```
        pc = pc + 1;
        doacci(data);
    end
    endtask


    /* operate on accumulator */
    task doacci;
    input[7:0] sr;
    reg[3:0] null4;
    reg[7:0] null8;
        case(ir[5:3])
            0: // ADD ADI
                begin
                    {cac, null4} = acc + sr;
                    {cc, acc} = {1'b0, acc} + sr;
                    calpsz(acc);
                end

            1: // ADC ACI
                begin
                    {cac, null4} = acc + sr + cc;
                    {cc, acc} = {1'b0, acc} + sr + cc;
                    calpsz(acc);
                end

            2: // SUB SUI
                begin
                    {cac, null4} = acc - sr;
                    {cc, acc} = {1'b0, acc} - sr;
                    calpsz(acc);
                end

            3: // SBB SBI
                begin
                    {cac, null4} = acc - sr - cc;
                    {cc, acc} = {1'b0, acc} - sr - cc;
                    calpsz(acc);
                end

            4: // ANA ANI
                begin
                    acc = acc & sr;
                    cac = 1;
                    cc = 0;
```

```
                calpsz(acc);
            end

        5: // XRA XRI
            begin
                acc = acc ^ sr;
                cac = 0;
                cc = 0;
                calpsz(acc);
            end

        6: // ORA ORI
            begin
                acc = acc | sr;
                cac = 0;
                cc = 0;
                calpsz(acc);
            end

        7: // CMP CPI
            begin
                {cac, null4} = acc - sr;
                {cc, null8} = {1'b0, acc} - sr;
                calpsz(null8);
            end
    endcase
endtask

/* rotate acc and special instructions */
task racc_spec;
    case(ir[5:3])
        0: // RLC
            begin
                acc = {acc[6:0], acc[7]};
                cc = acc[7];
            end

        1: // RRC
            begin
                acc = {acc[0], acc[7:1]};
                cc = acc[0];
            end

        2: // RAL
            {cc, acc} = {acc, cc};
```

```
        3: // RAR
            {acc, cc} = {cc, acc};

        4: // DAA, decimal adjust
            begin
                if((acc[3:0] > 9) || cac) acc = acc + 6;
                if((acc[7:4] > 9) || cc) {cc, acc} = {1'b0, acc} + 'h60;
            end

        5: // CMA
            acc = ~acc;

        6: // STC
            cc = 1;

        7: // CMC
            cc = ~cc;
    endcase
endtask


/* increment and decrement register pair */
task inx_dcx;
    case(ir[5:3])
        0: {regb, regc} = {regb, regc} + 1; // INX B
        1: {regb, regc} = {regb, regc} - 1; // DCX B
        2: {regd, rege} = {regd, rege} + 1; // INX D
        3: {regd, rege} = {regd, rege} - 1; // DCX D
        4: {regh, regl} = {regh, regl} + 1; // INX H
        5: {regh, regl} = {regh, regl} - 1; // DCX H
        6: sp = sp + 1;                     // INX SP
        7: sp = sp - 1;                     // DCX SP
    endcase
endtask

/* load register pair immediate */
task lrpi;
    case(ir[5:4])
        0: adread({regb, regc}); // LXI B
        1: adread({regd, rege}); // LXI D
        2: adread({regh, regl}); // LXI H
        3: adread(sp);           // LXI SP
    endcase
endtask
```

```
/* add into regh, regl pair */
task addhl;
begin
    case(ir[5:4])
        0: {cc, regh, regl} = {1'b0, regh, regl} + {regb, regc}; // DAD B
        1: {cc, regh, regl} = {1'b0, regh, regl} + {regd, rege}; // DAD D
        2: {cc, regh, regl} = {1'b0, regh, regl} + {regh, regl}; // DAD H
        3: {cc, regh, regl} = {1'b0, regh, regl} + sp;           // DAD SP
    endcase
    holdreq;
    holdreq;
end
endtask


/* store and load instruction */
task sta_lda;
reg[15:0] ra;
    case(ir[5:3])
        0: memwrite(acc, {regb, regc}); // STAX B
        1: memread(acc, {regb, regc});  // LDAX B
        2: memwrite(acc, {regd, rege}); // STAX D
        3: memread(acc, {regd, rege});  // LDAX D

        4: // SHLD
            begin
                adread(ra);
                memwrite(regl, ra);
                memwrite(regh, ra + 1);
            end
        5: // LHLD
            begin
                adread(ra);
                memread(regl, ra);
                memread(regh, ra + 1);
            end

        6: // STA
            begin
                adread(ra);
                memwrite(acc, ra);
            end
        7: // LDA
```

```
            begin
                adread(ra);
                memread(acc, ra);
            end
    endcase
endtask

/* push register pair from stack */
task push;
    case(ir[5:4])
        0: push2b(regb, regc); // PUSH B
        1: push2b(regd, rege); // PUSH D
        2: push2b(regh, regl); // PUSH H
        3: push2b(acc, {cs,cz,1'b1,cac,1'b1,cp,1'b1,cc}); // PUSH PSW
    endcase
endtask

/* push 2 bytes onto stack */
task push2b;
input[7:0] highb, lowb;
begin
    sp = sp - 1;
    memwrite(highb, sp);
    sp = sp - 1;
    memwrite(lowb, sp);
end
endtask


/* pop register pair from stack */
task pop;
reg null1;
    case(ir[5:4])
        0: pop2b(regb, regc); // POP B
        1: pop2b(regd, rege); // POP D
