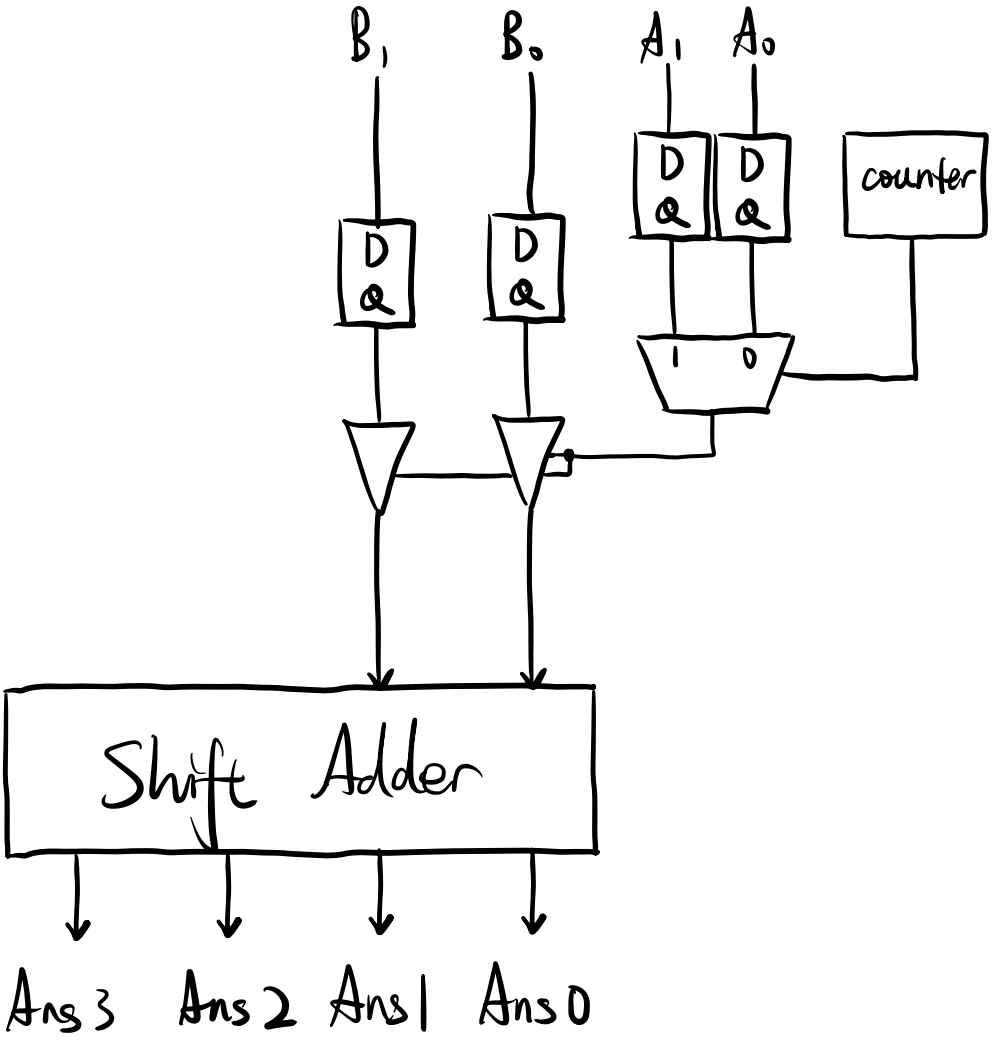
ECE3390 Assignment 3
Zhengzheng Yu
zyu322
251032911
Part 1 and 2

The block diagram
(clk is not shown, N=2 is used for simplicity. All flip-flops are DFFs.)



