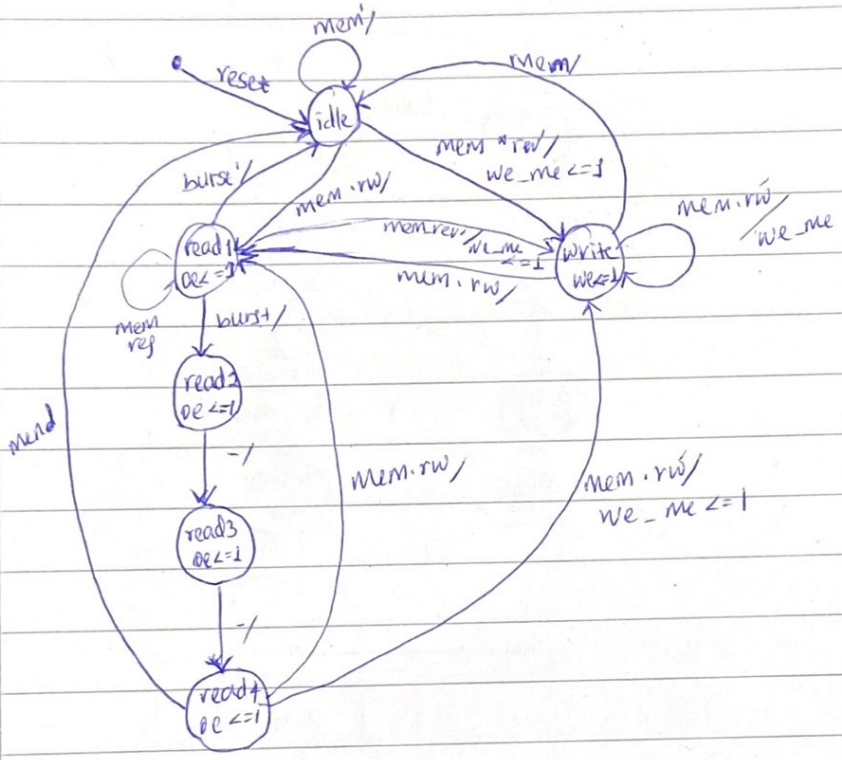
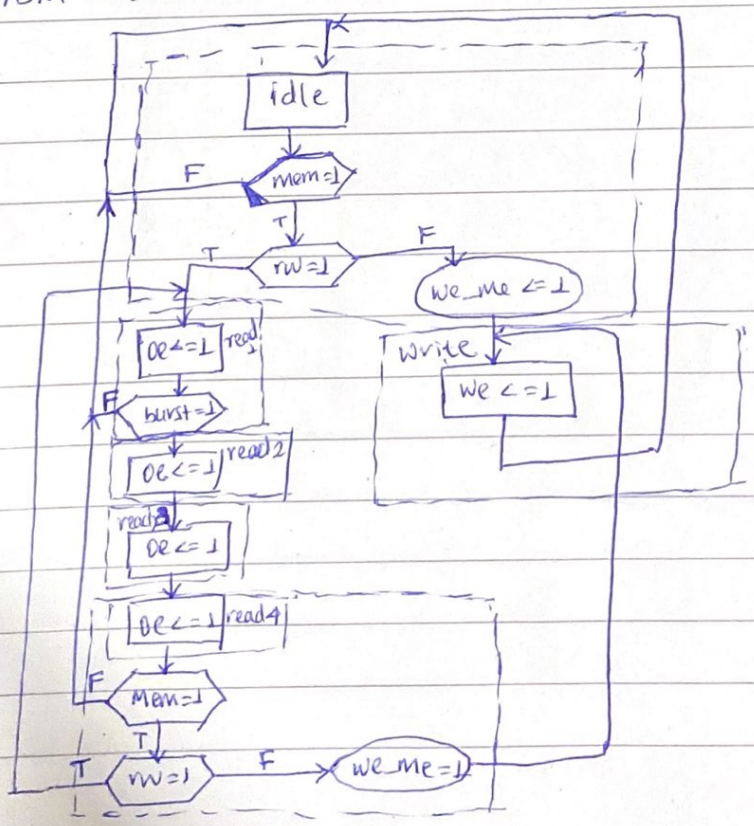
(1)

