'b01100
`define r13     5'b01101
`define r14     5'b01110
`define r15     5'b01111
`define r16     5'b10000
`define r17     5'b10001
`define r18     5'b10010
`define r19     5'b10011
`define r20     5'b10100
`define r21     5'b10101
`define r22     5'b10110
`define r23     5'b10111
`define r24     5'b11000
`define r25     5'b11001
`define r26     5'b11010
`define r27     5'b11011
`define r28     5'b11100
`define r29     5'b11101
`define r30     5'b11110
`define r31     5'b11111

// Symbollic Register names (assembler / compiler)

`define zero    5'b00000
```

```
`define at      5'b00001
`define v0      5'b00010
`define v1      5'b00011
`define a0      5'b00100
`define a1      5'b00101
`define a2      5'b00110
`define a3      5'b00111
`define t0      5'b01000
`define t1      5'b01001
`define t2      5'b01010
`define t3      5'b01011
`define t4      5'b01100
`define t5      5'b01101
`define t6      5'b01110
`define t7      5'b01111
`define s0      5'b10000
`define s1      5'b10001
`define s2      5'b10010
`define s3      5'b10011
`define s4      5'b10100
`define s5      5'b10101
`define s6      5'b10110
`define s7      5'b10111
`define t8      5'b11000
`define t9      5'b11001
`define k0      5'b11010
`define k1      5'b11011
`define gp      5'b11100
`define sp      5'b11101
`define s8      5'b11110
`define ra      5'b11111

// Opcode Assignments for `op Operations

`define SPECIAL     6'b000000
`define REGIMM      6'b000001

// Jump! and Branch!
`define J           6'b000010
`define JAL         6'b000011
`define BEQ         6'b000100
`define BNE         6'b000101
`define BLEZ        6'b000110
`define BGTZ        6'b000111

`define BLTZ        6'b010100
`define BGEZ        6'b010101

`define ADDI        6'b001000
`define ADDIU       6'b001001
`define SLTI        6'b001010
`define SLTIU       6'b001011
`define ANDI        6'b001100
`define ORI         6'b001101
`define XORI        6'b001110
`define LUI         6'b001111

// OUTI instrction
`define OUTI        6'b010110

//`define   COP0        6'b010000   // No Coprocessor
//`define   COP1        6'b010001
//`define   COP2        6'b010010
//`define   COP3        6'b010011
//`define   BEQL        6'b010100   // No Delay slots in this architecture
//`define   BNEL        6'b010101
//`define   BLEZL       6'b010110
`define BGTZL       6'b010111

`define LB          6'b100000
`define LH          6'b100001
//`define   LWL         6'b100010       // Do not allow unaligned
`define LW          6'b100011
`define LBU         6'b100100
`define LHU         6'b100101
//`define   LWR         6'b100110       // Do not allow unaligned

`define SB          6'b101000
`define SH          6'b101001
//`define   SWL         6'b101010       // Do not allow unaligned
`define SW          6'b101011
```

```
`define SWR         6'b101110
`define CACHE       6'b101111

/*************************************
* Added the Network Stuff Here
* network ISA is an I type function
* These are the following instructions:
*
* Send Msg:  SMSG dest, Data
* ***********************************
* This instruction dispatches a message with the given data
* to the network layer
*
* Recieve Msg: RMSG dest, listen style, data
* ***********************************
* This will make the processor wait until a message is obtained
* in accordance with the listening style.  The possible styles:
* 0 - any message is recieved (l_any    00000)
* 1 - message from specific id  (l_id   00001)
* 2 - message from specific id and specific data    (l_idD  00010)
* 3 - message of specific data                      (l_D    00011)
* 4 - message from specific id/data from north      (l_idDN 00100)
* 5 -  *************************   from south      (l_idDS 00101)
* 6 -  *************************   from east       (l_idDE 00110)
* 7 -  *************************   from west       (l_idDW 00111)
*
* Set ID:   SID id_number
* ***********************************
* This will set the node id of the cpu both in the CPU
* and in the network layer
*
*/
`define NET_MEM_START 127   // arbitrary point in mem

// CPU Side
// *************************
`define SMSG        6'b110000
`define RMSG        6'b110001
`define SID         6'b110010
`define BCST        6'b110011   // broadcast message (just like smsg but
                                // with special destination ID
`define SMSGR       6'b111000   // Register based smsg
`define BCSTR       6'b111001   // Register based bcst

// Net Side
// *************************
`define JALNET      6'b110100   // Jump and link to Net Instructions
`define NACK        6'b110101   // Net Ack
`define SNIP        6'b110110   // Set Network Instruction Pointer
`define NDJR        6'b110111   // Network Driven Jump Register    // R Type

// defined listening styles
`define l_any       5'b00000
`define l_id        5'b00001
`define l_D         5'b00010
`define l_idDN      5'b00100
`define l_idDS      5'b00101
`define l_idDE      5'b00110
`define l_idDW      5'b00111

/*************************************
* Removed this from the ISA since we dont
* really use it anyways
*************************************/
/*
`define LL          6'b110000
`define LWC1        6'b110001
`define LWC2        6'b110010
`define LWC3        6'b110011

`define LDC1        6'b110101
`define LDC2        6'b110110
`define LDC3        6'b110111

`define SC          6'b111000
`define SWC1        6'b111001
`define SWC2        6'b111010
`define SWC3        6'b111011

`define SDC1        6'b111101
`define SDC2        6'b111110
`define SDC3        6'b111111
```

```
//**************************************/

// Opcode Assignments for `SPECIAL function Operations

`define SLL         6'b000000
`define SRL         6'b000010
`define SRA         6'b000011
`define SLLV        6'b000100
`define SRLV        6'b000110
`define SRAV        6'b000111

`define JR          6'b001000
`define JALR        6'b001001

`define SYSCALL     6'b001100
`define BREAK       6'b001101

`define MFHI        6'b010000
`define MTHI        6'b010001
`define MFLO        6'b010010
`define MTLO        6'b010011

`define MULT        6'b011000
`define MULTU       6'b011001
`define DIV         6'b011010
`define DIVU        6'b011011
`define MOD         6'b110010

`define ADD         6'b100000
`define ADDU        6'b100001
`define SUB         6'b100010
`define SUBU        6'b100011
`define AND         6'b100100
`define OR          6'b100101
`define XOR         6'b100110
`define NOR         6'b100111

`define SLT         6'b101010
`define SLTU        6'b101011

// IN OUT Instructions
`define IN          6'b110000
`define OUT         6'b110001

//`define   TGE         6'b110000
//`define   TGEU        6'b110001
//`define   TLT         6'b110010
`define TLTU        6'b110011
`define TEQ         6'b110100

`define TNE         6'b110110

// Opcode Assignments for `REGIMM rt Operations

//`define   BLTZ        5'b00000
//`define   BGEZ        5'b00001
`define BLTZL       5'b00010
`define BGEZL       5'b00011

`define TGEI        5'b01000
`define TGEIU       5'b01001
`define TLTI        5'b01010
`define TLTIU       5'b01011
`define TEQI        5'b01100

`define TNEI        5'b01110

`define BLTZAL      5'b10000
`define BGEZAL      5'b10001
`define BLTZALL     5'b10010
`define BGEZALL     5'b10011


// Opcode Assignments for `COPz rs Operations

`define MF          5'b00000
`define CF          5'b00010
`define MT          5'b00100
`define CT          5'b00110

`define BC          5'b01000
```

```
// Opcode Assignments for `COPz rt Operations

`define BCF         5'b00000
`define BCT         5'b00001
`define BCFL        5'b00010
`define BCTL        5'b00011

// Possible pc Selects

`define pc_sel_nex      5'b00000
`define pc_sel_jmp      5'b00001
`define pc_sel_br       5'b00010
`define pc_sel_jmp_reg  5'b00011
`define pc_sel_net_jmp  5'b00100

// Possible ALU operations

`define alu_add         5'b00000
`define alu_sub         5'b00001
`define alu_or          5'b00010
`define alu_and         5'b00011
`define alu_xor         5'b00100
`define alu_nor         5'b00101
`define alu_slt         5'b00110
`define alu_sltu        5'b00111
`define alu_sll         5'b01000
`define alu_srl         5'b01001
`define alu_sra         5'b01010
`define alu_mult        5'b01011
`define alu_multu       5'b01100
`define alu_div         5'b01101
`define alu_divu        5'b01110
`define alu_mod         5'b01111

// Network Layer Defines
// We have 4 directions (this is technically scalable)
/*****************************************************
* Message is 32 bits:
*  8 bit Destination ID
*  8 bit Message
*  8 bit Origination ID
*  4 bit Age
*  2 bit origin dispatch direction
*  2 bit last taken direction
*****************************************************/

`define delay           #10

`define north           2'b00
`define south           2'b01
`define east            2'b10
`define west            2'b11

`define dest_id         31:24
`define msg_data        23:16
`define origin_id       15:8
`define msg_age          7:4
`define o_dir            3:2
`define l_dir            1:0

`define OptVal          10


// PLI Decode Routine
```

NDMA Suite C/C++ Code

```
#include "BASM.h"

// Op Code Map
std::map<std::string, OPS> s_mapOP;

// Implement Label Data Structure
static std::map<std::string, int> s_mapLabel;

int main(int argc, char *argv[])
{
    int iTokenCount = 0;
    int fDone = 0;
    char input[140], output[40];
    char op[10];
    char rs[6] = "00000";
    char rt[6] = "00000";
    char rd[6] = "00000";
    char sa[10] = "00000";
    char function[10], immediate[20], target[35];
    int ins_type = 0;
    int rs_arg, rt_arg, rd_arg, op_arg, imm_arg, fn_arg, tar_arg;
    int sa_arg;
    int iPC = 0;
    int cLine = 0;
    int stacksize = 0;
    int bytecount = 0;
    int fSRI = 0;

    // Data Storage
    int cData = 0;
    bool fUseLabel = false;

    // our delimiter in these cases is a ' '
    char delims[] = " ,()\t";
    char *temp = NULL;
    char filename[20];
    char **arg;

    // Check for valid input file parameter
    if(argc < 2 || strlen(argv[1]) == 0)
    {
        printf("No Input File Specified!\n");
        printf("Usage: assembler.exe filename.asm\n");
        system("PAUSE");
        return 0;
    }

    FILE *outputFile;
    FILE *inputFile;

    // Get Input File Name
    strcpy(filename, argv[1]);

    // Create Output File
    temp = strtok(argv[1], ".");
    strcat(temp, ".mem");

    outputFile = fopen(temp, "w");
    inputFile = fopen(filename, "r");

    if(inputFile == NULL)
    {
        perror("error opening file!");
    }
    if(outputFile == NULL)
    {
        perror("error opening/creating output file!");
    }

    // Initialize the Op Map
    (void) InitializeOPMap();

    // Need to scan through the input file for labels
    while(fDone != 1)
    {
        int i = 0;
        bool skip = false;
        char input2[100];
        char input3[100];
        fgets(input, 100, inputFile);
        strcpy(input2, input);
        strcpy(input3, input);
```

```
        if(feof(inputFile))
        {
            // Go back to start of file and get out of loop
            rewind(inputFile);
            fDone = 1;
            cLine = 0;
            iPC = 0;
        }
        else
        {
            char *temp2 = NULL;
            char *temp3 = NULL;

            temp2 = strtok(input, " ,()\t");
            temp3 = strtok(input2, delims);

            if( strcmp(temp3, "la") == 0  )
            {
                int i = 0;
                char la_args[8][40];
                while(temp3 != NULL)
                {
                    strcpy(la_args[i], temp3);
                    temp3 = strtok(NULL, delims);
                    i++;
                }
                if(i == 3)
                {
                    // first must check if label (labels must have letters?)
                    bool fLabel = false;
                    for(int j = 0; j < strlen(la_args[2]) - 1; j++)
                    {
                        if((la_args[2][j] < 48 || la_args[2][j] > 57) && la_args[2][j] != '-')
                        {
                            fLabel = true;
                            break;
                        }
                    }
                    if(fLabel)
                    {
                        iPC += 2;
                    }
                    else
                    {
                        iPC++;
                    }
                }
                else
                {
                    iPC++;
                }
            }
            else if(strcmp(temp3, "sri") == 0)
            {
                // SRI will always require three instructions
                //fSRI = true;
                iPC += 3;
            }
            else if(temp2[0] != '\n')
            {
                temp2 = strtok(temp2, " \n");


                if(s_mapOP.count(temp2) != 0 || strcmp(temp2, ".SetStack") == 0 ||
                   strcmp(temp2, ".Boot") == 0 )
                {
                    iPC++;
                }
                else if( strcmp(temp2, ".byte") == 0 )
                {
                    iPC += 2;
                }

            }
            cLine++;
        }

        // check if label
        for(i = 0; i < strlen(input); i++)
        {
```

```
            if(input[i] == ':')
            {
                temp = strtok(input, " :");
                if(s_mapLabel.empty() || s_mapLabel.count(temp) == 0)
                {
                    printf("label: %s found at PC:%d Line: %d\n", temp, iPC, cLine++);
                    s_mapLabel[temp] = iPC;
                }
                else
                {
                    printf("%s label redefined at line: %d\n", temp, cLine);
                    system("PAUSE");
                    exit(0);
                }
                skip = true;
            }
        }
        if(skip) continue;
    }

    // Reset fDone
    fDone = 0;

    while (fDone != 1)
    {
        iTokenCount = 0;
        int i = 0;
        bool skip = false;
        fgets(input, 100, inputFile);
        fSRI = 0;

        if(feof(inputFile))
        {
            fDone = 1;
        }
        else
        {
            // increment line count
            cLine++;
        }

        iTokenCount = 0;

        // Quick and Dirty Token Count for allocation issues
        for(i = 0; i < strlen(input); i++)
        {
            if(input[i] == delims[0] || input[i] == delims[1] ||
               input[i] == delims[2] || input[i] == delims[3] ||
               input[i] == delims[4] )
            {
                iTokenCount++;
            }
        }

        if(iTokenCount == 0 && strlen(input) > 0)
        {
            iTokenCount = 1;
        }

        temp = strtok(input, delims);
        i = 0;
        arg = new char*[iTokenCount];

        while(temp != NULL)
        {
               arg[i] = new char[strlen(temp)];
               strcpy(arg[i], temp);
               temp = strtok(NULL, delims);
               i++;
        } // end while

        // One Line Comments
        if(arg[0][0] == '#')
        {
            // skip the line
            continue;
        }

        // Check for SRI
        if(strcmp(arg[0], "sri") == 0)
        {
            // SRI will always require three instructions
```

```
            fSRI = 2;
            iTokenCount -= 1;
            //iPC += 3;
        }

        // Directives
        if(arg[0][0] == '.')
        {
            // Store Directive in temp
            temp = strtok(arg[0], ".");
            if(strcmp(temp, "SetStack") == 0)
            {
                cData = 0;
                if(iTokenCount == 1)
                {
                    char *tempy = strtok(arg[1], "/n");
                    strcpy(immediate, dTob(atoi(tempy)));
                    printf("0011010000011101%s // ori $sp, $0, %d (SetStack)\n", immediate, atoi(tempy));
                    fprintf(outputFile, "0011010000011101%s // ori $sp, $0, %d (SetStack)\n", immediate, atoi(
                        tempy));
                    iPC++;
                    stacksize = atoi(tempy);
                }
                else
                {
                    printf("Stack Size not designated\n");
                }

            }
            else if(strcmp(temp, "Boot") == 0)
            {
                cData = 0;
                if(iTokenCount == 1)
                {
                    int target_pc = 0;

                    // Get PC of program entry point
                    strcpy(op, "000010");
                    ins_type = J_TYPE;

                    // Get rid of termination character
                    char *tempy = strtok(arg[1], "\n");

                    // Check for Label
                    if(s_mapLabel.count(tempy) != 0)
                    {
                        strcpy(target, dTob26(s_mapLabel[tempy]));
                        target_pc = s_mapLabel[tempy];
                    }
                    else
                    {
                        strcpy(target, dTob26(atoi(arg[1])));
                        target_pc = atoi(arg[1]);
                    }

                    strcpy(output, op);
                    strcat(output, target);
                    printf("%s // %s %s pc:%d\n", output, arg[0], arg[1], target_pc);
                    fprintf(outputFile, "%s // %s %s pc:%d\n", output, arg[0], arg[1], target_pc);
                    strcpy(immediate, dTob(atoi(tempy)));
                    iPC++;
                }
                else
                {
                    printf("Boot point not designated\n");
                }
            }
            else if(strcmp(temp, "byte") == 0)
            {
                if(iTokenCount == 1)
                {
                    // Two instructions required, load value into $1
                    // then store $1 into memory:
                    // ori $1, $0, value
                    // sw $1, cData($0)
                    // ori
                    // 001101 00000 00001 0000000000000101 // ori $1, $0, 5
                    // sw
                    // 101011 00000 00001 0000000000000100 // sw $1, 4($0)