Shift Adder:

Current Output = Previous Output sll 1 + Input

The code

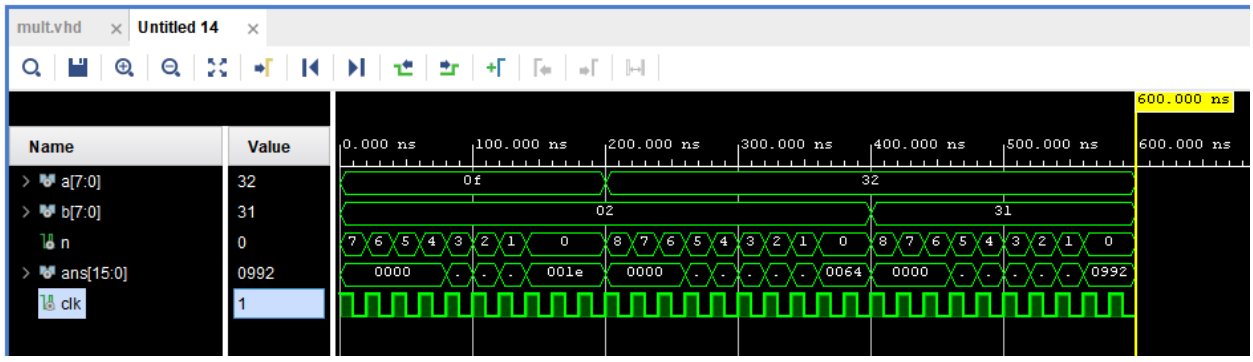
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
USE ieee.numeric_std.ALL;

entity mult is
end mult;

architecture Behavioral of mult is
signal a: unsigned(7 downto 0) := x"0f";
signal b: unsigned(7 downto 0) := x"02";
signal n: integer :=7;
signal ans : unsigned(15 downto 0) := x"0000";
signal clk : std_logic := '1';