        2: pop2b(regh, regl); // POP H
        3: pop2b(acc,
            {cs, cz, null1, cac, null1, cp, null1, cc}); // POP PSW
    endcase
endtask

/* pop 2 bytes from stack */
task pop2b;
output[7:0] highb, lowb;
begin
```

```
        memread(lowb, sp);
        sp = sp + 1;
        memread(highb, sp);
        sp = sp + 1;
    end
    endtask

    /* check hold request */
    task holdreq;
    begin
        aleff = 0;
        s0ff = 0;
        s1ff = 1;
        iomff = 0;
        addr = pc;
        if(hold) begin
            holdff = 1;
            acontrol = 0;
            dcontrol = 0;
            @ec2 hldaff = 1;
        end
        else begin
            acontrol = 1;
            dcontrol = 1;
        end
        @ec1 dcontrol = 0;
        @ec1 @ec2;
    end
    endtask

    /* conditional jump, call and return instructions */
    task condjcr;
    reg branch;
    begin
        case(ir[5:3])
            0: branch = !cz; // JNZ CNZ RNZ
            1: branch = cz;  // JZ  CZ  RZ
            2: branch = !cc; // JNC CNC RNC
            3: branch = cc;  // JC  CC  RC
            4: branch = !cp; // JPO CPO RPO
            5: branch = cp;  // JPE CPE RPE
            6: branch = !cs; // JP  CP  RP
            7: branch = cs;  // JM  CM  RM
        endcase
        if(branch)
```

```
            case(ir[2:0])
                0: // return
                    pop2b(pc[15:8], pc[7:0]);

                2: // jump
                    adread(pc);

                4: // call
                    begin :call
                        reg [15:0] newpc;
                        adread(newpc);
                        push2b(pc[15:8], pc[7:0]);
                        pc = newpc;
                    end

                default no_instruction;
            endcase
        else
            case(ir[2:0])
                0: ;
                2, 4:
                    begin
                        memread(data, pc);
                        pc = pc + 2;
                    end
                default no_instruction;
            endcase
    end
    endtask


    /* restart instructions */
    task restart;
    begin
        push2b(pc[15:8], pc[7:0]);
        case(ir[5:3])
            0: pc = 'h00; // RST 0
            1: pc = 'h08; // RST 1
            2: pc = 'h10; // RST 2
            3: pc = 'h18; // RST 3
            4: pc = 'h20; // RST 4
            5: pc = 'h28; // RST 5
            6: pc = 'h30; // RST 6
            7: pc = 'h38; // RST 7
        endcase
```

```
end
endtask


/* new instructions - except for NOP */
task newops;
    case(ir[5:3])
        0: ; // NOP

        4: // RIM
            begin
                acc = {sid, intmask[7:5], intmask[3:0]};
                if(trapi) begin
                    intmask[3] = inte;
                    trapi = 0;
                end
            end

        6: // SIM
            begin
                if(acc[3]) begin
                    intmask[2:0] = acc[2:0];
                    intmask[6:5] = intmask[6:5] & acc[1:0];
                end
                intmask[8] = acc[4];
                if(acc[6]) @ec1 @ec1 @ec2 sodff = acc[7];
            end

        default no_instruction;
    endcase
endtask


/* decode 1 instructions */
task decode1;
    case(ir[5:4])
        0: pop2b(pc[15:8], pc[7:0]); // RET
        2: pc = {regh, regl}; // PCHL
        3: sp = {regh, regl}; // SPHL
        default no_instruction;
    endcase
endtask


/* decode 2 instructions */
```

```
task decode2;
reg[7:0] saveh, savel;
    case(ir[5:3])
        0: adread(pc); // JMP

        2: // OUT
            begin
                memread(data, pc);
                pc = pc + 1;
                iowrite(data);
            end

        3: // IN
            begin
                memread(data, pc);
                pc = pc + 1;
                ioread(data);
            end

        4: // XTHL
            begin
                saveh = regh;
                savel = regl;
                pop2b(regh, regl);
                push2b(saveh, savel);
            end

        5: // XCHG
            begin
                saveh = regh;
                savel = regl;
                regh = regd;
                regl = rege;
                regd = saveh;
                rege = savel;
            end

        6: // DI, disable interrupt
            {intmask[6:5], intmask[3]} = 0;

        7: // EI, enable interrupt
            intmask[3] = 1;

        default no_instruction;
    endcase
```

```
endtask


/* decode 3 instructions */
task decode3;
    case(ir[5:4])
        0: // CALL
            begin :call
                reg [15:0] newpc;
                adread(newpc);
                push2b(pc[15:8], pc[7:0]);
                pc = newpc;
            end

        default no_instruction;
    endcase
endtask


/* fetch address from pc+1, pc+2 */
task adread;
output[15:0] address;
begin
    memread(address[7:0], pc);
    pc = pc + 1;
    memread(address[15:8], pc);
    if(!int) pc = pc + 1;
end
endtask


/* calculate cp cs and cz */
task calpsz;
input[7:0] tr;
begin
    cp = ^tr;
    cz = tr == 0;
    cs = tr[7];
end
endtask


/* undefined instruction */
task no_instruction;
begin
```