                    // Get rid of termination character
```

```
                char *tempy = strtok(arg[1], "\n");

                strcpy(immediate, dTob(atoi(tempy)));
                printf("00110100000001000%s // ori $8, $0, %d (.byte)\n",
                       immediate, atoi(tempy));
                fprintf(outputFile, "00110100000001000%s // ori $8, $0, %d (.byte)\n",
                       immediate, atoi(tempy));

                strcpy(immediate, dTob(cData));
                printf("10101100000001000%s // sw $8, %d($0) (.byte)\n",
                        immediate, cData);
                fprintf(outputFile, "10101100000001000%s // sw $8, %d($0) (.byte)\n",
                        immediate, cData);



                //printf(".byte store:%d at %d\n", atoi(tempy), cData);
                cData++;
                iPC += 2;
            }
            else
            {
                printf(".byte incorrect argument\n");
            }
        }
        else cData = 0;


        continue;
    }

    // Now we need to split up the instructions into different types
    // start off left to right
    if(iTokenCount - (int)(fSRI) > 8) {
           printf("Error: too many arguments %d\n", iTokenCount);
    //} else if(iTokenCount == 4) {
         // R type
         // ______________________________________________________________
         // | op * 6 | rs * 5 | rt * 5 | rd * 5 | sa * 5 | function * 6 |
         // template:
         // opopoprssssrttttrddddsaaaafuncti

         // op code


    }
    else if(iTokenCount - (int)(fSRI) <= 8 && iTokenCount - (int)(fSRI) > 2)
    {
         // I type
         // ________________________________________________
         // | op * 6 | rs * 5 | rt * 5  | immediate * 16 |
         // template
         // opopoprssssrttttimmediateimmedia
         // op code

        // ************** I_TYPE ***************
        ins_type = OpFunctionSADecode(arg[0 + (int)fSRI], op, function, sa);

        fUseLabel = false;

        if(ins_type == NOT_TYPE)
        {
            // Op Decode Failure
            printf("OP Decode Failure!\n");
            continue;
        }


        if(ins_type == I_TYPE) {
            if(s_mapOP[arg[0 + (int)fSRI]] == OP_LUI || s_mapOP[arg[0 + (int)fSRI]] == OP_SMSG ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_BCSTR )
            {
                imm_arg = 2 + (int)fSRI;
            }
            else
            {
                imm_arg = 3 + (int)fSRI;
            }

            rs_arg = 2 + (int)fSRI;
            rt_arg = 1 + (int)fSRI;
```

```
            if(s_mapOP[arg[0 + (int)fSRI]] == OP_LW ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_LH || s_mapOP[arg[0 + (int)fSRI]] == OP_LHU ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_LB || s_mapOP[arg[0 + (int)fSRI]] == OP_LBU ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_SW ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_SH ||
               s_mapOP[arg[0 + (int)fSRI]] == OP_SB )
            {
                imm_arg = 2 + (int)fSRI;
                rs_arg = 3 + (int)fSRI;
                rt_arg = 1 + (int)fSRI;
            }
            else if(s_mapOP[arg[0 + (int)fSRI]] == OP_BGTZ || s_mapOP[arg[0 + (int)fSRI]] == OP_BGEZ ||
                    s_mapOP[arg[0 + (int)fSRI]] == OP_BLTZ || s_mapOP[arg[0 + (int)fSRI]] == OP_BLEZ ||
                    s_mapOP[arg[0 + (int)fSRI]] == OP_OUTI )
            {
                imm_arg = 2 + (int)fSRI;
                rs_arg = 1 + (int)fSRI;
            }

        } else if(ins_type == R_TYPE) {
            if(s_mapOP[arg[0 + (int)fSRI]] == OP_IN || s_mapOP[arg[0 + (int)fSRI]] == OP_OUT )
            {
                rd_arg = 1 + (int)fSRI;
                rs_arg = 2 + (int)fSRI;
                rt_arg = 2 + (int)fSRI; // duplicate rs into rt since rt is dont care
            }
            else
            {
                rs_arg = 2 + (int)fSRI;
                rt_arg = 3 + (int)fSRI;
                rd_arg = 1 + (int)fSRI;
            }
            //sa_arg = 4;
            //fn_arg = 5;
        }
        else if(ins_type == SR_TYPE)
        {
            rd_arg = 1 + (int)fSRI;
            rs_arg = 2 + (int)fSRI;
            rt_arg = 2 + (int)fSRI;
            sa_arg = 3 + (int)fSRI;
        }
        else if(ins_type == J_TYPE)
        {
            // J Type Instructions
        }
        else if(ins_type == P_TYPE)
        {
            if(s_mapOP[arg[0 + (int)fSRI]] == OP_LA)
            {
                rt_arg = 1 + (int)fSRI;
                imm_arg = 2 + (int)fSRI;
                rs_arg = 3 + (int)fSRI;
            }
        }

        // RD only used for R and SR types
        // sid doesnt have rs or rd
        if(strcmp(arg[0 + (int)fSRI], "sid") == 0)
        {
            strcpy(rd, "00000");
        }
        else if(ins_type == R_TYPE || ins_type == SR_TYPE)
        {
            RegDecode(arg[rd_arg], rd);
        }
        else
        {
            strcpy(rd, "00000");
        }
        // Cap the string
        rd[5] = '\0';


         // RS = 0 always for lui, smsg, sid
        if(strcmp(arg[0 + (int)fSRI], "lui") == 0 || strcmp(arg[0 + (int)fSRI], "smsg") == 0 ||
           s_mapOP[arg[0 + (int)fSRI]] == OP_BCSTR ||
           strcmp(arg[0 + (int)fSRI], "sid") == 0 || (s_mapOP[arg[0 + (int)fSRI]] == OP_LA && iTokenCount <= 3
               ))
        {
            strcpy(rs, "00000");
```

```
        }
        else if(ins_type != J_TYPE)
        {
            RegDecode(arg[rs_arg], rs);
        }
        else
        {
            strcpy(rs, "00000");
        }
        rs[5] = '\0';

         // RT
         if(strcmp(arg[0 + (int)fSRI], "sid") == 0 )
         {
            strcpy(rt, "00000");
         }
         else if(ins_type != J_TYPE)
         {
            RegDecode(arg[rt_arg], rt);
         }
         else
         {
             strcpy(rt, "00000");
         }
         rt[5] = '\0';

         // Immediate
         // only need to worry about labels for immediate
         if(ins_type == I_TYPE || ins_type == P_TYPE)
         {
             // Get rid of termination character
             char *tempy = strtok(arg[imm_arg], "\n");

             if(s_mapLabel.count(tempy) > 0)
             {
                 int target_pc;
                 if( s_mapOP[arg[0 + (int)fSRI]] == OP_BNE ||
                     s_mapOP[arg[0 + (int)fSRI]] == OP_BEQ ||
                     s_mapOP[arg[0 + (int)fSRI]] == OP_BGTZ ||
                     s_mapOP[arg[0 + (int)fSRI]] == OP_BLEZ)
                 {
                    target_pc = s_mapLabel[tempy] - iPC;
                    if(target_pc > 0) target_pc--;  // jump ahead one already
                 }
                 else
                 {
                    fUseLabel = true;
                    target_pc = s_mapLabel[tempy];
                    strcpy(target, dTob26(target_pc));
                 }

                 strcpy(immediate, dTob(target_pc));
                 //printf("cur PC:%d br:%d\n", iPC, target_pc);
             }
             else if(arg[imm_arg][0] != 'x')
             {
                 // Apparently we need to potentially tokenize
                 // the immediate string to handle operands
                 int operands[4];
                 int opcount = 0;
                 char imm_arg_cpy[20];

                 strcpy(imm_arg_cpy, arg[imm_arg]);

                 char *tempy = strtok(imm_arg_cpy, " +");

                 while(tempy != NULL)
                 {
                    operands[opcount] = atoi(tempy);
                    opcount++;
                    tempy = strtok(NULL, " +");
                 }
                 if(opcount == 1)
                 {
                    strcpy(immediate, dTob(atoi(imm_arg_cpy)));
                 }
                 else
                 {
                    int temp_imm = 0;
                    for(int i = 0; i < opcount; i++)
                    {
                        temp_imm += operands[i];
```

```
                }
                strcpy(immediate, dTob(temp_imm));
            }
        }
        else
        {
            strcpy(immediate, hTob(arg[imm_arg]));
        }
    }
    else
    {
        strcpy(immediate, "0000000000000000");
    }

    // Shift Amount
    if(ins_type == SR_TYPE)
    {
       char temp_sa[20];
       strcpy(temp_sa, dTob(atoi(arg[sa_arg])));
       for(int c = 0; c < 5; c++)
       {
           sa[4 - c] = temp_sa[15 - c];
       }
    }
    else
    {
       strcpy(sa, "00000");
    }

   // ASSEMBLE INSTRUCTIONS
    if(ins_type == I_TYPE)
    {
       strcpy(output, op);
       if( s_mapOP[arg[0 + (int)fSRI]] == OP_BCSTR )
       {
           // Flip rs and rt order for BCSTR
           strcat(output, rt);
           strcat(output, rs);
       }
       else
       {
           strcat(output, rs);
           strcat(output, rt);
       }

       strcat(output, immediate);

       if(strcmp(arg[0 + (int)fSRI], "lui") == 0 || strcmp(arg[0 + (int)fSRI], "smsg") == 0 ||
          s_mapOP[arg[0 + (int)fSRI]] == OP_BCSTR )
       {
           if(!fSRI)
           {
               printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
               fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
               iPC++;
           }
           else
           {
               char temp_reg[6];
               char first_half[17];
               char second_half[17];
               char *half_ptr = &output[16];

               // Reg Decode Destination
               RegDecode(arg[1], temp_reg);
               // First half
               strncpy(first_half, output, 16);
               first_half[16] = '\0';
               // Second Half
               strncpy(second_half, half_ptr, 16);
               second_half[16] = '\0';

               // ORI to reset the register
               printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                   temp_reg, arg[1], arg[0], arg[1], arg[2]);
               fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                   temp_reg, arg[1], arg[0], arg[1], arg[2]);

               // LUI to set first half
               printf("00111100000%s%s // lui %s, %s (SRI)\n",
                   temp_reg, first_half, arg[1], first_half);
               fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
```

```
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if(s_mapOP[arg[0 + (int)fSRI]] == OP_LW ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_LH || s_mapOP[arg[0 + (int)fSRI]] == OP_LHU ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_LB || s_mapOP[arg[0 + (int)fSRI]] == OP_LBU ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_SW ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_SH ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_SB )
        {
            if(!fSRI)
            {
                printf("%s // %s %s, %s(%s)\n", output, arg[0], arg[1], arg[2], arg[3]);
                fprintf(outputFile, "%s // %s %s, %s(%s)\n", output, arg[0], arg[1], arg[2], arg[3]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if(s_mapOP[arg[0 + (int)fSRI]] == OP_BGTZ || s_mapOP[arg[0 + (int)fSRI]] == OP_BGEZ ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_BLTZ || s_mapOP[arg[0 + (int)fSRI]] == OP_BLEZ ||
                s_mapOP[arg[0 + (int)fSRI]] == OP_OUTI )
        {
            if(!fSRI)
            {
                printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
```

```
                    first_half[16] = '\0';
                    // Second Half
                    strncpy(second_half, half_ptr, 16);
                    second_half[16] = '\0';

                    // ORI to reset the register
                    printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);

                    // LUI to set first half
                    printf("00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);
                    fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);

                    // ORI to set first half
                    printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                    fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                    iPC += 3;
                }
            }
            else
            {
                if(!fSRI)
                {
                    printf("%s // %s %s, %s, %s\n", output, arg[0], arg[1], arg[2], arg[3]);
                    fprintf(outputFile, "%s // %s %s, %s, %s\n", output, arg[0], arg[1], arg[2], arg[3]);
                    iPC++;
                }
                else
                {
                    char temp_reg[6];
                    char first_half[17];
                    char second_half[17];
                    char *half_ptr = &output[16];

                    // Reg Decode Destination
                    RegDecode(arg[1], temp_reg);
                    // First half
                    strncpy(first_half, output, 16);
                    first_half[16] = '\0';
                    // Second Half
                    strncpy(second_half, half_ptr, 16);
                    second_half[16] = '\0';

                    // ORI to reset the register
                    printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);

                    // LUI to set first half
                    printf("00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);
                    fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);

                    // ORI to set first half
                    printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                    fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                    iPC += 3;
                }
            }
        }
        else if(ins_type == P_TYPE)
        {
            strcpy(output, op);
            strcat(output, rs);
            strcat(output, rt);
            //strcat(output, "00000");
            strcat(output, immediate);

            if(s_mapOP[arg[0 + (int)fSRI]] == OP_LA && iTokenCount > 3)
            {
```

```
                printf("%s // %s %s, %s(%s)\n", output, arg[0], arg[1], arg[2], arg[3]);
                fprintf(outputFile, "%s // %s %s, %s(%s)\n", output, arg[0], arg[1], arg[2], arg[3]);
                iPC++;
            }
            if(s_mapOP[arg[0 + (int)fSRI]] == OP_LA && iTokenCount == 3)
            {
                if(fUseLabel == true)
                {
                    // In case of Label use means we're loading in a
                    // data address so we jal and then save value into register
                    printf("000011%s // jal %s (data load)\n", target, arg[2]);
                    fprintf(outputFile, "000011%s // jal %s (data load)\n", target, arg[2]);

                    printf("00000000000000001%s00000100000 // add %s, $0, $1 (data load)\n", rt, arg[1]);
                    fprintf(outputFile, "00000000000000001%s00000100000 // add %s, $0, $1 (data load)\n", rt,
                        arg[1]);
                    iPC += 2;
                }
                else
                {
                    printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                    iPC++;
                }

                //printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                //fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                //iPC++;
            }
        }
        else if(ins_type == SR_TYPE)
        {
            strcpy(output, op);
            strcat(output, rs);
            strcat(output, rt);
            strcat(output, rd);
            strcat(output, sa);
            strcat(output, function);
            if(!fSRI)
            {
                printf("%s // %s %s, %s, %s", output, arg[0], arg[1], arg[2], arg[3]);
                fprintf(outputFile, "%s // %s %s, %s, %s", output, arg[0], arg[1], arg[2], arg[3]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if(ins_type = R_TYPE)
```

```
        {
            strcpy(output, op);
            strcat(output, rs);
            strcat(output, rt);
            strcat(output, rd);
            strcat(output, sa);
            strcat(output, function);
            if(s_mapOP[arg[0]] == OP_IN || s_mapOP[arg[0]] == OP_OUT)
            {
                if(!fSRI)
                {
                    printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
                    iPC++;
                }
                else
                {
                    char temp_reg[6];
                    char first_half[17];
                    char second_half[17];
                    char *half_ptr = &output[16];

                    // Reg Decode Destination
                    RegDecode(arg[1], temp_reg);
                    // First half
                    strncpy(first_half, output, 16);
                    first_half[16] = '\0';
                    // Second Half
                    strncpy(second_half, half_ptr, 16);
                    second_half[16] = '\0';

                    // ORI to reset the register
                    printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);

                    // LUI to set first half
                    printf("00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);
                    fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);

                    // ORI to set first half
                    printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                    fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                    iPC += 3;
                }
            }
            else
            {
                if(!fSRI)
                {
                    printf("%s // %s %s, %s, %s\n", output, arg[0], arg[1], arg[2], arg[3]);
                    fprintf(outputFile, "%s // %s %s, %s, %s\n", output, arg[0], arg[1], arg[2], arg[3]);
                    iPC++;
                }
                else
                {
                    char temp_reg[6];
                    char first_half[17];
                    char second_half[17];
                    char *half_ptr = &output[16];

                    // Reg Decode Destination
                    RegDecode(arg[1], temp_reg);
                    // First half
                    strncpy(first_half, output, 16);
                    first_half[16] = '\0';
                    // Second Half
                    strncpy(second_half, half_ptr, 16);
                    second_half[16] = '\0';

                    // ORI to reset the register
                    printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);
                    fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                        temp_reg, arg[1], arg[0], arg[1], arg[2]);
```

```
                    // LUI to set first half
                    printf("00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);
                    fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                        temp_reg, first_half, arg[1], first_half);