begin
a<=x"32" after 200 ns;
b<=x"31" after 400 ns;
clk <= not clk after 10 ns;
process(clk,a,b) begin
 if (clk'event and clk='1') then
    case n is
    when 0=> if (a'event or b'event) then
                n<=8; end if;
    when others =>
    n<=n-1;
    end case;
end if; end process;
process(n)begin
    if (n=8)then ans<=x"0000";
    else if (a(n)='1')then
            ans<=ans+ans+b;
        else ans<=ans+ans;
        end if;
    end if;
end process;
END;
```

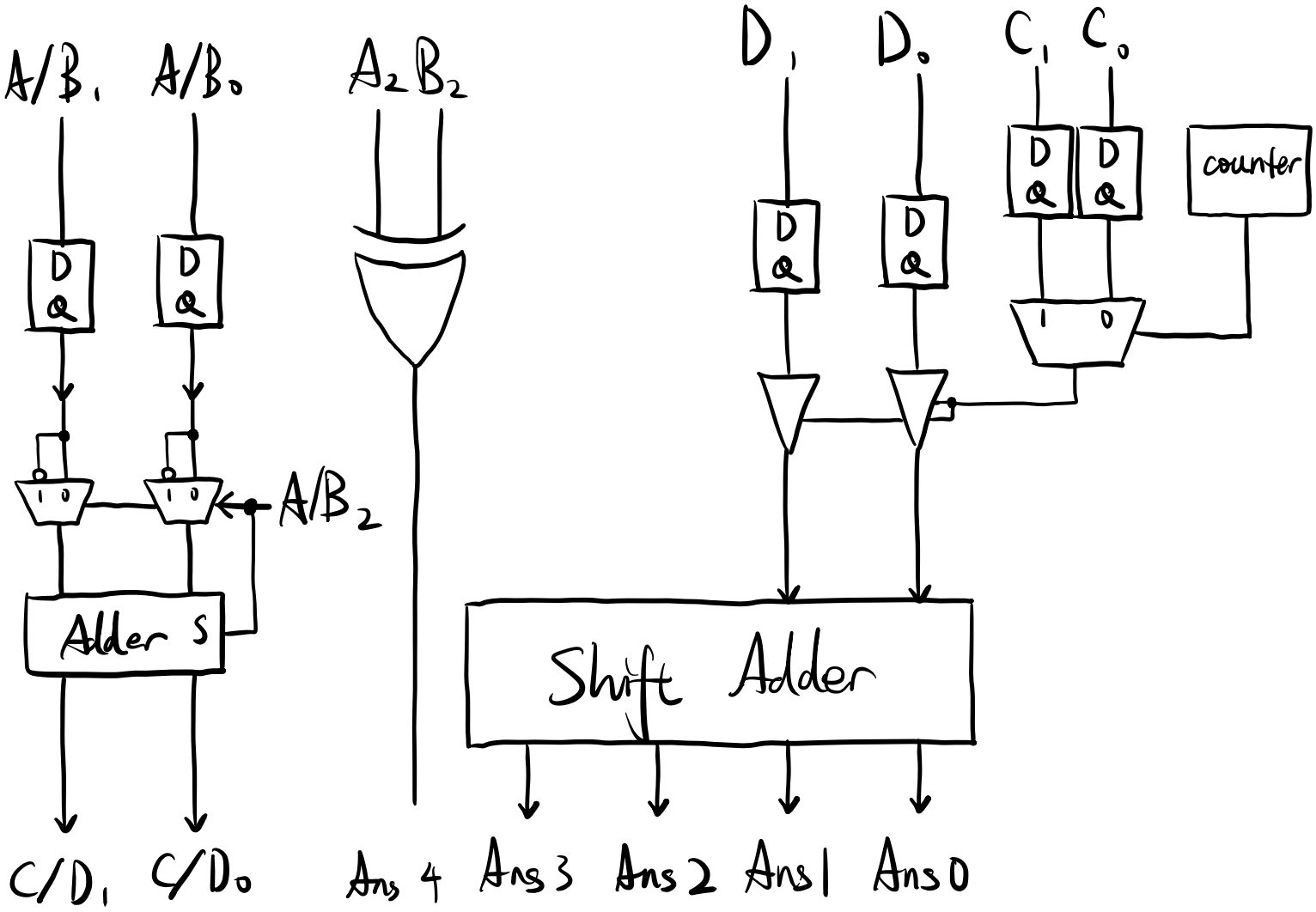
Simulation results



Pretty self explanatory, 0f*2=1e, 32*2=64,32*31=992.
It takes 1 reset cycle (n=8) and 7 shift&add cycles to complete multiplying.

Negative numbers
This multiplier cannot process 2'Comp negative numbers, it has to convert both numbers to it's absolute value first, multiply them, and output the sign separately.

The block diagram
(clk is not shown, N=3 is used for simplicity. All flip-flops are DFFs.)



Adder:

Adds S to the Input

Shift Adder:

Current Output = Previous Output sll 1 + Input

ECE3390 Assignment 3
Zhengzheng Yu
Rongyou Jiang

Part3

The state machine

The code

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY Proc9090tutorial2021sim IS
END Proc9090tutorial2021sim;
ARCHITECTURE aaa OF Proc9090tutorial2021sim IS
    type myArray is array (0 to 9) of unsigned(7 downto 0);
    signal Mem:myArray:=(
        "00011110", "00110010",  -- MVI E,32h
        "00111110", "00010100",  -- MVI A,14h
        "11011101", "00000000",
        "00000000", "00000000",
        "00000000", "00000000"
 --
 -- etc...  Your object code for your Prime number program goes here...
 --
    );
    type myRegs is array (0 to 7) of unsigned(7 downto 0);
    signal Reg:myRegs:=(x"00",x"00",x"00",x"00",x"00",x"00",x"00",x"00");
    signal clk : bit := '1'; -- start high, so clock rise will happen after simulation initializes
    signal PC,IR,Tc: unsigned(7 downto 0) :=x"00";
    signal Cy, Zf, Sf : unsigned(0 downto 0) := "0";

-- add any signals needed for your MUL and DIV here....

    signal mul: integer :=8;  --when 0, not doing multiplication;
    signal ans : unsigned(15 downto 0) := x"0000";--buffer for product



BEGIN
    clk <= not clk after 1 ps;
    process(clk)
        variable src,dst, dsrc,ddst : integer;
        variable tmp8: unsigned(7 downto 0);
        variable tmp17: unsigned(16 downto 0);
        begin
        if (clk'event and clk='1') then
            if mul=8 then Tc <= Tc+1;
            end if; -- cycle counter will increment on clock rise.  Important Note: LHS (after clock) <= RHS (before
clock)
```

```
            CASE Tc is
                when x"00" => IR <= Mem(to_integer(PC));   -- Fetch opcode in Tc "00" cycle
                              PC <= PC+1;
                when x"01" =>
                    src := to_integer(IR(2 downto 0));
                    dst := to_integer(IR(5 downto 3));
                    CASE IR(7 downto 6) is
                        when "00" =>
                            if IR(2 downto 0)="100" then Reg(dst)<=Reg(dst)+1; Zf<="0"; if Reg(dst)+1=x"00" then
Zf<="1"; end if; end if;  -- INR dst
                            if IR(2 downto 0)="101" then Reg(dst)<=Reg(dst)-1; Zf<="0"; if Reg(dst)-1=x"00" then
Zf<="1"; end if; end if;  -- DCR dst
                            if IR(5 downto 0)="010111" then
                                          Reg(7) <= Reg(7)(6 downto 0) & Cy;  --  RAL: Rotate Accumulator left through
carry bit Cy
                                          Cy<=Reg(7)(7 downto 7); -- set after clock
                                          end if;
                            if IR(5 downto 0)="011111" then
                                      Reg(7) <= Cy & Reg(7)(7 downto 1); --  RAR: Rotate Accumulator right through
carry bit Cy
                                      Cy<=Reg(7)(0 downto 0);
                                      end if;