a) state diagram



b) ASM chart



1-c,

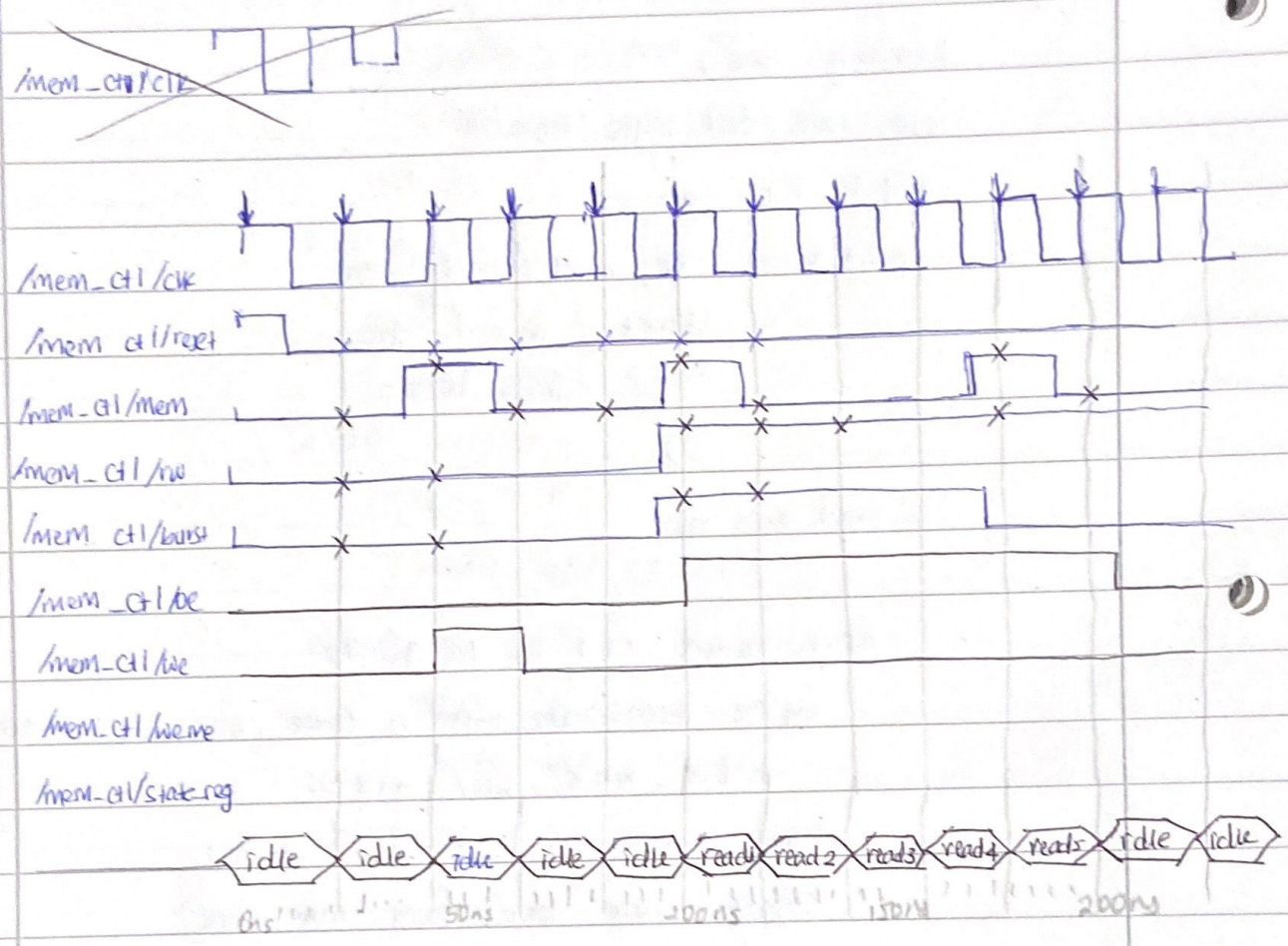
```
library ieee;
use iee.std_logic_1164.all;
entity mem_ctrl
    Port (
        clk, reset : in std_logic;
        mem : in std_logic;
        rw : in std_logic
        burst : in std_logic;
        oe : out std_logic;
        we : out std_logic;
        we_me : out std_logic;
        );
end entity
```