                    // ORI to set first half
                    printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                    fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                    iPC += 3;
                }
            }
        }
    }
    /*else if(iTokenCount - (int)(fSRI) == 2)
    {
        // J type
        // ____________________
        // | op * 6 | target * 26|
        // template
        // opopoptargettargettargettargetta
        /*if(s_mapOP[arg[0]] == OP_LA)
        {
            strcpy(output, op);
            strcat(output, rs);
            strcat(output, rt);
            //strcat(output, "00000");
            strcat(output, immediate);
            printf("%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
            fprintf(outputFile, "%s // %s %s, %s\n", output, arg[0], arg[1], arg[2]);
            iPC++;
        }

        printf("J Type instruction\n");
    }*/
    else if(iTokenCount - (int)(fSRI) <= 2)
    {

        if( strcmp(arg[0 + (int)fSRI], "nop\n") == 0 || strcmp(arg[0 + (int)fSRI], "nop") == 0 )
        {
            memset(output, '0', 32);
            output[32] = '\0';

            if(!fSRI)
            {
                printf("%s // %s", output, arg[0]);
                fprintf(outputFile, "%s // %s", output, arg[0]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
```

```
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_JALNET )
        {
            strcpy(op, "110100");
            ins_type = J_TYPE;
            strcpy(target, dTob26(0));
            strcpy(output, op);
            strcat(output, target);

            if(!fSRI)
            {
                printf("%s // %s", output, arg[0]);
                fprintf(outputFile, "%s // %s", output, arg[0]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_NDJR )
        {
            strcpy(op, "000000");
            ins_type = J_TYPE;

            // Get rid of termination character
            char *tempy = strtok(arg[1 + (int)fSRI], "\n");

            // Convert to register code
            RegDecode(tempy, rs);

            strcpy(output, op);
            strcat(output, rs);
            strcat(output, "00000000000000110111");
            if(!fSRI)
            {
                printf("%s // %s %s\n", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s\n", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
```

```
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_BCST )
        {
            strcpy(op, "110011");
            ins_type = J_TYPE;
            strcpy(target, dTob26(atoi(arg[1 + (int)fSRI])));
            strcpy(output, op);
            strcat(output, target);
            if(!fSRI)
            {
                printf("%s // %s %s", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
```

```
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_SNIP )
        {
            strcpy(op, "110110");
            ins_type = J_TYPE;
            strcpy(target, dTob26(atoi(arg[1 + (int)fSRI])));
            strcpy(output, op);
            strcat(output, target);
            if(!fSRI)
            {
                printf("%s // %s %s", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_SID )
        {
            strcpy(op, "110010");
            ins_type = J_TYPE;
            strcpy(target, dTob26(atoi(arg[1 + (int)fSRI])));
            strcpy(output, op);
            strcat(output, target);
            if(!fSRI)
            {
                printf("%s // %s %s", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
```

```
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_J )
        {
            strcpy(op, "000010");
            ins_type = J_TYPE;

            // Get rid of termination character
            char *tempy = strtok(arg[1 + (int)fSRI], "\n");

            // Check for Label
            if(s_mapLabel.count(tempy) != 0)
            {
                strcpy(target, dTob26(s_mapLabel[tempy]));
            }
            else
            {
                strcpy(target, dTob26(atoi(arg[1])));
            }

            strcpy(output, op);
            strcat(output, target);

            if(!fSRI)
            {
                printf("%s // %s %s\n", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s\n", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
```

```
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_JAL )
        {
            strcpy(op, "000011");
            ins_type = J_TYPE;
            int target_pc = 0;

            // Get rid of termination character
            char *tempy = strtok(arg[1 + (int)fSRI], "\n");

            // Check for Label
            if(s_mapLabel.count(tempy) != 0)
            {
                strcpy(target, dTob26(s_mapLabel[tempy]));
                target_pc = s_mapLabel[tempy];
            }
            else
            {
                strcpy(target, dTob26(atoi(arg[1])));
                target_pc = atoi(arg[1]);
            }

            strcpy(output, op);
            strcat(output, target);
            if(!fSRI)
            {
                printf("%s // %s %s (%d)\n", output, arg[0], arg[1], target_pc);
                fprintf(outputFile, "%s // %s %s (%d)\n", output, arg[0], arg[1], target_pc);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s (%d))\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2], target_pc);
                fprintf(outputFile, "001101000000%s0000000000000000 // ori %s,$0,0(%s %s, %s (%d))\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2], target_pc);

                // LUI to set first half
                printf("001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "001111000000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if( s_mapOP[arg[0 + (int)fSRI]] == OP_JR )
        {
            strcpy(op, "000000");
            ins_type = J_TYPE;

            // Get rid of termination character
            char *tempy = strtok(arg[1 + (int)fSRI], "\n");

            // Convert to register code
            RegDecode(tempy, rs);

            strcpy(output, op);
            strcat(output, rs);
            strcat(output, "00000000000000001000");
            if(!fSRI)
```

```
            {
                printf("%s // %s %s\n", output, arg[0], arg[1]);
                fprintf(outputFile, "%s // %s %s\n", output, arg[0], arg[1]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                    temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                iPC += 3;
            }
        }
        else if(s_mapOP[arg[0 + (int)fSRI]] == OP_BREAK)
        {
            ins_type = R_TYPE;
            strcpy(output, "00000000000000000000000000001101 ");
            if(!fSRI)
            {
                printf("%s // %s\n", output, arg[0]);
                fprintf(outputFile, "%s // %s\n", output, arg[0]);
                iPC++;
            }
            else
            {
                char temp_reg[6];
                char first_half[17];
                char second_half[17];
                char *half_ptr = &output[16];

                // Reg Decode Destination
                RegDecode(arg[1], temp_reg);
                // First half
                strncpy(first_half, output, 16);
                first_half[16] = '\0';
                // Second Half
                strncpy(second_half, half_ptr, 16);
                second_half[16] = '\0';

                // ORI to reset the register
                printf("00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);
                fprintf(outputFile, "00110100000%s0000000000000000 // ori %s,$0,0(%s %s, %s)\n",
                    temp_reg, arg[1], arg[0], arg[1], arg[2]);

                // LUI to set first half
                printf("00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);
                fprintf(outputFile, "00111100000%s%s // lui %s, %s (SRI)\n",
                    temp_reg, first_half, arg[1], first_half);

                // ORI to set first half
                printf("001101%s%s%s // ori %s, %s, %s (SRI)\n",
```

```
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);
                    fprintf(outputFile, "001101%s%s%s // ori %s, %s, %s (SRI)\n",
                        temp_reg, temp_reg, second_half, arg[1], arg[1], second_half);

                    iPC += 3;
                }
            }
            else if( strcmp(arg[0 + (int)fSRI], "quit") == 0)
            {
                printf("Done!");
                fDone = 1;
                break;
            }
        }
        else if(iTokenCount == 0)
        {
               printf("no arguments\n");
        }
    }

    printf("Results\n");
    printf("****************************************\n");
    printf("Total lines of assembly: %d\n", iPC);
    if(iPC >= stacksize)
    {
        printf("ERROR: Program will not fit in memory of size %d\n", stacksize);
    }
    else
    {
        printf("OK FIT: Program takes up %2.1f%% of %d length memory\n",
            100.0*((float)(iPC)/(float)(stacksize)), stacksize);
    }
    printf("****************************************\n\n");

    // Clean up
    fclose(outputFile);
    fclose(inputFile);

    system("PAUSE");
    strcpy(input, "done");
}
```

```
// Mips-Like Assembler ver 0.4
// Idan Beck, 2007
/*********************************************************************
* This is a mips like assembler takes in a mips like input assembly file
* and outputs a machine code file with annotated instruction comments as
* is acceptable by verilog standards so that the output can be used as
* a memory file for initializing a instruction memory.
*
* Bugs:
* - Does not deal with jump / branch instructions
* - Does not deal with load / store instructions
*
* Planned Work :
* 1. Implement jump instructions
* 2. Implement branch instructions
* 3. Implement loads and stores
*
* apr11: SMSG added
* apr12: SID added
*********************************************************************/

#pragma once

#define _CRT_SECURE_NO_DEPRECATE 1

#include <map>
#include <string>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

#define DELIMIT ' ()'



enum OPS {
    // Memory Ops
    OP_LW,
    OP_LH,
    OP_LHU,
    OP_LB,
    OP_LBU,
    OP_SW,
    OP_SH,
    OP_SB,

    // Network Ops
    // *************
    // CPU Side
    OP_SMSG,
    OP_RMSG,
    OP_SID,
    OP_BCST,
    OP_SMSGR,
    OP_BCSTR,
    // Net Side
    OP_JALNET,
    OP_NACK,
    OP_SNIP,
    OP_NDJR,
    // *************

    // Immediate OPs
    OP_LUI,
    OP_ADDI,
    OP_ADDIU,
    OP_SLTI,
    OP_SLTIU,
    OP_ANDI,
    OP_XORI,
    OP_ORI,
    OP_ADD,
    OP_ADDU,
    OP_SUB,
    OP_SUBU,
    OP_AND,
    OP_OR,
    OP_XOR,
    OP_SLT,
    OP_SLTU,
    OP_SLL,
    OP_SRL,
```

```
    OP_SRA,

    // Branching
    OP_BNE,
    OP_BEQ,

    OP_BGTZ,
    OP_BGEZ,
    OP_BLTZ,
    OP_BLEZ,

    OP_J,
    OP_JR,
    OP_JAL,

    // IN/OUT Instructions
    OP_IN,
    OP_OUT,
    OP_OUTI,

    // Mult div mod
    OP_MULT,
    OP_MULTU,
    OP_DIV,
    OP_DIVU,
    OP_MOD,

    // PSEUDO Instructions
    OP_LA,
    OP_SRI,

    // Control OPS
    OP_NOP,
    OP_BREAK,
    OP_QUIT
};

enum InstructionTypes {
    I_TYPE,
    J_TYPE,
    R_TYPE,
    SR_TYPE,
    P_TYPE,
    NET_TYPE,
    NOT_TYPE
};

char* dTob(int input);
char* dTob26(int input);
char* hTob(char *hex);

int RegDecode(char *pszReg, char *pszOut);

void InitializeOPMap();
int OpFunctionSADecode(char *AsmOp, char* OpCode, char* FunctionCode, char *SACode);
```

```cpp
#include "BASM.h"

// Op Code Map
extern std::map<std::string, OPS> s_mapOP;

// 16 bit immediate converter (support for signed variables)
char* dTob(int input){
	//int i, j, k;
	int negFlag = 0;
	char toutput[17];
	char output[17];

	if(input < 0)
	{
		negFlag = 1;
		input = (-1 * input) - 1;
	}

	// do the first one for sake of string stuff
	strcpy(toutput, "");

	while(input > 0)
	{
			if(input & 1 == 1)
			{
			    if(negFlag) strcat(toutput, "0");
			    else strcat(toutput, "1");
			}
			else
			{
			    if(negFlag) strcat(toutput, "1");
			    else strcat(toutput, "0");
			}
			input >>= 1;
	}

	//output = new char[16];
	if(negFlag) memset(output, '1', sizeof(output));
	else memset(output, '0', sizeof(output));

	output[16] = '\0';

	for(int i = 15; i >= 0; i--)
	{
	    if(strlen(toutput) > (15 - i) ) output[i] = toutput[15 - i];
	}
	output[16] = '\0';

	return output;
}

// 26 bit immediate converter (unsigned only)

char* dTob26(int input){
	int i, j, k;
	int negFlag = 0;
	char toutput[27];
	char output[27];

	if(input < 0){
		negFlag = 1;
		input = input*-1;
	}

	// do the first one for sake of string stuff
	if(input&1 == 1)strcpy(toutput, "1");
	else strcpy(toutput, "0");
	input >>= 1;

	while(input > 0) {
			if(input&1 == 1)strcat(toutput, "1");
			else strcat(toutput, "0");
			input >>= 1;
	}

	//output = new char[16];
	memset(output, '0', sizeof(output));
	output[26] = '\0';

	for(i=25;i>=0;i--){
	    if(strlen(toutput) > (25 - i) ) output[i] = toutput[25 - i];
	}
```

```
      output[26] = '\0';

      return output;
}

// returns 16 bit hex number

char* hTob(char *hex){
    char toutput[17];
    char output [17];
    int len;

    memset(toutput, '\0', 17);
    for(len = 0;len < strlen(hex);len++) {
        switch(hex[len]) {
            case '0':   strcat(toutput, "0000");
                        break;
            case '1':   strcat(toutput, "0001");
                        break;
            case '2':   strcat(toutput, "0010");
                        break;
            case '3':   strcat(toutput, "0011");
                        break;
            case '4':   strcat(toutput, "0100");
                        break;
            case '5':   strcat(toutput, "0101");
                        break;
            case '6':   strcat(toutput, "0110");
                        break;
            case '7':   strcat(toutput, "0111");
                        break;
            case '8':   strcat(toutput, "1000");
                        break;
            case '9':   strcat(toutput, "1001");
                        break;
            case 'A':   strcat(toutput, "1010");
                        break;
            case 'B':   strcat(toutput, "1011");
                        break;
            case 'C':   strcat(toutput, "1100");
                        break;
            case 'D':   strcat(toutput, "1101");
                        break;
            case 'E':   strcat(toutput, "1110");
                        break;
            case 'F':   strcat(toutput, "1111");
                        break;
            case 'x':   if(len != 0) strcat(toutput, "xxxx");
                        break;
            default:    strcat(toutput, "xxxx");
                        break;
        }
    }

    toutput[16] = '\0';

    memset(output, '0', sizeof(output));
    output[16] = '\0';
    len = 0;
    for(int i=16 - strlen(toutput) ;i<16;i++){
        output[i] = toutput[len];
        len++;
    }
    output[16] = '\0';

    return output;
}

int RegDecode(char *pszReg, char *pszOut)
{
    if(pszReg == NULL)
    {
        if(pszOut == NULL)
        {
            return 0;
        }
        else
        {
            strcpy(pszOut, "00000");
            return 0;
        }
    }
```

```
    // Get rid of termination character
    char *tempy = strtok(pszReg, "\n");

    if(     strcmp(tempy, "$0") == 0  || strcmp(tempy, "$zero") == 0 ) strncpy(pszOut, dTob(0)+11, 5);
    else if(strcmp(tempy, "$1") == 0  || strcmp(tempy, "$at") == 0 )   strncpy(pszOut, dTob(1)+11, 5);
    else if(strcmp(tempy, "$2") == 0  || strcmp(tempy, "$v0") == 0 )   strncpy(pszOut, dTob(2)+11, 5);
    else if(strcmp(tempy, "$3") == 0  || strcmp(tempy, "$v1") == 0 )   strncpy(pszOut, dTob(3)+11, 5);
    else if(strcmp(tempy, "$4") == 0  || strcmp(tempy, "$a0") == 0 )   strncpy(pszOut, dTob(4)+11, 5);
    else if(strcmp(tempy, "$5") == 0  || strcmp(tempy, "$a1") == 0 )   strncpy(pszOut, dTob(5)+11, 5);
    else if(strcmp(tempy, "$6") == 0  || strcmp(tempy, "$a2") == 0 )   strncpy(pszOut, dTob(6)+11, 5);
    else if(strcmp(tempy, "$7") == 0  || strcmp(tempy, "$a3") == 0 )   strncpy(pszOut, dTob(7)+11, 5);
    else if(strcmp(tempy, "$8") == 0  || strcmp(tempy, "$t0") == 0 )   strncpy(pszOut, dTob(8)+11, 5);
    else if(strcmp(tempy, "$9") == 0  || strcmp(tempy, "$t1") == 0 )   strncpy(pszOut, dTob(9)+11, 5);
    else if(strcmp(tempy, "$10") == 0 || strcmp(tempy, "$t2") == 0 )   strncpy(pszOut, dTob(10)+11, 5);
    else if(strcmp(tempy, "$11") == 0 || strcmp(tempy, "$t3") == 0 )   strncpy(pszOut, dTob(11)+11, 5);
    else if(strcmp(tempy, "$12") == 0 || strcmp(tempy, "$t4") == 0 )   strncpy(pszOut, dTob(12)+11, 5);
    else if(strcmp(tempy, "$13") == 0 || strcmp(tempy, "$t5") == 0 )   strncpy(pszOut, dTob(13)+11, 5);
    else if(strcmp(tempy, "$14") == 0 || strcmp(tempy, "$t6") == 0 )   strncpy(pszOut, dTob(14)+11, 5);
    else if(strcmp(tempy, "$15") == 0 || strcmp(tempy, "$t7") == 0 )   strncpy(pszOut, dTob(15)+11, 5);
    else if(strcmp(tempy, "$16") == 0 || strcmp(tempy, "$s0") == 0 )   strncpy(pszOut, dTob(16)+11, 5);
    else if(strcmp(tempy, "$17") == 0 || strcmp(tempy, "$s1") == 0 )   strncpy(pszOut, dTob(17)+11, 5);
    else if(strcmp(tempy, "$18") == 0 || strcmp(tempy, "$s2") == 0 )   strncpy(pszOut, dTob(18)+11, 5);
    else if(strcmp(tempy, "$19") == 0 || strcmp(tempy, "$s3") == 0 )   strncpy(pszOut, dTob(19)+11, 5);
    else if(strcmp(tempy, "$20") == 0 || strcmp(tempy, "$s4") == 0 )   strncpy(pszOut, dTob(20)+11, 5);
    else if(strcmp(tempy, "$21") == 0 || strcmp(tempy, "$s5") == 0 )   strncpy(pszOut, dTob(21)+11, 5);
    else if(strcmp(tempy, "$22") == 0 || strcmp(tempy, "$s6") == 0 )   strncpy(pszOut, dTob(22)+11, 5);
    else if(strcmp(tempy, "$23") == 0 || strcmp(tempy, "$s7") == 0 )   strncpy(pszOut, dTob(23)+11, 5);
    else if(strcmp(tempy, "$24") == 0 || strcmp(tempy, "$t8") == 0 )   strncpy(pszOut, dTob(24)+11, 5);
    else if(strcmp(tempy, "$25") == 0 || strcmp(tempy, "$t9") == 0 )   strncpy(pszOut, dTob(25)+11, 5);
    else if(strcmp(tempy, "$26") == 0 || strcmp(tempy, "$k0") == 0 )   strncpy(pszOut, dTob(26)+11, 5);
    else if(strcmp(tempy, "$27") == 0 || strcmp(tempy, "$k1") == 0 )   strncpy(pszOut, dTob(27)+11, 5);
    else if(strcmp(tempy, "$28") == 0 || strcmp(tempy, "$gp") == 0 )   strncpy(pszOut, dTob(28)+11, 5);
    else if(strcmp(tempy, "$29") == 0 || strcmp(tempy, "$sp") == 0 )   strncpy(pszOut, dTob(29)+11, 5);
    else if(strcmp(tempy, "$30") == 0 || strcmp(tempy, "$s8") == 0 )   strncpy(pszOut, dTob(30)+11, 5);
    else if(strcmp(tempy, "$31") == 0 || strcmp(tempy, "$ra") == 0 )   strncpy(pszOut, dTob(31)+11, 5);