                            if IR(3 downto 0)="1001" then                              -- DAD yy
                                      dsrc := to_integer(IR(5 downto 4))*2;
                                      tmp17 := ("0" & Reg(4) & Reg(5)) + ("0" & Reg(dsrc) & Reg(dsrc+1));
                                      Reg(5) <= tmp17(7 downto 0);
                                      Reg(4) <= tmp17(15 downto 8);
                                      Cy <= tmp17(16 downto 16);
                                      end if;
                            if IR(2 downto 0)="110" then Reg(dst) <= Mem(to_integer(PC));  --  MVI  dst,data8
                                                    PC <= PC+1;  end if;
                            if IR(3 downto 0)="0001" then
                                      ddst := to_integer(IR(5 downto 4))*2;
                                      Reg(ddst+1)<=Mem(to_integer(PC));       -- LXI
                                      PC <= PC+1; -- LXI doesn't clear Tc yet, since more cycles follow
                            else Tc <= x"00"; end if;
                        when "01" => Reg(dst) <= Reg(src);
                            Tc<=x"00";
                        when "10" =>
                            if(IR(5 downto 3)="000") then Reg(7)<=Reg(7)+Reg(src); end if;  -- ADD src  -- add register
"src" to accumulator
```

```
                        -- all of the other vanilla ALU operators go here...  But I found I didn't need them for this
assignment
                        -- the following line implements the CMP instruction:
                           if(IR(5 downto 3)="111") then tmp8:=Reg(7)-Reg(src); Sf<=tmp8(7 downto 7); Zf<="0";
if(tmp8=x"00") then Zf<="1"; end if; end if;
                           Tc<=x"00";
                        when "11" =>
                           if(IR(5 downto 0))="011101" then  --  multiply stuff goes here (does not reset Tc)

                                       -- first cycle of multiply goes here

                        elsif(IR(5 downto 0))="101101" then -- DIV stuff goes here (does not reset Tc)

                                       -- first cycle of your divider goes here

                        else
                           if IR(5 downto 0)="111110" then tmp8:=Reg(7)-Mem(to_integer(PC)); PC<=PC+1; Sf<=tmp8(7
downto 7); Zf<="0"; if(tmp8=x"00") then Zf<="1"; end if; end if; --CPI
                           if IR(5 downto 0)="101011" then Reg(4)<=Reg(2); Reg(2)<=Reg(4); Reg(5)<=Reg(3);
Reg(3)<=Reg(5); end if; -- XCHG
                           if IR(5 downto 0)="000011" then  PC<=Mem(to_integer(PC));  end if;
-- JMP addr8
                           if IR(5 downto 0)="010010" then if Cy="0" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JNC addr8
                           if IR(5 downto 0)="011010" then if Cy="1" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JC addr8
                           if IR(5 downto 0)="000010" then if Zf="0" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JNZ addr8
                           if IR(5 downto 0)="001010" then if Zf="1" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JZ addr8
                           if IR(5 downto 0)="110010" then if Sf="0" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JP addr8
                           if IR(5 downto 0)="111010" then if Sf="1" then PC<=Mem(to_integer(PC)); else PC<=PC+1; end
if; end if;  -- JM addr8
                           Tc<=x"00";     -- the Jump addresses on Proc9090 are 1-byte (not 2-bytes like on the 8080)
                         end if;
                      when others =>
                      end case;
                --when x"02" => ---  MUL, DIV, and LXI  will flow here when Tc="02"

                when others => ---  MUL, DIV, and LXI  will flow here when Tc="02" or higher
                CASE IR(7 downto 6) is
```