1-c

```
architecture two_seg_arch of mem_ctrl is
    type mc_state_type is
        (idle, read1, read2, read3, read4, write);
    signal state_reg, state_next : mc_state_type;
    begin
      process (clk, reset)
      begin
        if (reset = '1') then
            state_reg <= idle;
        else if (clk'event and clk = '1') then
            state_reg <= state_next;
        end if;
      end process;
      Process (state_reg, mem, rw, burst)
      begin
        oe <= '0';
        we <= '0';
        we_me <= '0';
        case state_reg is
          when idle =>
            if mem = '1' then
              if rw = '1' then
                state_next <= read1;
              else
                state_next <= write;
                we_me <= '1';
              end if;
            else
              stat_next <= idle;
            end if;
```

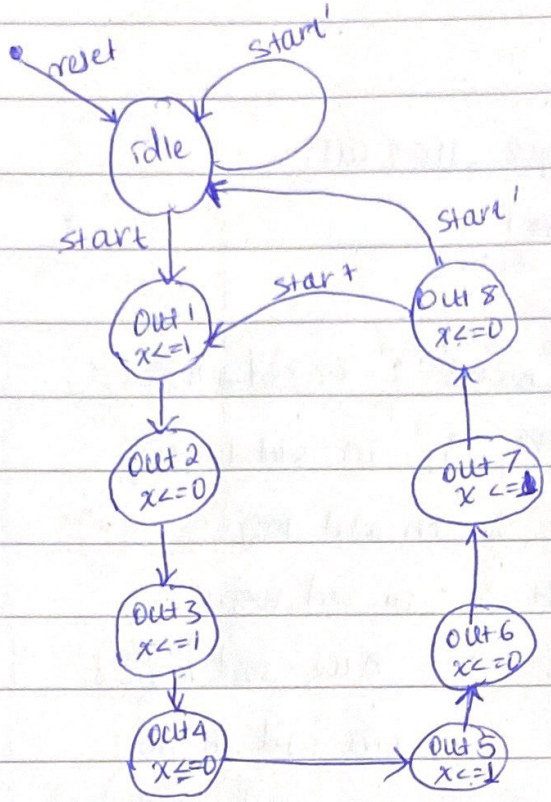
```
when write =>
        state_next <= idle;
        we <= '1';
when read1 =>
        if (burst = '1') then
            state_next <= read2;
        else
            state_next <= idle;
        end if
        oe <= '1';
when read2 =>
        state_next <= read3;
        oe <= '1';
when read3 =>
        state_next <= read4;
        oe <= '1';
when read4 =>
        state_next <= idle;
        oe <= '1';
    end case;
end process;
end two_seg_arch;
```

1. d)

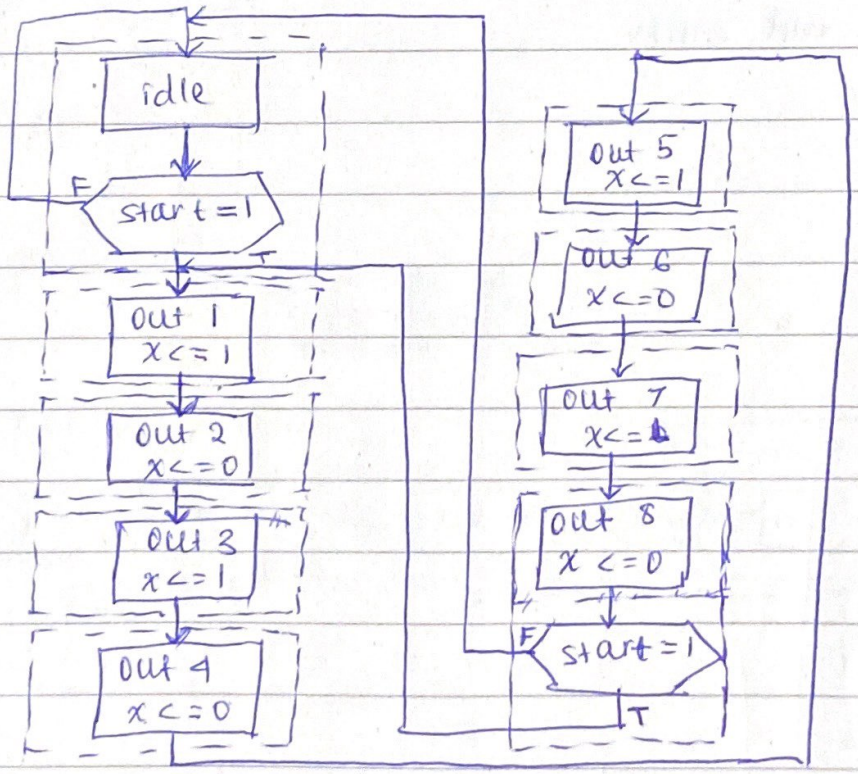

/mem_ctl/clk
/mem_ctl/reset
/mem_ctl/mem
/mem_ctl/rw
/mem_ctl/burst
/mem_ctl/be
/mem_ctl/we
/mem_ctl/wemo
/mem_ctl/state_reg
idle
idle
idle
idle
idle
read1
read2
read3
read4
read5
idle
idle
0ns
50ns
100ns
150ns
200ns