    // Cap pszReg
    pszOut[5] = '\0';

    return 1;
}

void InitializeOPMap()
{
    // Memory Ops
    s_mapOP["lw"] = OP_LW;
    s_mapOP["lh"] = OP_LH;
    s_mapOP["lhu"] = OP_LHU;
    s_mapOP["lb"] = OP_LB;
    s_mapOP["lbu"] = OP_LBU;
    s_mapOP["sw"] = OP_SW;
    s_mapOP["sh"] = OP_SH;
    s_mapOP["sb"] = OP_SB;

    // Immediate Ips
    s_mapOP["lui"] = OP_LUI;
    s_mapOP["addi"] = OP_ADDI;
    s_mapOP["addiu"] = OP_ADDIU;
    s_mapOP["slti"] = OP_SLTI;
    s_mapOP["sltiu"] = OP_SLTIU;
    s_mapOP["andi"] = OP_ANDI;
    s_mapOP["xori"] = OP_XORI;
    s_mapOP["ori"] = OP_ORI;

    // Network OPS
    // **********************
    // CPU side
    s_mapOP["smsg"] = OP_SMSG;
    s_mapOP["rmsg"] = OP_RMSG;
    s_mapOP["sid"] = OP_SID;
    s_mapOP["bcst"] = OP_BCST;
    s_mapOP["smsgr"] = OP_SMSGR;
    s_mapOP["bcstr"] = OP_BCSTR;
    // Net Side
    s_mapOP["jalnet"] = OP_JALNET;
    s_mapOP["jalnet\n"] = OP_JALNET;
    s_mapOP["nack"] = OP_NACK;
    s_mapOP["snip"] = OP_SNIP;
    s_mapOP["ndjr"] = OP_NDJR;
    // **********************
```

```
    // R-Type Ops
    s_mapOP["add"] = OP_ADD;
    s_mapOP["addu"] = OP_ADDU;
    s_mapOP["sub"] = OP_SUB;
    s_mapOP["subu"] = OP_SUBU;
    s_mapOP["and"] = OP_AND;
    s_mapOP["or"] = OP_OR;
    s_mapOP["xor"] = OP_XOR;
    s_mapOP["slt"] = OP_SLT;
    s_mapOP["sltu"] = OP_SLTU;
    s_mapOP["sll"] = OP_SLL;
    s_mapOP["srl"] = OP_SRL;
    s_mapOP["sra"] = OP_SRA;

    // Branching
    s_mapOP["bne"] = OP_BNE;
    s_mapOP["beq"] = OP_BEQ;
    s_mapOP["bgtz"] = OP_BGTZ;
    s_mapOP["bgez"] = OP_BGEZ;
    s_mapOP["bltz"] = OP_BLTZ;
    s_mapOP["blez"] = OP_BLEZ;

    s_mapOP["j"] = OP_J;
    s_mapOP["jr"] = OP_JR;
    s_mapOP["jal"] = OP_JAL;

    // IN/OUT Instructions
    s_mapOP["in"] = OP_IN;
    s_mapOP["out"] = OP_OUT;
    s_mapOP["outi"] = OP_OUTI;

    // Mult Div Mod
    s_mapOP["mult"] = OP_MULT;
    s_mapOP["multu"] = OP_MULTU;
    s_mapOP["div"] = OP_DIV;
    s_mapOP["divu"] = OP_DIVU;
    s_mapOP["mod"] = OP_MOD;

    // Pseudo Instructions
    s_mapOP["la"] = OP_LA;
    s_mapOP["sri"] = OP_SRI;

    // Control Ops
    // repeat with terminator for convinience on single argument ops
    s_mapOP["nop"]  = OP_NOP;
    s_mapOP["nop\n"] = OP_NOP;
    s_mapOP["break"] = OP_BREAK;
    s_mapOP["break\n"] = OP_BREAK;
    s_mapOP["quit"] = OP_QUIT;
    s_mapOP["quit\n"] = OP_QUIT;
}

int OpFunctionSADecode(char *AsmOp, char* OpCode, char* FunctionCode, char *SACode)
{
    int ins_type = NOT_TYPE;
    if(AsmOp == NULL)
    {
        return NOT_TYPE;
    }

    switch(s_mapOP[AsmOp])
            {
                // Memory Ops
                case OP_LW:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "100011");
                        ins_type = I_TYPE;
                        break;
                    }

                case OP_LH:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "100001");
                        ins_type = I_TYPE;
                        break;
                    }

                case OP_LHU:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "100101");
                        ins_type = I_TYPE;
                        break;
```

```
        }

    case OP_LB:
        {
            if(OpCode != NULL) strcpy(OpCode, "100000");
            ins_type = I_TYPE;
            break;
        }

    case OP_LBU:
        {
            if(OpCode != NULL) strcpy(OpCode, "100100");
            ins_type = I_TYPE;
            break;
        }

    case OP_SW:
        {
            if(OpCode != NULL) strcpy(OpCode, "101011");
            ins_type = I_TYPE;
            break;
        }

    case OP_SH:
        {
            if(OpCode != NULL) strcpy(OpCode, "101001");
            ins_type = I_TYPE;
            break;
        }

    case OP_SB:
        {
            if(OpCode != NULL) strcpy(OpCode, "101000");
            ins_type = I_TYPE;
            break;
        }

    // Immediate Ops
    case OP_LUI:
        {
            if(OpCode != NULL) strcpy(OpCode, "001111");
            ins_type = I_TYPE;
            break;
        }

    case OP_ADDI:
        {
            if(OpCode != NULL) strcpy(OpCode, "001000");
            ins_type = I_TYPE;
            break;
        }

    case OP_ADDIU:
        {
            if(OpCode != NULL) strcpy(OpCode, "001001");
            ins_type = I_TYPE;
            break;
        }

    case OP_SLTI:
        {
            if(OpCode != NULL) strcpy(OpCode, "001010");
            ins_type = I_TYPE;
            break;
        }

    case OP_SLTIU:
        {
            if(OpCode != NULL) strcpy(OpCode, "001011");
            ins_type = I_TYPE;
            break;
        }

    case OP_ANDI:
        {
            if(OpCode != NULL) strcpy(OpCode, "001100");
            ins_type = I_TYPE;
            break;
        }

    case OP_XORI:
        {
```

```
            if(OpCode != NULL) strcpy(OpCode, "001110");
            ins_type = I_TYPE;
            break;
        }

    case OP_ORI:
        {
            if(OpCode != NULL) strcpy(OpCode, "001101");
            ins_type = I_TYPE;
            break;
        }

    // ***** Pseudo Instructions *****
    case OP_LA:
        {
            // LA is just another encoding for <-- needs to be addi
            if(OpCode != NULL) strcpy(OpCode, "001000");
            ins_type = P_TYPE;
            break;
        }

    case OP_SRI:
        {
            // LA is just another encoding for <-- needs to be addi
            //if(OpCode != NULL) strcpy(OpCode, "001000");
            ins_type = P_TYPE;
            break;
        }

    // ***** branching instructions ****
    case OP_BNE:
        {
            if(OpCode != NULL) strcpy(OpCode, "000101");
            ins_type = I_TYPE;
            break;
        }

    case OP_BEQ:
        {
            if(OpCode != NULL) strcpy(OpCode, "000100");
            ins_type = I_TYPE;
            break;
        }

    case OP_BGTZ:
        {
            if(OpCode != NULL) strcpy(OpCode, "000111");
            ins_type = I_TYPE;
            break;
        }

    case OP_BGEZ:
        {
            if(OpCode != NULL) strcpy(OpCode, "010101");
            ins_type = I_TYPE;
            break;
        }

    case OP_BLTZ:
        {
            if(OpCode != NULL) strcpy(OpCode, "010100");
            ins_type = I_TYPE;
            break;
        }

    case OP_BLEZ:
        {
            if(OpCode != NULL) strcpy(OpCode, "000110");
            ins_type = I_TYPE;
            break;
        }
    // ***** IN OUT INSTRUCTIONS *****
    case OP_OUTI:
        {
            if(OpCode != NULL) strcpy(OpCode, "010110");
            ins_type = I_TYPE;
            break;
        }

    case OP_IN:
        {
            if(OpCode != NULL) strcpy(OpCode, "000000");
```

```
            if(FunctionCode != NULL) strcpy(FunctionCode, "110000");
            if(SACode != NULL) strcpy(SACode, "00000");
            ins_type = R_TYPE;
            break;
        }

    case OP_OUT:
        {
            if(OpCode != NULL) strcpy(OpCode, "000000");
            if(FunctionCode != NULL) strcpy(FunctionCode, "110001");
            if(SACode != NULL) strcpy(SACode, "00000");
            ins_type = R_TYPE;
            break;
        }

    // ***** Network Instructions ****
    // CPU Side
    case OP_SMSG:
        {
            if(OpCode != NULL) strcpy(OpCode, "110000");
            ins_type = I_TYPE;
            break;
        }

    case OP_RMSG:
        {
            if(OpCode != NULL) strcpy(OpCode, "110001");
            ins_type = I_TYPE;
            break;
        }

    case OP_SID:
        {
            if(OpCode != NULL) strcpy(OpCode, "110010");
            ins_type = J_TYPE;
            break;
        }

    case OP_BCST:
        {
            if(OpCode != NULL) strcpy(OpCode, "110011");
            ins_type = J_TYPE;
            break;
        }

    // register based bcst and smsg
    case OP_SMSGR:
        {
            if(OpCode != NULL) strcpy(OpCode, "111000");
            ins_type = I_TYPE;
            break;
        }

    case OP_BCSTR:
        {
            if(OpCode != NULL) strcpy(OpCode, "111001");
            ins_type = I_TYPE;
            break;
        }

    // NET SIDE
    case OP_SNIP:
        {
            if(OpCode != NULL) strcpy(OpCode, "110110");
            ins_type = NET_TYPE;
            break;
        }

    case OP_JALNET:
        {
            if(OpCode != NULL) strcpy(OpCode, "110100");
            ins_type = NET_TYPE;
            break;
        }

    case OP_NDJR:
        {
            if(OpCode != NULL) strcpy(OpCode, "110111");
            ins_type = NET_TYPE;
            break;
        }
```

```
            // ********* R_TYPE ********************
            case OP_ADD:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100000");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_ADDU:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100001");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_SUB:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100010");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_SUBU:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100011");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_AND:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100100");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_OR:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100101");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_XOR:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "100110");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_SLT:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "101010");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }

            case OP_SLTU:
                {
                    if(OpCode != NULL) strcpy(OpCode, "000000");
                    if(FunctionCode != NULL) strcpy(FunctionCode, "101011");
                    if(SACode != NULL) strcpy(SACode, "00000");
                    ins_type = R_TYPE;
                    break;
                }
```

```
                case OP_SLL:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "000000");
                        //if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = SR_TYPE;
                        break;
                    }

                case OP_SRL:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "000010");
                        //if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = SR_TYPE;
                        break;
                    }

                case OP_SRA:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "000011");
                        //if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = SR_TYPE;
                        break;
                    }

                // Mult Div Mod
                case OP_MULT:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "011000");
                        if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = R_TYPE;
                        break;
                    }

                case OP_MULTU:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "011001");
                        if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = R_TYPE;
                        break;
                    }

                case OP_DIV:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "011010");
                        if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = R_TYPE;
                        break;
                    }

                case OP_DIVU:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "011011");
                        if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = R_TYPE;
                        break;
                    }

                case OP_MOD:
                    {
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        if(FunctionCode != NULL) strcpy(FunctionCode, "110010");
                        if(SACode != NULL) strcpy(SACode, "00000");
                        ins_type = R_TYPE;
                        break;
                    }

                default:
                    {
                        printf("Error!: %s not recognized", AsmOp);
                        if(OpCode != NULL) strcpy(OpCode, "000000");
                        break;
                    }
            } // end switch(s_mapOP[arg[0]])

    return ins_type;
```

```
}
```

```
// This is a (hopefully) simple bootloader application
// which takes an input as a .mem file from the NDMA
// assembler and then sends it out to RS-232 port to
// be received by the NDMA CPU's boot loader module

// Usage:
// NDMABootLoad COM# filename.mem
//  # is the COM port number (such as COM4)
//  filename.mem is the mem file to boot load from the NDMA assembler

#include <windows.h>
#include <AtlBase.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

#define DELAY_MS 15

// Declerations
HRESULT InitPort(HANDLE *phComPort, char *pszPortName);
BOOL TransmitInstruction(HANDLE hComPort, unsigned long int PC, unsigned long int Ins);
BOOL EndTransmit(HANDLE hComPort);

int main(int argc, char *argv[])
{
    int filelength = 0;
    time_t timeStart;
    time_t timeEnd;

    if(argc < 3 || strlen(argv[1]) == 0 || strlen(argv[2]) == 0)
    {
        printf("Usage: NDMABootLoad COM# filename.mem\n");
        printf("# - Com Port Number\n");
        printf("filename.mem - memfile to boot load from the NDMA assembler\n");
        system("PAUSE");
        return 0;
    }

    FILE *fileInput;
    fileInput = fopen(argv[2], "r");

    if(fileInput == NULL)
    {
        printf("Error Opening File: %s\n", argv[2]);
        perror("Error");
        system("PAUSE");
        return 0;
    }



    HANDLE hComPort = NULL;

    // Transmitting
    DWORD dwBytesWritten;
    DWORD dwBytesToWrite;
    char szWriteString[40];

    // Receiving
    DWORD dwBytesRead;
    char szReadString[40];
    DWORD dwEventMask = NULL;

    // PC
    unsigned long int PC = 0;
    // Ins
    unsigned long int Ins;
    // Ins In
    char input[100];
    BOOL fDone = FALSE;


    if(InitPort(&hComPort, argv[1]) != S_OK)
    {
        printf("Failed to Initialize Port %s\n", argv[1]);
        system("PAUSE");
        return E_FAIL;
    }

    // Send Boot Command
    strcpy(szWriteString, "boot");
```

```
    dwBytesToWrite = strlen(szWriteString);

    if(!WriteFile(hComPort, szWriteString, dwBytesToWrite, &dwBytesWritten, NULL))
    {
        printf("Failed to write %s to Com Port\n", szWriteString);
    }

    // Give the data a millasecond to arrive
    Sleep(DELAY_MS*5);

    // Wait for boot ready command
    while(dwEventMask == NULL)
    {
        WaitCommEvent(hComPort, &dwEventMask, NULL);
    }

    memset(szReadString, 0, sizeof(szReadString));
    if(!(::ReadFile(hComPort, szReadString, sizeof(szReadString), &dwBytesRead, NULL)))
    {
        printf("Failed to read Com Port\n");
    }

    if(strcmp(szReadString, "ready") == 0)
    {
        printf("Boot Loader Ready...\n");
    }
    else
    {
        printf("Boot Loader Cannot Initialize!\n");
        printf("Please Reset Machine\n");
        fclose(fileInput);
        fDone = TRUE;
        system("PAUSE");
        return 0;
    }
    // Give it a moment to recover
    Sleep(DELAY_MS*5);