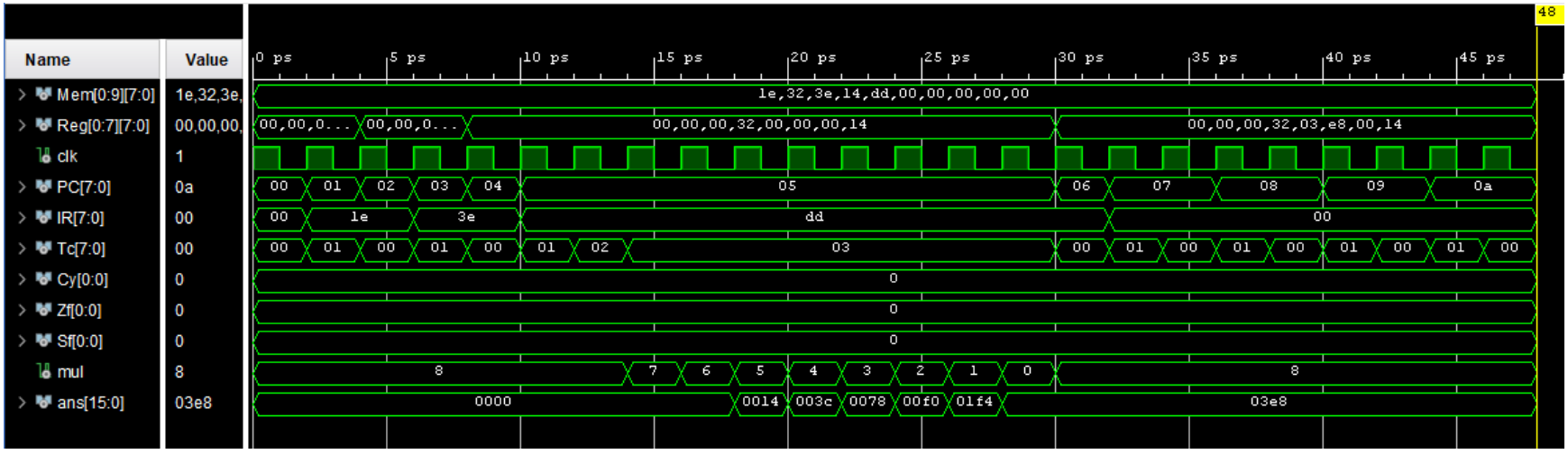
```
                    when "00" =>
                        if IR(3 downto 0)="0001" then
                                    Reg(ddst)<=Mem(to_integer(PC)); -- second byte of LXI (little Endian)
                                    PC <= PC+1;  end if;
                        Tc<=x"00"; -- now after second byte, can reset clock
                    when "11" =>
                        if(IR(5 downto 0))="011101" then  --  multiply stuff goes here (does not reset Tc)
                            if mul=0 then
                                mul<=8;
                                Reg(4)<=ans(15 downto 8);
                                Reg(5)<=ans(7 downto 0);
                                PC<=PC+1;
                                Tc<=x"00";
                            else mul<=mul-1; --dec counter
                            end if;
                        elsif(IR(5 downto 0))="101101" then -- DIV stuff goes here (does not reset Tc)

                       -- put the second cycle of your DIV implementation here

          --              Zf<="0"; if (remainder=x"00") then Zf<="1"; end if;    end if;   -- This line will set Zf to 1
if the remainder is zero
                        Tc<=x"00"; -- now can reset clock
                        end if;
                    when others =>
                end case;
             --when others =>
            end case;
        end if;
    end process;


    process(mul)
    begin
        if mul=8 then
        else
            if (Reg(3)(mul)='1')then
            ans<=ans+ans+Reg(7);
            else ans<=ans+ans;
            end if;
        end if;
    end process;
END;
```

The waveform



loads 32h into E and 24h into A
computes 32*24=3E8
load into HL