(2)

a)


reset
start'
idle
start
start'
start
Out 1
x<=1
Out 8
x<=0
Out 2
x<=0
Out 7
x<=1
Out 3
x<=1
Out 6
x<=0
Out 4
x<=0
Out 5
x<=1

b)


idle
F
start=1
T
Out 1
x<=1
Out 2
x<=0
Out 3
x<=1
Out 4
x<=0
Out 5
x<=1
Out 6
x<=0
Out 7
x<=1
Out 8
x<=0
F
start=1
T

2-c

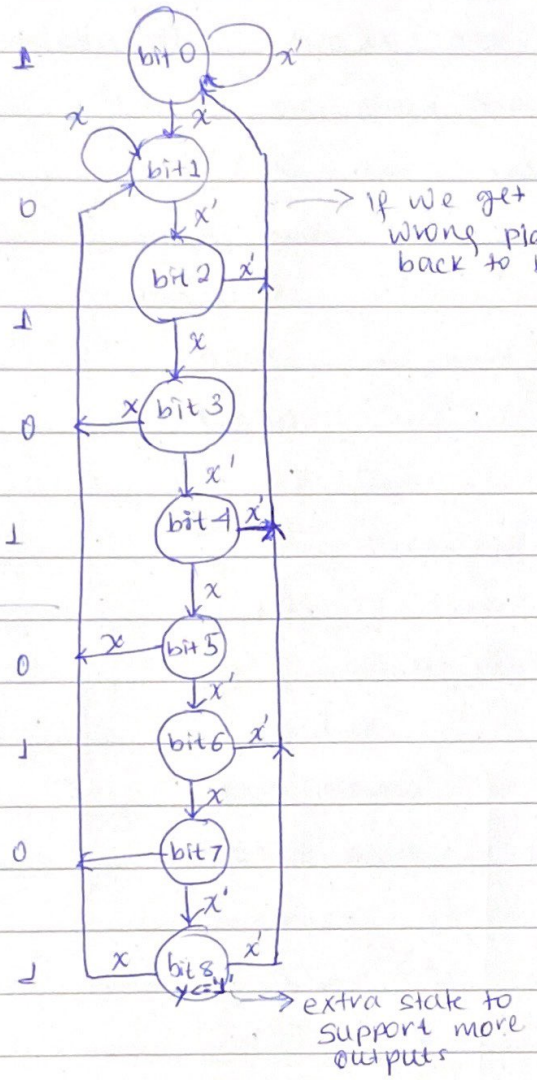
```
library ieee;
use ieee.std_logic_1164.all;
entity Patgen is
port ( clk, reset : in std_logic;
       start : in std_logic;
       V : out std_logic
     );
end entity;

architecture rtl of patgen is
  type fsm_state_type is (idle, bit1, bit2, bit3,
    bit5, bit6, bit7, bit8 );
  signal state_reg : fsm_state_type;
  signal state_next : fsm_state_type;
begin
  process (clk, reset)
  begin
    if (reset = '1') then
    state_reg <= idle;
    else if (clk'event and clk='1') then
      state_reg <= state_next;
    end if;
  end process;
  process (state_reg, start)
  begin
    x <= '0';
    case state_reg is
```