    // FILE LENGTH
    // ******************************
    filelength = 0;
    while(!feof(fileInput))
    {
        fgets(input, 100, fileInput);
        filelength++;
    }

    printf("%d lines in input file %s\n", filelength, argv[2]);
    rewind(fileInput);

    // PARSE FILE
    // ******************************
    PC = 0;
    timeStart = time(NULL);
    while(fDone != TRUE)
    {
        fgets(input, 100, fileInput);
        if(feof(fileInput))
        {
            fDone = TRUE;
        }

        Ins = 0;

        for(int i = 0; i < 32; i++)
        {
            if(input[i] == '1')
            {
                Ins += (double)pow((double)2, (int)(31 - i));
            }
        }

        if(!TransmitInstruction(hComPort, PC, Ins))
        {
            printf("BootLoader Error!\n");
            printf("Please Reset Machine\n");
            fclose(fileInput);
            fDone = TRUE;
            system("PAUSE");
            return 0;
        }
        else
```

```
        {
            timeEnd = time(NULL);
            printf("PC: %d Current Progress: %2.0f%% time: %.1f seconds EST: %.1f\r",
                PC, 100.0*((float)(PC)/(float)(filelength)), difftime(timeEnd, timeStart),
                ((float)(filelength)*(float)difftime(timeEnd, timeStart))/(float)(PC) - (float)difftime(timeEnd,
                    timeStart));
        }
        PC++;
    }

    (void)EndTransmit(hComPort);

    fclose(fileInput);
    //system("PAUSE");

    return 0;
}

BOOL TransmitInstruction(HANDLE hComPort, unsigned long int PC, unsigned long int Ins)
{
    // Transmitting
    DWORD dwBytesWritten;
    DWORD dwBytesToWrite;
    char szWriteString[40];

    // Receiving
    DWORD dwBytesRead;
    char szReadString[40];
    DWORD dwEventMask = NULL;

    // PC
    unsigned long int cPC = 0;
    unsigned long int rPC;

    // Ins
    unsigned long int sIns;
    unsigned long int rIns;

    // Dispatch PC
    cPC = PC;
    memset(szWriteString, 0, sizeof(cPC));
    for(int i = 0; i < 4; i++)
    {
        //long int andor = 255 << 8*i;
        char result = (cPC >> 8*i) & 255;
        szWriteString[i] = result;
    }
    dwBytesToWrite = sizeof(cPC);
    if(!WriteFile(hComPort, szWriteString, dwBytesToWrite, &dwBytesWritten, NULL))
    {
        printf("Failed to write %s to Com Port\n", szWriteString);
        return FALSE;
    }

    // Receive back PC
    dwEventMask = NULL;
    while(dwEventMask == NULL)
    {
        WaitCommEvent(hComPort, &dwEventMask, NULL);
    }

    // Give the data a millasecond to arrive
    Sleep(DELAY_MS);

    memset(szReadString, 0, sizeof(szReadString));
    if(!(::ReadFile(hComPort, szReadString, sizeof(szReadString), &dwBytesRead, NULL)))
    {
        printf("Failed to read Com Port\n");
        return FALSE;
    }

    rPC = 0;
    // Decode PC Out
    for(int i = 0; i < 4; i++)
    {
        rPC += ((unsigned long int)(unsigned char)szReadString[i] << 8*i);
    }
    // Compare PC values
    if(rPC == cPC)
    {
        //printf("Matched PC:%u\n", rPC);
    }
```

```
    else
    {
        printf("Mismatch on PC out:%u in:%u\n", cPC, rPC);
        return FALSE;
    }
    // Wait a ms
    Sleep(DELAY_MS);

    // Dispatch Instruction
    sIns = Ins;
    memset(szWriteString, 0, sizeof(sIns));
    for(int i = 0; i < 4; i++)
    {
        //long int andor = 255 << 8*i;
        char result = (sIns >> 8*i) & 255;
        szWriteString[i] = result;
    }
    dwBytesToWrite = sizeof(sIns);
    if(!WriteFile(hComPort, szWriteString, dwBytesToWrite, &dwBytesWritten, NULL))
    {
        printf("Failed to write %s to Com Port\n", szWriteString);
        return FALSE;
    }

    // Receive back Instruction
    dwEventMask = NULL;
    while(dwEventMask == NULL)
    {
        WaitCommEvent(hComPort, &dwEventMask, NULL);
    }

    // Give the data a millasecond to arrive
    Sleep(DELAY_MS);

    memset(szReadString, 0, sizeof(szReadString));
    if(!(::ReadFile(hComPort, szReadString, sizeof(szReadString), &dwBytesRead, NULL)))
    {
        printf("Failed to read Com Port\n");
        return FALSE;
    }

    rIns = 0;
    // Decode Instruction Out
    for(int i = 0; i < 4; i++)
    {
        rIns += ((unsigned long int)(unsigned char)szReadString[i] << 8*i);
    }
    // Compare Instruction values
    if(rIns == sIns)
    {
        //printf("Matched Ins:%x\n", rIns);
    }
    else
    {
        printf("Mismatch Ins out:%x in:%x\n", sIns, rIns);
        return FALSE;
    }
    // Wait a moment
    Sleep(DELAY_MS);

    return TRUE;
}

BOOL EndTransmit(HANDLE hComPort)
{
    // Transmitting
    DWORD dwBytesWritten;
    DWORD dwBytesToWrite;
    char szWriteString[40];

    // Receiving
    DWORD dwBytesRead;
    char szReadString[40];
    DWORD dwEventMask = NULL;

    // PC
    unsigned long int cPC = 0;
    unsigned long int rPC;

    // Send end transmit
    // Dispatch PC
    cPC = 0xffffffff;
```

```
    memset(szWriteString, 0, sizeof(cPC));
    for(int i = 0; i < 4; i++)
    {
        //long int andor = 255 << 8*i;
        char result = (cPC >> 8*i) & 255;
        szWriteString[i] = result;
    }
    dwBytesToWrite = sizeof(cPC);
    if(!WriteFile(hComPort, szWriteString, dwBytesToWrite, &dwBytesWritten, NULL))
    {
        printf("Failed to write %s to Com Port\n", szWriteString);
        return FALSE;
    }
    // Receive back PC
    dwEventMask = NULL;
    while(dwEventMask == NULL)
    {
        WaitCommEvent(hComPort, &dwEventMask, NULL);
    }

    // Give the data a millasecond to arrive
    Sleep(DELAY_MS);

    memset(szReadString, 0, sizeof(szReadString));
    if(!(::ReadFile(hComPort, szReadString, sizeof(szReadString), &dwBytesRead, NULL)))
    {
        printf("Failed to read Com Port\n");
        return FALSE;
    }

    rPC = 0;
    // Decode PC Out
    for(int i = 0; i < 4; i++)
    {
        rPC += ((unsigned long int)(unsigned char)szReadString[i] << 8*i);
    }
    // Compare PC values
    if(rPC == cPC)
    {
        printf("Matched End Transmit PC:%x\n", rPC);
    }
    else
    {
        printf("Mismatch on End Transmit PC out:%u in:%u\n", cPC, rPC);
        return FALSE;
    }
    // Wait a ms
    Sleep(DELAY_MS);
    return TRUE;
}

HRESULT InitPort(HANDLE *phComPort, char *pszPortName)
{
    OLECHAR* oleChar = NULL;
    oleChar = (OLECHAR*)calloc(strlen(pszPortName)+1, sizeof(OLECHAR));
    MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED, pszPortName, -1, oleChar, strlen(pszPortName)+1);
    BSTR bstrPortName = SysAllocString(oleChar);
    free(oleChar);
    oleChar = NULL;

    *phComPort = ::CreateFile((LPCWSTR)bstrPortName,
                              GENERIC_READ | GENERIC_WRITE,     // access (rw)
                              0,                                // share (no sharing)
                              0,                                // Security
                              OPEN_EXISTING,                    // Creation
                              0,                                // Overlapped Operation
                              0                                 // No Template
                              );

    if(*phComPort == NULL || *phComPort == INVALID_HANDLE_VALUE)
    {
        printf("Could not open %s port, error:%d\n", pszPortName, GetLastError());
        if(GetLastError() == ERROR_ACCESS_DENIED)
        {
            printf("Port %s Access Denied\n", pszPortName);
            return E_FAIL;
        }
        else if(GetLastError() == ERROR_FILE_NOT_FOUND)
        {
            printf("Port %s Does Not Exist\n", pszPortName);
            return E_FAIL;
        }
```

```
    }

    // Set Up Com Port State with Device Control Block
    DCB dcb = {0};
    dcb.DCBlength = sizeof(DCB);
    if(!(::GetCommState(*phComPort, &dcb)))
    {
        printf("Failed to get Com Device State Error: %d\n", GetLastError());
        return E_FAIL;
    }

    dcb.BaudRate = CBR_115200;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = ONESTOPBIT;

    if(!(::SetCommState(*phComPort, &dcb)))
    {
        printf("Failed to set Com Device State Error: %d\n", GetLastError());
        return E_FAIL;
    }

    // Register for Receive Events
    SetCommMask(*phComPort, EV_RXCHAR);

    // Print out settings
    printf("port %s connected\nBaud Rate: %d\nByte Size: %d\nStop Bits:%d\n",
            pszPortName, dcb.BaudRate, dcb.ByteSize, dcb.StopBits);

    return S_OK;
}
```

```
// NDMA Merge is a simple program that can be used to fuse together multiple assembler
// files to be used with the NDMA assembler.  This is mostly useful for the assembly
// based APIs which otherwise would need to be stiched in manually.

#include <windows.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    if(argc == 1 || argc > 3)
    {
        printf("Usage: NDMAMerge compiler_output.asm assembler_api.asm\n");
        printf("This program will insert the assembler_api.asm code inbetween\n");
        printf("the .Boot directive and the rest of the compiler output\n");
        goto Error;
    }
    else
    {
        printf("NDMA Merge: %s with %s\n", argv[1], argv[2]);
    }

    FILE *fileA;
    FILE *fileB;
    FILE *fileOutput;
    char pszOutput[15];
    char input[100];
    int filechooser = 0;
    int fDone = 0;

    strcpy(pszOutput, "ndmamerge.asm");

    fileA = fopen(argv[1], "r");
    if(fileA == NULL)
    {
        printf("Error Opening: %s\n", argv[1]);
        goto Error;
    }

    fileB = fopen(argv[2], "r");
    if(fileB == NULL)
    {
        printf("Error Opening: %s\n", argv[2]);
        goto Error;
    }

    fileOutput = fopen(pszOutput, "w");
    if(fileOutput == NULL)
    {
        printf("Error Opening Output File: %s", pszOutput);
        goto Error;
    }

    while(!fDone)
    {
        if(filechooser == 0)
        {
            fgets(input, 100, fileA);
            fprintf(fileOutput, "%s", input);
            if(feof(fileA))
            {
                fDone = 1;
                fprintf(fileOutput, "\n");
            }

        }
        else if(filechooser == 1)
        {
            fgets(input, 100, fileB);
            fprintf(fileOutput, "%s", input);
            if(feof(fileB))
            {
                filechooser = 0;
                fprintf(fileOutput, "\n");
            }

        }

        char *tempx = NULL;
        tempx = strtok(input, " ,()\t");

        if(strcmp(tempx, ".Boot") == 0)
        {
```

```
            printf("Boot Directive Found!\n");
            filechooser = 1;
        }
    }

    //system("PAUSE");
    return 0;

Error:
    //system("PAUSE");
    return 0;
}
```

```
@ECHO OFF

REM Usage: ndmamake file.c api.asm COM#
REM This will compile file.c into a asm file with the RCC compiler and then
REM splice api.asm into the file using NDMA Merge.  Then the NDMA Bootloader
REM will bootload the memory file that is output from the NDMA Assembler on the
REM port designated in the third argument written as COM# (e.g. COM4 for COM4)
REM Example:
REM ndmamake filename.c api.asm COM3 \r

REM Check for first argument
REM ################################################
IF NOT EXIST %CD%\%1 GOTO NOTFOUND
IF NOT EXIST %CD%\%2 GOTO NOTFOUNDAPI
REM ################################################

REM Compile the ANSI C File
REM ################################################
rcc -target=ndma/ndmaOS %1 > rcc_code.asm
rcc -target=ndma/ndmaOS %1
IF NOT EXIST %CD%\rcc_code.asm GOTO NOTCREATEDASM
ECHO rcc_code.asm created successfully
REM ################################################

REM Merge RCC asm and API asm
REM ################################################
NDMAMerge rcc_code.asm %2
IF NOT EXIST %CD%\ndmamerge.asm GOTO NOTCREATEDMERGE
ECHO ndmamerge.asm created successfully
REM ################################################

REM Assembler / Link the ndmamerge.asm
REM ################################################
NDMAAssembler ndmamerge.asm
IF NOT EXIST %CD%\ndmamerge.mem GOTO NOTCREATEDMEM
ECHO ndmamerge.mem compiled successfully
REM ################################################

REM Bootload ndmamerge.mem
REM ################################################
NDMABootloader %3 ndmamerge.mem
REM ################################################

ECHO NDMA Make done!
GOTO END

REM Error Cases

:NOTFOUND
ECHO %1 not found!
GOTO END

:NOTCREATEDASM
ECHO rcc failure!
GOTO END

:NOTFOUNDAPI
ECHO %2 not found!
GOTO END

:NOTCREATEDMERGE
ECHO NDMA Merge Failure!
GOTO END

:NOTCREATEDMEM
ECHO NDMA Assembler Failure!
GOTO END

:END
```

Visual Nodes C/C++ Code

```
// visualNodes.cpp : Defines the entry point for the application.
//

#include "stdafx.h"
#include "visualNodes.h"


#define MAX_LOADSTRING 100
#define CPU_NODE_DIM 20
#define DELAY_COUNT 10


// Global Variables:
HGLRC           hRC=NULL;                       // Permanent Rendering Context
HDC             hDC=NULL;                       // Private GDI Device Context
HWND            hWnd=NULL;                      // Holds Our Window Handle
HINSTANCE       hInstance;                          // current instance

bool    keys[256];                              // Array Used For The Keyboard Routine
bool    active=TRUE;                                // Window Active Flag Set To TRUE By Default
bool    fullscreen=false;                           // Fullscreen Flag Set To Fullscreen Mode By Default
bool    npress=false;
bool    f2press=false;
bool    autostep=false;
int     delaycounter=0;



cpuNet *myNet;

float rotateAngle = 0;

TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

// Opengl Functions



// Resizes the window
GLvoid ReSizeGLScene(GLsizei width, GLsizei height)             // Resize And Initialize The GL Window
{
    if (height==0)                                  // Prevent A Divide By Zero By
    {
        height=1;                           // Making Height Equal One
    }

    glViewport(0, 0, width, height);                    // Reset The Current Viewport

    glMatrixMode(GL_PROJECTION);                        // Select The Projection Matrix
    glLoadIdentity();                               // Reset The Projection Matrix

    // Calculate The Aspect Ratio Of The Window
    gluPerspective(45.0f,(GLfloat)width/(GLfloat)height,0.1f,300.0f);

    glMatrixMode(GL_MODELVIEW);                         // Select The Modelview Matrix
    glLoadIdentity();                               // Reset The Modelview Matrix
}


// Opengl Initialize
int InitGL(GLvoid)                                  // All Setup For OpenGL Goes Here
{
    glShadeModel(GL_SMOOTH);                            // Enables Smooth Shading
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);                   // Black Background
    glClearDepth(1.0f);                             // Depth Buffer Setup
    glEnable(GL_DEPTH_TEST);                            // Enables Depth Testing
    glDepthFunc(GL_LEQUAL);                             // The Type Of Depth Test To Do
    glHint(GL_PERSPECTIVE_CORRECTION_HINT , GL_NICEST);         // Really Nice Perspective Calculations

    myNet = new cpuNet(CPU_NODE_DIM);
    //myNet->getNode(564)->sendMsg(23, new Color_3(.75, 0, 0));
    //myNet->getNode(277)->sendMsg(200, new Color_3(0, .75, 0));
    //myNet->getNode(403)->sendMsg(738, new Color_3(0, 0, .75));
    myNet->getNode(40)->sendMsg(5, new Color_3(0, .75, .0));
    //myNet->getNode(1561)->sendMsg(821, new Color_3(.75, 0, .75));
```

```
    myNet->getNode(250)->sendMsg(55, new Color_3(0, .75, .75));

    //myNet->removeNode(210);
    //myNet->removeNode(211);
    //myNet->removeNode(212);
    //myNet->removeNode(213);
    //myNet->getNode(350)->sendMsg(25);


    return TRUE;                                // Initialization Went OK
}


// Do our drawing!
int DrawGLScene(GLvoid)                             // Here's Where We Do All The Drawing
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);         // Clear The Screen And The Depth Buffer
    glLoadIdentity();   // Reset The Current Modelview Matrix

    glTranslatef(0.0f ,0.0f, -6*myNet->cpuDim);

    rotateAngle += .1;
    if(rotateAngle > 360) rotateAngle = 0;

    glPushMatrix();

    myNet->renderNet();
    glPopMatrix();
    return TRUE;                                // Everything Went OK
}


// Destroy Window on Exit
GLvoid KillGLWindow(GLvoid)                             // Properly Kill The Window
{
    if (fullscreen)                             // Are We In Fullscreen Mode?
    {
        ChangeDisplaySettings(NULL,0);                  // If So Switch Back To The Desktop
        ShowCursor(TRUE);                       // Show Mouse Pointer
    }
    if (hRC)                                // Do We Have A Rendering Context?
    {

        if (!wglMakeCurrent(NULL,NULL))                 // Are We Able To Release The DC And RC Contexts?
        {

            MessageBox(NULL,_T("Release Of DC And RC Failed."),_T("SHUTDOWN ERROR"),MB_OK | MB_ICONINFORMATION);
        }

        if (!wglDeleteContext(hRC))                 // Are We Able To Delete The RC?
        {

            MessageBox(NULL,_T("Release Rendering Context Failed."),_T("SHUTDOWN ERROR"),MB_OK |
                MB_ICONINFORMATION);
        }
        hRC=NULL;                               // Set RC To NULL
    }
    if (hDC && !ReleaseDC(hWnd,hDC))                    // Are We Able To Release The DC
    {
        //MessageBox(NULL,_T("Release Device Context Failed."),_T("SHUTDOWN ERROR"),MB_OK | MB_ICONINFORMATION);
        hDC=NULL;                               // Set DC To NULL
    }
    if (hWnd && !DestroyWindow(hWnd))                   // Are We Able To Destroy The Window?
    {
        //MessageBox(NULL,_T("Could Not Release hWnd."),_T("SHUTDOWN ERROR"),MB_OK | MB_ICONINFORMATION);
        hWnd=NULL;                              // Set hWnd To NULL
    }
    if (!UnregisterClass((LPCWSTR)"OpenGL",hInstance))              // Are We Able To Unregister Class
    {
        //MessageBox(NULL,_T("Could Not Unregister Class."),_T("SHUTDOWN ERROR"),MB_OK | MB_ICONINFORMATION);
        hInstance=NULL;                         // Set hInstance To NULL
    }
}


// Create window on enter
BOOL CreateGLWindow(LPCTSTR title, int width, int height, int bits, bool fullscreenflag)
{
    GLuint      PixelFormat;                        // Holds The Results After Searching For A Match
    WNDCLASS    wc;                         // Windows Class Structure
    DWORD       dwExStyle;                      // Window Extended Style
    DWORD       dwStyle;                        // Window Style
```

```
    RECT WindowRect;                                // Grabs Rectangle Upper Left / Lower Right Values
    WindowRect.left=(long)0;                            // Set Left Value To 0
    WindowRect.right=(long)width;                           // Set Right Value To Requested Width
    WindowRect.top=(long)0;                             // Set Top Value To 0
    WindowRect.bottom=(long)height;                         // Set Bottom Value To Requested
    fullscreen=fullscreenflag;                          // Set The Global Fullscreen Flag
    hInstance               = GetModuleHandle(NULL);                // Grab An Instance For Our Window
    wc.style            = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;   // Redraw On Move, And Own DC For Window
    wc.lpfnWndProc      = (WNDPROC) WndProc;                    // WndProc Handles Messages
    wc.cbClsExtra       = 0;                                    // No Extra Window Data
    wc.cbWndExtra       = 0;                                    // No Extra Window Data
    wc.hInstance        = hInstance;                            // Set The Instance
    wc.hIcon            = LoadIcon(NULL, IDI_WINLOGO);          // Load The Default Icon
    wc.hCursor          = LoadCursor(NULL, IDC_ARROW);          // Load The Arrow Pointer
    wc.hbrBackground    = NULL;                                 // No Background Required For GL
    wc.lpszMenuName     = NULL;                                 // We Don't Want A Menu
    wc.lpszClassName    = _T("OpenGL");                             // Set The Class Name
    if (!RegisterClass(&wc))                        // Attempt To Register The Window Class
    {
        MessageBox(NULL,_T("Failed To Register The Window Class."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               // Exit And Return FALSE
    }
    if (fullscreen)                                 // Attempt Fullscreen Mode?
    {
        DEVMODE dmScreenSettings;                       // Device Mode
        memset(&dmScreenSettings,0,sizeof(dmScreenSettings));       // Makes Sure Memory's Cleared
        dmScreenSettings.dmSize=sizeof(dmScreenSettings);       // Size Of The Devmode Structure
        dmScreenSettings.dmPelsWidth    = width;            // Selected Screen Width
        dmScreenSettings.dmPelsHeight   = height;           // Selected Screen Height
        dmScreenSettings.dmBitsPerPel   = bits;             // Selected Bits Per Pixel
        dmScreenSettings.dmFields=DM_BITSPERPEL|DM_PELSWIDTH|DM_PELSHEIGHT;
        // Try To Set Selected Mode And Get Results.  NOTE: CDS_FULLSCREEN Gets Rid Of Start Bar.
        if (ChangeDisplaySettings(&dmScreenSettings,CDS_FULLSCREEN)!=DISP_CHANGE_SUCCESSFUL)
        {

            // If The Mode Fails, Offer Two Options.  Quit Or Run In A Window.
            if (MessageBox(NULL,(LPCWSTR)_T("The Requested Fullscreen Mode Is Not Supported By\nYour Video Card.
                Use Windowed Mode Instead?"),_T("Open GL"),MB_YESNO|MB_ICONEXCLAMATION)==IDYES)
            {

                fullscreen=FALSE;               // Select Windowed Mode (Fullscreen=FALSE)
            }
            else
            {
                // Pop Up A Message Box Letting User Know The Program Is Closing.
                MessageBox(NULL,_T("Program Will Now Close."),_T("ERROR"),MB_OK|MB_ICONSTOP);
                return FALSE;                   // Exit And Return FALSE
            }
        }
    }
    if (fullscreen)                                 // Are We Still In Fullscreen Mode?
    {
        dwExStyle=WS_EX_APPWINDOW;                      // Window Extended Style
        dwStyle=WS_POPUP;                           // Windows Style
        ShowCursor(FALSE);                          // Hide Mouse Pointer
    }
    else
    {
        dwExStyle=WS_EX_APPWINDOW | WS_EX_WINDOWEDGE;           // Window Extended Style
        dwStyle=WS_OVERLAPPEDWINDOW;                    // Windows Style
    }
    AdjustWindowRectEx(&WindowRect, dwStyle, FALSE, dwExStyle);     // Adjust Window To True Requested Size
    if (!(hWnd=CreateWindowEx(  dwExStyle,              // Extended Style For The Window
                    _T("OpenGL"),               // Class Name
                    title,                  // Window Title
                    WS_CLIPSIBLINGS |           // Required Window Style
                    WS_CLIPCHILDREN |           // Required Window Style
                    dwStyle,                // Selected Window Style
                    0, 0,                   // Window Position
                    WindowRect.right-WindowRect.left,   // Calculate Adjusted Window Width
                    WindowRect.bottom-WindowRect.top,   // Calculate Adjusted Window Height
                    NULL,                   // No Parent Window
                    NULL,                   // No Menu
                    hInstance,              // Instance
                    NULL)))                 // Don't Pass Anything To WM_CREATE
    {
        KillGLWindow();                             // Reset The Display
        //MessageBox(NULL,"Window Creation Error.","ERROR",MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                               // Return FALSE
    }

    static  PIXELFORMATDESCRIPTOR pfd=                  // pfd Tells Windows How We Want Things To Be
```

```
    {
        sizeof(PIXELFORMATDESCRIPTOR),                  // Size Of This Pixel Format Descriptor
        1,                              // Version Number
        PFD_DRAW_TO_WINDOW |                        // Format Must Support Window
        PFD_SUPPORT_OPENGL |                        // Format Must Support OpenGL
        PFD_DOUBLEBUFFER,                       // Must Support Double Buffering
        PFD_TYPE_RGBA,                          // Request An RGBA Format
        bits,                               // Select Our Color Depth
        0, 0, 0, 0, 0, 0,                       // Color Bits Ignored
        0,                              // No Alpha Buffer
        0,                              // Shift Bit Ignored
        0,                              // No Accumulation Buffer
        0, 0, 0, 0,                             // Accumulation Bits Ignored
        16,                             // 16Bit Z-Buffer (Depth Buffer)
        0,                              // No Stencil Buffer
        0,                              // No Auxiliary Buffer
        PFD_MAIN_PLANE,                         // Main Drawing Layer
        0,                              // Reserved
        0, 0, 0                             // Layer Masks Ignored
    };

    if (!(hDC=GetDC(hWnd)))                         // Did We Get A Device Context?
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Can't Create A GL Device Context."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }

    if (!(PixelFormat=ChoosePixelFormat(hDC,&pfd)))             // Did Windows Find A Matching Pixel Format?
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Can't Find A Suitable PixelFormat."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }

    if(!SetPixelFormat(hDC,PixelFormat,&pfd))               // Are We Able To Set The Pixel Format?
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Can't Set The PixelFormat."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }

    if (!(hRC=wglCreateContext(hDC)))                   // Are We Able To Get A Rendering Context?
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Can't Create A GL Rendering Context."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }

    if(!wglMakeCurrent(hDC,hRC))                        // Try To Activate The Rendering Context
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Can't Activate The GL Rendering Context."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }

    ShowWindow(hWnd,SW_SHOW);                       // Show The Window
    SetForegroundWindow(hWnd);                      // Slightly Higher Priority
    SetFocus(hWnd);                             // Sets Keyboard Focus To The Window
    ReSizeGLScene(width, height);                       // Set Up Our Perspective GL Screen

    if (!InitGL())                              // Initialize Our Newly Created GL Window
    {
        KillGLWindow();                         // Reset The Display
        MessageBox(NULL,_T("Initialization Failed."),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
        return FALSE;                           // Return FALSE
    }
    return TRUE;                                // Success
}




int APIENTRY _tWinMain(HINSTANCE hInstance,
                       HINSTANCE hPrevInstance,
                       LPTSTR    lpCmdLine,
                       int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
```

```
    UNREFERENCED_PARAMETER(lpCmdLine);


    MSG msg;
    bool done = FALSE;
    HACCEL hAccelTable;


    /*
    // Ask The User Which Screen Mode They Prefer
    if (MessageBox(NULL,_T("Would You Like To Run In Fullscreen Mode?"), _T("Start FullScreen?"),MB_YESNO|
        MB_ICONQUESTION)==IDNO)
    {
        fullscreen=FALSE;                       // Windowed Mode
    }*/

    // Create Our OpenGL Window
    if (!CreateGLWindow(_T("OpenGL Framework"),640,480,16,fullscreen))
    {
        return 0;                               // Quit If Window Was Not Created
    }

    while(!done)                                    // Loop That Runs Until done=TRUE
    {
    if (PeekMessage(&msg,NULL,0,0,PM_REMOVE))           // Is There A Message Waiting?
        {
            if (msg.message==WM_QUIT)               // Have We Received A Quit Message?
            {
                done=TRUE;                  // If So done=TRUE
            }
            else                                // If Not, Deal With Window Messages
            {
                TranslateMessage(&msg);             // Translate The Message
                DispatchMessage(&msg);              // Dispatch The Message
            }
        }
        else                                    // If There Are No Messages
        {
            // Draw The Scene.  Watch For ESC Key And Quit Messages From DrawGLScene()
            if (active)                         // Program Active?
            {
                if (keys[VK_ESCAPE])                // Was ESC Pressed?
                {
                    done=TRUE;              // ESC Signalled A Quit
                }
                else                            // Not Time To Quit, Update Screen
                {
                    DrawGLScene();              // Draw The Scene
                    if(autostep == false){
                        if(!npress && keys['N']) {
                            npress = true;
                            myNet->updateNet();
                        }
                        if(!keys['N']) npress = false;
                    } else {
                        delaycounter++;

                        if( delaycounter > DELAY_COUNT )
                        {
                            delaycounter = 0;
                            myNet->updateNet();
                        }
                    }

                    SwapBuffers(hDC);           // Swap Buffers (Double Buffering)

                }
            }
            if (keys[VK_F1])                    // Is F1 Being Pressed?
            {
                keys[VK_F1]=FALSE;              // If So Make Key FALSE
                KillGLWindow();                 // Kill Our Current Window
                fullscreen=!fullscreen;             // Toggle Fullscreen / Windowed Mode
                // Recreate Our OpenGL Window
                if (!CreateGLWindow(_T("OpenGL Framework"),640,480,16,fullscreen))
                {
                    MessageBox(NULL,_T("Sheitz!"),_T("ERROR"),MB_OK|MB_ICONEXCLAMATION);
                    return 0;               // Quit If Window Was Not Created
                }
            }
            if (keys[VK_F2] && !f2press){
                f2press = true;
```

```
                autostep = !autostep;
            }
            if(!keys[VK_F2]) f2press = false;

        }
    }
    // Shutdown
    KillGLWindow();                                 // Kill The Window
    return (int)(msg.wParam);                       // Exit The Program
}



    /*
    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_VISUALNODES, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_VISUALNODES));

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
    */



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
//  COMMENTS:
//
//    This function and its usage are only necessary if you want this code
//    to be compatible with Win32 systems prior to the 'RegisterClassEx'
//    function that was added to Windows 95. It is important to call this function
//    so that the application will get 'well formed' small icons associated
//    with it.
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_VISUALNODES));
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCE(IDC_VISUALNODES);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
```

```
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInst, int nCmdShow)
{
   HWND hWnd;

   hInstance = hInst; // Store instance handle in our global variable

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE:  Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;


    switch (message)
    {
    case WM_COMMAND:
        wmId    = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
            case IDM_ABOUT:
                DialogBox(hInstance, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;

            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;

        }
        break;

    case WM_SYSCOMMAND:                     // Intercept System Commands
        {
            switch (wParam)                     // Check System Calls
            {
                case SC_SCREENSAVE:             // Screensaver Trying To Start?
                case SC_MONITORPOWER:               // Monitor Trying To Enter Powersave?
                return 0;                   // Prevent From Happening
            }
            break;                          // Exit
        }

    case WM_KEYDOWN:                        // Is A Key Being Held Down?
        {
            keys[wParam] = TRUE;                    // If So, Mark It As TRUE
            return 0;                       // Jump Back
        }

    case WM_KEYUP:                          // Has A Key Been Released?
        {
            keys[wParam] = FALSE;                   // If So, Mark It As FALSE
            return 0;                       // Jump Back
        }
```

```
    case WM_SIZE:                           // Resize The OpenGL Window
        {
            ReSizeGLScene(LOWORD(lParam),HIWORD(lParam));       // LoWord=Width, HiWord=Height
            return 0;                       // Jump Back
        }

    case WM_DESTROY:
        PostQuitMessage(0);
        break;