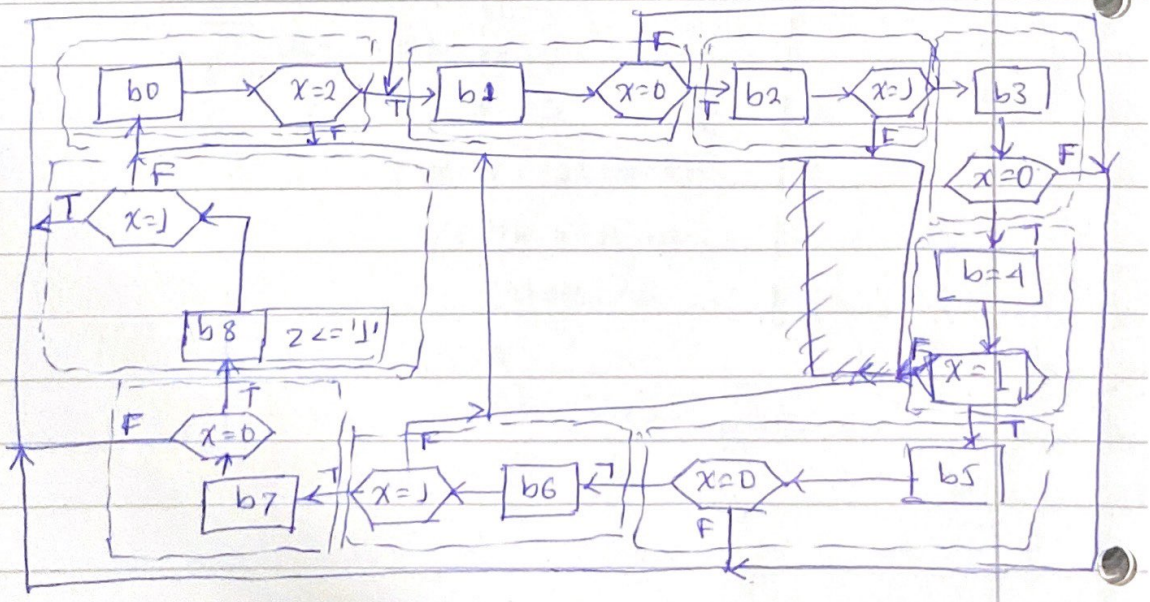
```
when idle =>
    if x = '1' then
        state_next <= bit1;
    end if;
when bit1 =>
    x <= '1';
    state_next <= bit2;
when bit2 =>
    x <= '0';
    state_next <= bit3;
when bit3 =>
    x <= '1';
    state_next <= bit4;
when bit4 =>
    x <= '0';
    state_next <= bit5;
when bit5 =>
    x <= '1';
    state_next <= bit6;
when bit6 =>
    x <= '0'
    state_next <= bit7;
when bit7 =>
    x <= '1';
    state_next <= bit8;
when bit8 =>
    x <= '0';
    if x = '1' then
        state_next <= bit1;
    else
        state_next <= bit0;
    end if; end case; end process;
```

(3)

a)


bit 0
bit 1
bit 2
bit 3
bit 4
bit 5
bit 6
bit 7
bit 8
y<='1'
x
x'
1
0
1
0
1
0
1
1
if we get 0 at the wrong place we go back to bit 0
If we get 1 in the wrong place we go back to bit 1
extra state to support more outputs

b)


b0
x=2
b1
x=0
b2
x=1
b3
x=0
b=4
x=1
b5
x=0
b6
x=1
b7
x=0
b8
z<='1'
x=1
T
F

3c -

```
library ieee;
use ieee.std_logic_1164.all;
entity patgen is
    port ( clk, reset : in std_logic;
           x : in std_logic;
           z : out std_logic;
         );
end entity

architecture rtl of patgen is
    type pattern_state_type is
      (idle, b1, b2, b3, b4, b5, b6, b7, b8);
    signal state_reg, state_next : pattern_state_type;

begin
    process (clk, reset)
    begin
        if (reset = '1') then
            state_reg <= idle;
        else if (clk'event and clk = '1') then
            state_reg <= state_next;
        end if;
    end process;
    process (state_reg, x)
```

```
begin
    case state_reg is
        when b0 =>
            if x='1' then
                state_next <= b1;
            else
                state_next <= b0;
            end if;
        when b1 =>
            if x='0' then
                state_next <= b2;
            else
                state_next <= b0;
            end if;
        when b2 =>
            if x='1' then
                state_next <= b3;
            else
                state_next <= b0;
            end if;
        when b3 =>
            if x='0' then
                state_next <= b4;
            else
                state_next <= b0;
            end if;
        when b4 =>
            if x='1' then
                state_next <= b5;
            else
                state_next <= b0;
            end if;
```

```
When b5 =>
    if x='0' then
        state_next <= b6;
    else
        state_next <= b0;
    end if;
When b6 =>
    if x='1' then
        state_next <= b7;
    else
        state_next <= b0;
    end if;
When b7 =>
    if x='0' then
        state_next <= b8;
    else
        state_next <= b0;
    end if;
When b8 =>
    if x='1' then
        state_next <= b1;
    else
        state_next <= b0;
    end if;
end case
end process
z <= '1' when state_reg = b8 else '0';
end rtl;
```