    }
    return DefWindowProc(hWnd, message, wParam, lParam);
}

// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

```
#define CPU_SPACING 4

class cpuNet {
private:

public:
    cpuNode *topLeft;
    cpuNode *topRight;
    cpuNode *bottomLeft;
    cpuNode *bottomRight;
    int nodeCount;
    int cpuDim;
    int maxDistance;
    int timeDelay;

    cpuNet () {};
    cpuNet (int);

    void printNet();
    void renderNet();
    void updateNet();

    cpuNode* getNode(int id);
    void removeNode(int id);

};
```

```
#include "stdafx.h"

cpuNet::cpuNet(int cpuDim){
    float leftX = -1*(cpuDim/2)*CPU_SPACING;
    float topY = (cpuDim/2)*CPU_SPACING;


    int i = 1;
    this->nodeCount = 0;
    this->cpuDim = cpuDim;

    this->topLeft = new cpuNode(i);

    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;


    //rowCur->setLoc(0, 0, 0);
    //printf("cur ID: %d ", colCur->getID());
    i++;
    this->nodeCount++;

    // Create original net
    // links left and right and up/down on the left edge

    for(int j=0;j < cpuDim;j++){
        rowCur->setLoc(Point_3(leftX, topY - (j)*CPU_SPACING, 0));
        // start at one due to the start of the row
        for(int k=1;k < cpuDim;k++){
            colCur->createRight(i);
            this->nodeCount++;
            i++;
            colCur = colCur->getRight();
            colCur->setLoc(Point_3(leftX + (k*CPU_SPACING), topY - (j)*CPU_SPACING, 0));
            //printf("cur ID: %d ", colCur->getID());
            // bottom right
            if(j == cpuDim - 1 && k == cpuDim - 1) this->bottomRight = colCur;
            // bottom left
            if(j == cpuDim - 1 && k == 1) this->bottomLeft = colCur;
            // top right
            if(j == 0 && k == 1) this->topRight = colCur;
        }

        //printf("\n");

        // not on last line
        if(j != cpuDim - 1) {
            rowCur->createDown(i);
            this->nodeCount++;
            i++;
            rowCur = rowCur->getDown();
            colCur = rowCur;

            //printf("cur ID: %d ", colCur->getID());
        }

    }



    // Now we need to link up and down
    // colCur reset as top left
    // also locate them accordingly
    rowCur = this->topLeft;
    colCur = this->topLeft;
    for(int j=0;j<cpuDim - 1;j++){

        colCur->getDown()->setUp(colCur);
        colCur = colCur->getRight();
        for(int k=1;k<cpuDim;k++){
            colCur->getLeft()->getDown()->getRight()->setUp(colCur);
            colCur->setDown(colCur->getLeft()->getDown()->getRight());
            colCur = colCur->getRight();
        }
        rowCur = rowCur->getDown();
        colCur = rowCur;
    }

}


void cpuNet::printNet(){
```

```
    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;
    while(rowCur != NULL){
        while(colCur != NULL){
            colCur->printNode();
            colCur = colCur->getRight();
        }
        rowCur = rowCur->getDown();
        colCur = rowCur;
    }
}

void cpuNet::renderNet(){
    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;
    cpuMsg *tmp = new cpuMsg();

    while(rowCur != NULL){
        while(colCur != NULL){
            for(int i=0;i<4;i++){
                if(queue_length(colCur->msg_q[i]) > 0) {
                    tmp = (cpuMsg *)queue_first_data(colCur->msg_q[i]);
                    switch(tmp->state){
                        case CREATE:    colCur->c.set(0,0,1);
                                        break;
                        case TRANSIT:   colCur->c.set(tmp->c);
                                        break;
                        case ALTERED:   colCur->c.set(1,1,0);
                                        break;
                        case RESOLVEWAIT1:  colCur->c.set(1,0,1);
                                            break;
                        case RESOLVEWAIT2:  colCur->c.set(1,0,.5);
                                            break;
                        case RESOLVED:  colCur->c.set(0,1,0);
                                        break;

                    }
                }
            }
            if(queue_length(colCur->res_msg_q) > 0) colCur->c.set(0, 1, 0);
            else if(colCur->hasMsg == false) colCur->c.set(1,1,1);
            colCur->renderNode();
            colCur = colCur->n_getRight();
        }
        rowCur = rowCur->n_getDown();
        colCur = rowCur;
    }
}

void cpuNet::updateNet(){
    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;
    while(rowCur != NULL){
        while(colCur != NULL){
            //colCur->fireNode();
            colCur->q_fireNode();
            colCur = colCur->n_getRight();
        }
        rowCur = rowCur->n_getDown();
        colCur = rowCur;
    }


    // refresh msg states
    rowCur = this->topLeft;
    colCur = this->topLeft;
    cpuMsg *tmp = new cpuMsg();
    while(rowCur != NULL){
        while(colCur != NULL){
            colCur->hasMsg = false;
            for(int i=0;i<4;i++){
                if(queue_length(colCur->msg_q[i]) > 0){
                    colCur->hasMsg = true;
                    tmp = (cpuMsg *)queue_first_data(colCur->msg_q[i]);
                    if(tmp->state == ALTERED){
                        tmp->state = TRANSIT;
                    }
                }
            }
            colCur = colCur->n_getRight();
        }
        rowCur = rowCur->n_getDown();
```

```
        colCur = rowCur;
    }
}

cpuNode* cpuNet::getNode(int id){
    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;
    while(rowCur != NULL){
        while(colCur != NULL){
            if(colCur->ID == id) return colCur;
            colCur = colCur->n_getRight();
        }
        rowCur = rowCur->n_getDown();
        colCur = rowCur;
    }
    return NULL;
}

void cpuNet::removeNode(int id){
    cpuNode *rowCur = this->topLeft;
    cpuNode *colCur = this->topLeft;
    while(rowCur != NULL){
        while(colCur != NULL){
            if(colCur->ID == id) {
                colCur->severNode();
                return;
            }
            colCur = colCur->n_getRight();
        }
        rowCur = rowCur->n_getDown();
        colCur = rowCur;
    }
    return;
}
```

```
class cpuNode {

private:
    // This is known only to cpu node
    // represent physical connections
    cpuNode *up;
    cpuNode *down;
    cpuNode *left;
    cpuNode *right;

    // For net use
    cpuNode *n_up;
    cpuNode *n_down;
    cpuNode *n_left;
    cpuNode *n_right;

    // physical location
    Point_3 Loc;


public:

    // color
    Color_3 c;

    // current message
    //cpuMsg *msg[4];
    bool hasMsg;

    // add msg queue
    queue_t msg_q[4];

    queue_t res_msg_q;


    // This is available to other nodes
    int ID;

    cpuNode ();
    cpuNode (int);
    cpuNode (int, cpuNode*, cpuNode*, cpuNode*, cpuNode*);

    int getID();
    void printNode();

    //Represent Node ala Box
    // With half "pipes" refering to an attempted connection
    // Broken connections can be seen as an attempted connection by one
    // node and no return connection by the other
    void renderNode();

    int createRight(int);
    int createLeft(int);
    int createUp(int);
    int createDown(int);

    cpuNode* getUp();
    cpuNode* getDown();
    cpuNode* getLeft();
    cpuNode* getRight();

    cpuNode* n_getUp();
    cpuNode* n_getDown();
    cpuNode* n_getLeft();
    cpuNode* n_getRight();

    int setUp(cpuNode*);
    int setDown(cpuNode*);
    int setLeft(cpuNode*);
    int setRight(cpuNode*);

    void severUp();
    void severDown();
    void severLeft();
    void severRight();

    void severNode();

    void sendMsg(int, Color_3*);
    void sendMsg(cpuMsg*);
```

```
    void replyMsg(int, int, int, Color_3*);
    void resolveMsg(int);
    void destroyMsg(int);

    void setLoc(Point_3);

    void fireNode();
    void q_fireNode();


};
```

```
#include "stdafx.h"

extern float rotateAngle;

cpuNode::cpuNode(){
    // uninitialized
    this->ID = -1;
    this->right = NULL;
    this->left = NULL;
    this->up = NULL;
    this->down = NULL;
    this->n_right = NULL;
    this->n_left = NULL;
    this->n_up = NULL;
    this->n_down = NULL;
    this->Loc = Point_3(0,0,0);
    this->c = Color_3(1,1,1);
    for(int i=0;i<4;i++) this->msg_q[i] = queue_new();
    this->res_msg_q = queue_new();
    this->hasMsg = false;

}

cpuNode::cpuNode(int id){
    // uinitialized but unconnected
    this->ID = id;
    this->right = NULL;
    this->n_right = NULL;
    this->left = NULL;
    this->n_left = NULL;
    this->up = NULL;
    this->n_up = NULL;
    this->down = NULL;
    this->n_down = NULL;
    this->Loc = Point_3(0,0,0);
    this->c = Color_3(1,1,1);
    for(int i=0;i<4;i++) this->msg_q[i] = queue_new();
    this->res_msg_q = queue_new();
    this->hasMsg = false;

}

cpuNode::cpuNode(int id, cpuNode *Up, cpuNode *Down, cpuNode *Right, cpuNode *Left){
    // initialized and connected
    this->ID = id;
    this->down = Down;
    this->n_down = Down;
    this->up = Up;
    this->n_up = Up;
    this->right = Right;
    this->n_right = Right;
    this->left = Left;
    this->n_left = Left;
    this->Loc = Point_3(0,0,0);
    this->c = Color_3(1,1,1);
    for(int i=0;i<4;i++) this->msg_q[i] = queue_new();
    this->res_msg_q = queue_new();
    this->hasMsg = false;

}

int cpuNode::getID() { return this->ID; }

void cpuNode::printNode(){
    /*printf("Node:%d ",    this->ID);

    // Up
    if(this->getUp() != NULL) printf("U:%d ", this->getUp()->ID);
    else printf("U:N ");
    //Down
    if(this->getDown() != NULL) printf("D:%d ", this->getDown()->ID);
    else printf("D:N ");
    //Right
    if(this->getRight() != NULL) printf("R:%d ", this->getRight()->ID);
    else printf("R:N ");
    //Left
    if(this->getLeft() != NULL) printf("L:%d\n", this->getLeft()->ID);
    else printf("L:N\n");*/
}

void cpuNode::renderNode(){
    glPushMatrix();
```

```
        glTranslatef(this->Loc.x, this->Loc.y, this->Loc.z); // Move Right 3 Units
        //glRotatef(rotateAngle, 1, 1, 1);
        glColor3f(this->c.r, this->c.g, this->c.b);

        glBegin(GL_QUADS); // Draw A Quad
            glVertex3f(-1.0f, 1.0f, 0.0f); // Top Left
            glVertex3f( 1.0f, 1.0f, 0.0f); // Top Right
            glVertex3f( 1.0f,-1.0f, 0.0f); // Bottom Right
            glVertex3f(-1.0f,-1.0f, 0.0f); // Bottom Left
        glEnd(); // Done Drawing The Quad

        if(this->up != NULL){
        glBegin(GL_QUADS); // Draw A Quad
            glVertex3f(-0.25f, 2.0f, 0.0f); // Top Left
            glVertex3f( 0.25f, 2.0f, 0.0f); // Top Right
            glVertex3f( 0.25f,1.0f, 0.0f); // Bottom Right
            glVertex3f(-0.25f,1.0f, 0.0f); // Bottom Left
        glEnd(); // Done Drawing The Quad
        }

        if(this->down != NULL){
        glBegin(GL_QUADS); // Draw A Quad
            glVertex3f(-0.25f, -1.0f, 0.0f); // Top Left
            glVertex3f( 0.25f, -1.0f, 0.0f); // Top Right
            glVertex3f( 0.25f,-2.0f, 0.0f); // Bottom Right
            glVertex3f(-0.25f,-2.0f, 0.0f); // Bottom Left
        glEnd(); // Done Drawing The Quad
        }

        if(this->left != NULL){
        glBegin(GL_QUADS); // Draw A Quad
            glVertex3f(-2.0f, 0.25f, 0.0f); // Top Left
            glVertex3f( -1.0f, 0.25f, 0.0f); // Top Right
            glVertex3f( -1.0f,-0.25f, 0.0f); // Bottom Right
            glVertex3f(-2.0f,-0.25f, 0.0f); // Bottom Left
        glEnd(); // Done Drawing The Quad
        }

        if(this->right != NULL){
        glBegin(GL_QUADS); // Draw A Quad
            glVertex3f(1.0f, 0.25f, 0.0f); // Top Left
            glVertex3f( 2.0f, 0.25f, 0.0f); // Top Right
            glVertex3f( 2.0f,-0.25f, 0.0f); // Bottom Right
            glVertex3f(1.0f,-0.25f, 0.0f); // Bottom Left
        glEnd(); // Done Drawing The Quad
        }



    glPopMatrix();
}

cpuNode* cpuNode::getUp(){return this->up;}
cpuNode* cpuNode::getDown(){return this->down;}
cpuNode* cpuNode::getLeft(){return this->left;}
cpuNode* cpuNode::getRight(){return this->right;}

cpuNode* cpuNode::n_getUp(){return this->n_up;}
cpuNode* cpuNode::n_getDown(){return this->n_down;}
cpuNode* cpuNode::n_getLeft(){return this->n_left;}
cpuNode* cpuNode::n_getRight(){return this->n_right;}

int cpuNode::setUp(cpuNode *u)
{
    this->up = u;
    this->n_up = u;
    if(this->up == u) return true;
    else return false;
}
int cpuNode::setDown(cpuNode *d)
{
    this->down = d;
    this->n_down = d;
    if(this->down == d) return true;
    else return false;
}
int cpuNode::setLeft(cpuNode *l)
{
    this->left = l;
    this->n_left = l;
    if(this->left == l) return true;
```

```
    else return false;
}
int cpuNode::setRight(cpuNode *r)
{
    this->right = r;
    this->n_right = r;
    if(this->right == r) return true;
    else return false;
}

void cpuNode::severUp()
{
    if(this->up != NULL) this->up = NULL;
}
void cpuNode::severDown()
{
    if(this->down != NULL) this->down = NULL;
}
void cpuNode::severLeft()
{
    if(this->left != NULL) this->left = NULL;
}
void cpuNode::severRight()
{
    if(this->right != NULL) this->right = NULL;
}



void cpuNode::setLoc(Point_3 a){
    this->Loc = a;
}


int cpuNode::createRight(int id){
    cpuNode *tempUp;
    cpuNode *tempDown;

    if(this->up != NULL) {
        tempUp = this->up->getRight();
    } else tempUp = NULL;

    if(this->down != NULL) tempDown = this->down->getRight();
    else tempDown = NULL;

    this->right = new cpuNode(id,
                              tempUp,
                              tempDown,
                              NULL,
                              this);
    this->n_right = this->right;

    if(this->right != NULL) return 1;
    else return 0;
}
int cpuNode::createLeft(int id){
    cpuNode *tempUp;
    cpuNode *tempDown;

    if(this->up != NULL) tempUp = this->up->getLeft();
    else tempUp = NULL;

    if(this->down != NULL) tempDown = this->down->getLeft();
    else tempDown = NULL;

    this->left  = new cpuNode(id,
                              tempUp,
                              tempDown,
                              this,
                              NULL);

    this->n_left = this->left;

    if(this->left != NULL) return 1;
    else return 0;
}

int cpuNode::createUp(int id){
    cpuNode *tempRight;
    cpuNode *tempLeft;

    if(this->left != NULL) tempLeft = this->left->getUp();
```

```
    else tempRight = NULL;

    if(this->right != NULL) tempRight = this->right->getUp();
    else tempLeft = NULL;

    this->up    = new cpuNode(id,
                              NULL,
                              this,
                              tempRight,
                              tempLeft);

    this->n_up = this->up;

    if(this->up != NULL) return 1;
    else return 0;
}

int cpuNode::createDown(int id){
    cpuNode *tempRight;
    cpuNode *tempLeft;

    if(this->left != NULL) tempLeft = this->left->getDown();
    else tempLeft = NULL;

    if(this->right != NULL) tempRight = this->right->getDown();
    else tempRight = NULL;

    this->down  = new cpuNode(id,
                              this,
                              NULL,
                              tempRight,
                              tempLeft);

    this->n_down = this->down;


    if(this->down != NULL) return 1;
    else return 0;
}

void cpuNode::severNode(){
    if(this->up != NULL) this->up->severDown();
    this->up = NULL;
    if(this->down != NULL) this->down->severUp();
    this->down = NULL;
    if(this->left != NULL) this->left->severRight();
    this->left = NULL;
    if(this->right != NULL) this->right->severLeft();
    this->right = NULL;
}

void cpuNode::sendMsg(int dest, Color_3 *color){
    this->hasMsg = true;
    cpuMsg *tmp;


    for(int i=0 ;i<4; i++) {
        tmp = new cpuMsg(this->ID, dest, i, color);
        queue_append(this->msg_q[i], tmp);
    }
}

void cpuNode::replyMsg(int dest, int dir, int ldir, Color_3 *color){
    //this->hasMsg = true;
    //for(int i=0;i<4;i++) this->msg[i] = NULL;
    for(int i=0;i<4;i++) {
        if(i != dir && i != ldir) queue_append(this->msg_q[i], new cpuMsg(this->ID, dest, i, color));
    }
}

void cpuNode::sendMsg(cpuMsg *m){
    //this->hasMsg = true;


    //also prevent message being passed back in the direction it came from
    for(int i=0;i<4;i++){
        switch(m->dir){

            case UP:    if(i != DOWN) {

                            if(m->lastdir == UP) queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m-
                                >dest_id, i, m->dir, m->c));
```

```
                        }
                        break;


            case DOWN:  if(i != UP) {
                            if(m->lastdir == DOWN) queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id,
                                m->dest_id, i, m->dir, m->c));
                            //queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->dir
                                ));
                        /*
                            if(m->lastdir == DOWN)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                            else if(m->lastdir == LEFT && i != RIGHT && i != DOWN)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, LEFT
                                    ));
                            else if(m->lastdir == RIGHT && i != LEFT && i != DOWN)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i,
                                    RIGHT));
                        */
                        }
                        break;

            case LEFT:  if(i != RIGHT) {
                            if(i == LEFT) queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->
                                dest_id, i, m->dir, m->c));
                        /*
                            if(m->lastdir == LEFT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                            else if(m->lastdir == UP && i != DOWN && i != LEFT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                            else if(m->lastdir == DOWN && i != UP && i != LEFT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                        */
                        }
                        break;

            case RIGHT: if(i != LEFT) {
                            if(i == RIGHT) queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->
                                dest_id, i, m->dir, m->c));
                        /*
                            if(m->lastdir == RIGHT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                            else if(m->lastdir == UP && i != DOWN && i != RIGHT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                            else if(m->lastdir == DOWN && i != UP && i != RIGHT)
                                queue_append(this->msg_q[i], new cpuMsg(m->orig_id, m->cur_id, m->dest_id, i, m->
                                    dir));
                        */
                        }
                        break;
        }

    }
}


void cpuNode::q_fireNode(){
    cpuMsg *tmp = new cpuMsg();
    int i = 0;

    if(queue_length(this->res_msg_q) > 0){
    // need to resolve already resolved msgs first
        tmp = (cpuMsg *)queue_first_data(this->res_msg_q);
        queue_dequeue(this->res_msg_q, tmp);

        if(tmp->state == RESOLVED){
                int dest = tmp->orig_id;
                int dir = tmp->dir;
                int ldir = tmp->lastdir;
                this->replyMsg(dest, dir, ldir, tmp->c);
                delete tmp;
        } else if(tmp->state == RESOLVEWAIT3){
                 tmp->state = RESOLVED;
```

```
                queue_append(this->res_msg_q, tmp);
        } else  if(tmp->state == RESOLVEWAIT2){
                tmp->state = RESOLVEWAIT3;
                queue_append(this->res_msg_q, tmp);
        } else  if(tmp->state == RESOLVEWAIT1){
                tmp->state = RESOLVEWAIT2;
                queue_append(this->res_msg_q, tmp);
        }
    }

for(int i=0;i<4;i++)
{
    if(queue_length(this->msg_q[i]) > 0)
    {

        tmp = (cpuMsg *)queue_first_data(this->msg_q[i]);
        queue_dequeue(this->msg_q[i], tmp);


        if( tmp->state != RESOLVEWAIT2 &&
            tmp->state != RESOLVEWAIT3 &&
            tmp->state != RESOLVED)
        {
            if(tmp->dest_id == this->ID)
            {
                tmp->state = RESOLVEWAIT1;
                queue_append(this->res_msg_q, tmp);
                break;
            }
            else if(tmp->state == TRANSIT || tmp->state == CREATE)
            {

                tmp->state = ALTERED;

                switch(tmp->dir)
                {
                    case UP:    if(this->getUp() != NULL)
                                {
                                    tmp->cur_id = this->getUp()->ID;
                                    this->getUp()->sendMsg(tmp);
                                    delete tmp;
                                }
                                else
                                {
                                    //this->destroyMsg(i);
                                    delete tmp;
                                }
                                break;


                    case DOWN:  if(this->getDown() != NULL)
                                {
                                    tmp->cur_id = this->getDown()->ID;
                                    this->getDown()->sendMsg(tmp);
                                    delete tmp;
                                }
                                else
                                {
                                    //this->destroyMsg(i);
                                    delete tmp;
                                }
                                break;

                    case LEFT:  if(this->getLeft() != NULL)
                                {
                                    tmp->cur_id = this->getLeft()->ID;
                                    this->getLeft()->sendMsg(tmp);
                                    delete tmp;
                                }
                                else
                                {
                                    //this->destroyMsg(i);
                                    delete tmp;
                                }
                                break;

                    case RIGHT: if(this->getRight() != NULL)
                                {
                                    tmp->cur_id = this->getRight()->ID;
                                    this->getRight()->sendMsg(tmp);
                                    delete tmp;
                                }
```

```
                                else
                                {
                                    //this->destroyMsg(i);
                                    delete tmp;
                                }
                                break;

                        default:    delete tmp;
                                    break;

                        }
                }
                else if(tmp->state == ALTERED)
                {
                    queue_append(this->msg_q[i], tmp);
                }

            }
            //delete tmp;
        }
}

}

void cpuNode::resolveMsg(int i){
    cpuMsg *tmp = new cpuMsg();

    tmp = (cpuMsg *)queue_first_data(this->msg_q[i]);
    queue_dequeue(this->msg_q[i], tmp);

    free(tmp);
}

void cpuNode::destroyMsg(int i){
    cpuMsg *tmp = new cpuMsg();

    tmp = (cpuMsg *)queue_first_data(this->msg_q[i]);
    queue_dequeue(this->msg_q[i], tmp);

    free(tmp);
}
```

```
#define UP 3
#define LEFT 2
#define DOWN 1
#define RIGHT 0

#define CREATE 0
#define TRANSIT 1
#define ALTERED 2
#define RESOLVEWAIT1 3
#define RESOLVEWAIT2 4
#define RESOLVEWAIT3 5
#define RESOLVED 6


class cpuMsg {
public:
    int orig_id;
    int cur_id;
    int dest_id;

    int dir;
    int lastdir;
    int state;

    Color_3 *c;

    cpuMsg ();
    cpuMsg (int c, int d, Color_3* color);
    cpuMsg (int c, int d, int dir, Color_3* color);
    cpuMsg (int o, int c, int d, int dir, int ldir, Color_3* color);

    void sendMsg(int destination);
};
```

```
#include "stdafx.h"


cpuMsg::cpuMsg(){
    this->orig_id = -1;
    this->cur_id = -1;
    this->dest_id = -1;
    // uninitialized direction
    this->dir = -1;
    this->state = CREATE;
    this->lastdir = -1;
}

cpuMsg::cpuMsg(int current, int destination, Color_3 *color){
    this->orig_id = current;
    this->cur_id = current;
    this->dest_id = destination;
    // uninitialized direction
    this->dir = -1;
    this->state = CREATE;
    this->lastdir = -1;
    this->c = color;
}

cpuMsg::cpuMsg(int current, int destination, int direction, Color_3 *color){
    this->orig_id = current;
    this->cur_id = current;
    this->dest_id = destination;
    this->dir = direction;
    this->state = CREATE;
    this->lastdir = direction;
    this->c = color;
}

cpuMsg::cpuMsg(int original, int current, int destination, int direction, int lastdirect, Color_3 *color){
    this->orig_id = original;
    this->cur_id = current;
    this->dest_id = destination;
    this->dir = direction;
    this->state = ALTERED;
    this->lastdir = lastdirect;
    this->c = color;
}
```

```c
/*
 * Generic queue manipulation functions
 */
#ifndef __QUEUE_H__
#define __QUEUE_H__



/*
 * any_t is a pointer.  This allows you to put arbitrary structures on
 * the queue.
 */
typedef void *any_t;

/*
 * PFany is a pointer to a function that can take two any_t arguments
 * and return an integer.
 */
typedef int (*PFany)(any_t, any_t);

/*
 * queue_t is a pointer to an internally maintained data structure.
 * Clients of this package do not need to know how queues are
 * represented.  They see and manipulate only queue_t's.
 */
typedef any_t queue_t;

typedef struct queueNode *queuenode_t;


/*
 * Return an empty queue. On error should return NULL.
 */
extern queue_t queue_new();

/*
 * Prepend an any_t to a queue (both specifed as parameters).  Return
 * 0 (success) or -1 (failure).
 */
extern int queue_prepend(queue_t, any_t);

/*
 * Appends an any_t to a queue (both specifed as parameters).  Return
 * 0 (success) or -1 (failure).
 */
extern int queue_append(queue_t, any_t);

/*
 * Dequeue and return the first any_t from the queue. Return 0
 * (success) and first item if queue is nonempty, or -1 (failure) and
 * NULL if queue is empty.
 */
extern int queue_dequeue(queue_t, any_t);

/*
 * Return the data element of the first node in the queue
 */
extern any_t queue_first_data(queue_t);

/*
 * Iterate the function parameter over each element in the queue.  The
 * additional any_t argument is passed to the function as its first
 * argument and the queue element is the second.
 * Return 0 (success) or -1 (failure).
 */
extern int queue_iterate(queue_t, PFany, any_t);

/*
 * Free the queue and return 0 (success) or -1 (failure).
 */
extern int queue_free (queue_t);

/*
 * Return the number of items in the queue.
 */
extern int queue_length(queue_t);

/*
 * Delete the specified item from the given queue.
 * Return -1 on error.
 */
extern int queue_delete(queue_t, any_t);
```

```
#endif __QUEUE_H__
```

```
#include "stdafx.h"

/*
 * Generic queue implementation.
 *
 */
#include "queue.h"
#include <stdlib.h>
#include <stdio.h>


/*
 * Create our nodes
 */
struct queueNode {
    struct queueNode *prev;
    struct queueNode *next;
    any_t data;
} qNode;


/*
 * create our queue struct for the actual linked list
 */

struct queue {
    struct queueNode *first;    // used to append
    struct queueNode *last; // used to pop
    int length;
};


//extern struct queue Q;

/*
 * Return an empty queue.
 */
queue_t queue_new() {
    struct queue *Q = (struct queue *) malloc(sizeof(struct queue));
    if(Q == NULL) return NULL;
    Q->first = NULL;
    Q->last = NULL;
    Q->length = 0;
    return (queue_t) Q;
}

/*
 * Prepend an any_t to a queue (both specifed as parameters).  Return
 * 0 (success) or -1 (failure).
 */
int queue_prepend(queue_t Q, any_t item) {
    // need to declare temp node here
    struct queueNode *t = (struct queueNode *) malloc(sizeof(struct queueNode));
    if(t == NULL) return -1;
        if(((struct queue *)Q)->length == 0){
            ((struct queue *)Q)->length++;
            t->prev = NULL;
            t->next = NULL;
            t->data = item;
            ((struct queue *)Q)->first = t;
            // set last to the only node in the list
            ((struct queue *)Q)->last  = t;
        } else {
            ((struct queue *)Q)->length++;
            t->prev = NULL;
            ((struct queue *)Q)->first->prev = t;
            t->next = ((struct queue *)Q)->first;
            t->data = item;
            ((struct queue *)Q)->first = t;
        }
        return 0;

}

/*
 * Append an any_t to a queue (both specifed as parameters). Return
 * 0 (success) or -1 (failure).
 */
int queue_append(queue_t Q, any_t item) {
    // need to declare temp node here
    struct queueNode *t = (struct queueNode *) malloc(sizeof(struct queueNode));
    if(t == NULL) return -1;
```

```
        if(((struct queue *)Q)->length == 0){
            ((struct queue *)Q)->length++;
            t->prev = NULL;
            t->next = NULL;
            t->data = item;
            ((struct queue *)Q)->first = t;
            // set last to the only node in the list
            ((struct queue *)Q)->last  = t;
        } else {
            ((struct queue *)Q)->length++;
            t->prev = ((struct queue *)Q)->last;
            t->next = NULL;
            t->data = item;
            ((struct queue *)Q)->last->next = t;
            ((struct queue *)Q)->last = t;
        }
        return 0;

}

/*
 * Dequeue and return the first any_t from the queue or NULL if queue
 * is empty.  Return 0 (success) or -1 (failure).
 */
int queue_dequeue(queue_t queue, any_t item) {
    // need to declare temp node here
    struct queueNode *t = (struct queueNode *) malloc(sizeof(struct queueNode));
    if(t == NULL) return -1;
    if(((struct queue *)queue)->length == 0){
        item = NULL;
        free(t);
    } else {
        ((struct queue *)queue)->length--;
        t = ((struct queue *)queue)->first;

        item = ((struct queue *)queue)->first->data;

        ((struct queue *)queue)->first = t->next;

        if(((struct queue *)queue)->length != 0) ((struct queue *)queue)->first->prev = NULL;
        else ((struct queue *)queue)->last = NULL;



        //t->prev = NULL;

        free(t);
    }
    return 0;
}

/*
 * Iterate the function parameter over each element in the queue.  The
 * additional any_t argument is passed to the function as its first
 * argument and the queue element is the second.  Return 0 (success)
 * or -1 (failure).
 */
int queue_iterate(queue_t queue, PFany f, any_t item) {
    struct queueNode *cur = ((struct queue *)queue)->first;
    if(cur == NULL){
        free(cur);
        return -1;
    } else {
        while(cur != NULL){
            cur->data = (any_t)f(item, cur->data);
            cur = cur->next;
        }
        free(cur);
    }
    return 0;
}

/*
 * Free the queue and return 0 (success) or -1 (failure).
 */
int queue_free (queue_t queue) {
    struct queueNode *cur = ((struct queue *)queue)->last->prev;
    if(cur == NULL){
        free(cur);
        free(queue);
        return -1;
    } else {
```

```
        while(cur != NULL){
            free(cur->next);
            cur = cur->prev;
        }
        free(queue);
        return 0;
    }
}

/*
 * Return the number of items in the queue.
 */
int queue_length(queue_t queue) {
    return ((struct queue *)queue)->length;
}

any_t queue_first_data(queue_t Q){
    return ((struct queue *)Q)->first->data;
}


/*
 * Delete the specified item from the given queue.
 * Return -1 on error.
 */
int queue_delete(queue_t queue, any_t item) {
    struct queueNode *cur = ((struct queue *)queue)->first;
    if(cur == NULL) {
        free(cur);
        return -1;
    } else {
        while(cur->data != item && cur != NULL){
            cur = cur->next;
        }
        if(cur != NULL){
            // means we found our item in the list
            // remove links to node and then free node
            ((struct queue *)queue)->length--;

            if(queue_length(queue) > 0){
                // if cur is first node?
                if(cur->prev != NULL) {
                    cur->prev->next = cur->next;
                } else ((struct queue *)queue)->first = cur->next;
                // if cur is last node
                if(cur->next != NULL) {
                    cur->next->prev = cur->prev;
                } else ((struct queue *)queue)->last = cur->prev;
                free(cur);
            } else {
                cur->prev = NULL;
                cur->next = NULL;
                cur = NULL;
                ((struct queue *)queue)->last = NULL;
            }
            //free(cur);
            return 0;
        } else {
            return -1;
        }
    }
}
```

```
#include "stdafx.h"

Point_3::Point_3(float x, float y, float z){
    this->x = x;
    this->y = y;
    this->z = z;
}

Point_3 Point_3::operator + (Point_3 a){
    Point_3 temp;
    temp.x = this->x + a.x;
    temp.y = this->y + a.y;
    temp.z = this->z + a.z;
    return (temp);
}

Point_3 Point_3::operator - (Point_3 a){
    Point_3 temp;
    temp.x = this->x - a.x;
    temp.y = this->y - a.y;
    temp.z = this->z - a.z;
    return (temp);
}

Vector_3::Vector_3(float x, float y, float z){
    this->x = x;
    this->y = y;
    this->z = z;
}

Vector_3 Vector_3::operator + (Vector_3 a){
    Vector_3 temp;
    temp.x = this->x + a.x;
    temp.y = this->y + a.y;
    temp.z = this->z + a.z;
    return (temp);
}

Vector_3 Vector_3::operator - (Vector_3 a){
    Vector_3 temp;
    temp.x = this->x - a.x;
    temp.y = this->y - a.y;
    temp.z = this->z - a.z;
    return (temp);
}

Color_3::Color_3(float r, float g, float b){
    this->r = r;
    this->g = g;
    this->b = b;
}



Color_3 Color_3::operator + (Color_3 a){
    Color_3 temp;
    temp.r = this->r + a.r;
    temp.g = this->g + a.g;
    temp.b = this->b + a.b;
    return (temp);
}

Color_3 Color_3::operator - (Color_3 a){
    Color_3 temp;
    temp.r = this->r - a.r;
    temp.g = this->g - a.g;
    temp.b = this->b - a.b;
    return (temp);
}

void Color_3::set(float r, float g, float b){
    this->r = r;
    this->g = g;
    this->b = b;
}

void Color_3::set(Color_3 *c){
    this->r = c->r;
    this->g = c->g;
    this->b = c->b;
}
```

```
class Point_3 {
public:
    float x;
    float y;
    float z;

    Point_3() {};
    Point_3(float, float, float);

    Point_3 operator + (Point_3);
    Point_3 operator - (Point_3);

};

class Vector_3 {
public:
    float x;
    float y;
    float z;

    Vector_3() {};
    Vector_3(float, float, float);

    Vector_3 operator + (Vector_3);
    Vector_3 operator - (Vector_3);
};

class Color_3 {
public:
    float r;
    float g;
    float b;

    Color_3() {};
    Color_3(float, float, float);

    Color_3 operator + (Color_3);
    Color_3 operator - (Color_3);

    void set(float, float, float);
    void set(Color_3*);
};
```