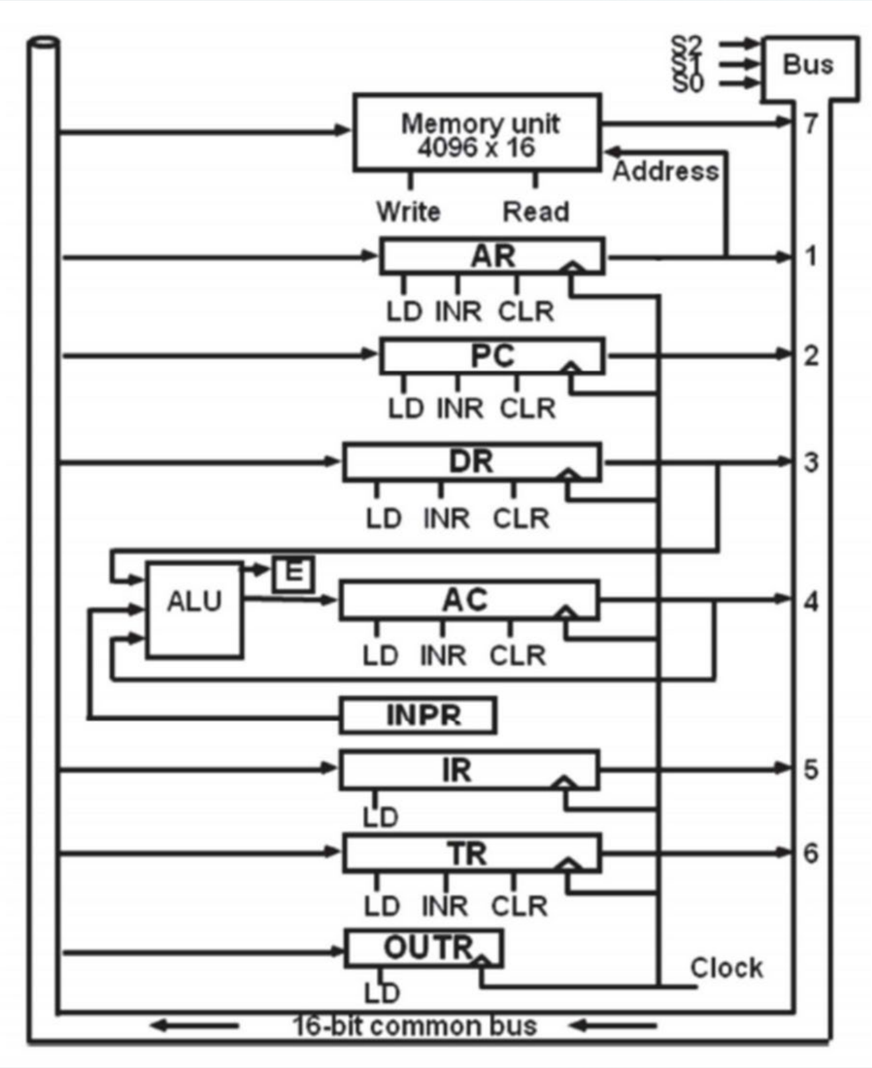
# MANO BASIC COMPUTER PROJECT

Eslam ahmed hosny
Mohamed Emad Abdelgilil
Basil Magdy Shrif

# INTRODUCTION




Register → STD_logic_vector

Flip flops → STD_logic

RAM -> array

# Register → STD_logic_vector

# OVER VIEW

```
    -- Main Registers
    signal PC  : STD_LOGIC_VECTOR(11 downto 0) := "000000000010";  -- PC = 2
    signal AR  : STD_LOGIC_VECTOR(11 downto 0) := (others => '0');
    signal IR  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
    signal DR  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
    signal AC  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
    signal OUTR_out: STD_LOGIC_VECTOR(7 downto 0)  := (others => '0');
    signal OUTR_in: STD_LOGIC_VECTOR(7 downto 0)  := (others => '0');
    signal INPR_in: STD_LOGIC_VECTOR(7 downto 0)  := (others => '0');
    signal INPR_out: STD_LOGIC_VECTOR(7 downto 0)  := (others => '0');
    signal TR  : STD_LOGIC_VECTOR(15 downto 0) := (others => '0');
```

```
-------------------------------------------------------------
 Main CPU Process
-------------------------------------------------------------
ocess(clk, reset)
  variable temp_add : unsigned(16 downto 0);
gin
```

# USE TR VARIPLE INSTEAD T
# FOR CARRY

## Flip flops →STD_logic

```
-- Flip-Flops
signal I   : STD_LOGIC := '0';
signal E   : STD_LOGIC := '0';
signal R   : STD_LOGIC := '0';
signal IEN : STD_LOGIC := '0';
signal FGI : STD_LOGIC := '0';
signal FGO : STD_LOGIC := '0';
```

# RAM ->array

```
type mem_type is array(0 to 4095) of STD_LOGIC_VECTOR(15 downto 0);
signal RAM : mem_type := (

    0   => x"0000",
    1   => x"4032",   -- (Example: Initialization or unused)
    2   => x"203E",   -- LDA (Load Accumulator)

   => x"7001", ----HLT
    -- Interrupt codes
    50  => x"F800",
    51  => x"F400",
    52  => x"F200",
    53  => x"F100",
    54  => x"F080",   -- Duplicate value, but index is unique
    55  => x"4000",   -- Moved from conflicting index 9
      -- Data for testing
      60  => x"00FF",   -- Data for AND
      61  => x"0001",   -- Data for ADD
      62  => x"00AA",   -- Data for LDA
      63  => x"5100",   -- BSA
      256 => x"0000",   -- Store PC -- 100
      257 => x"6130",   -- 101 = PC
      259 => x"7001",   -- HLT (Halt)
      304 => x"FFFF",


      others => (others => '0')
```

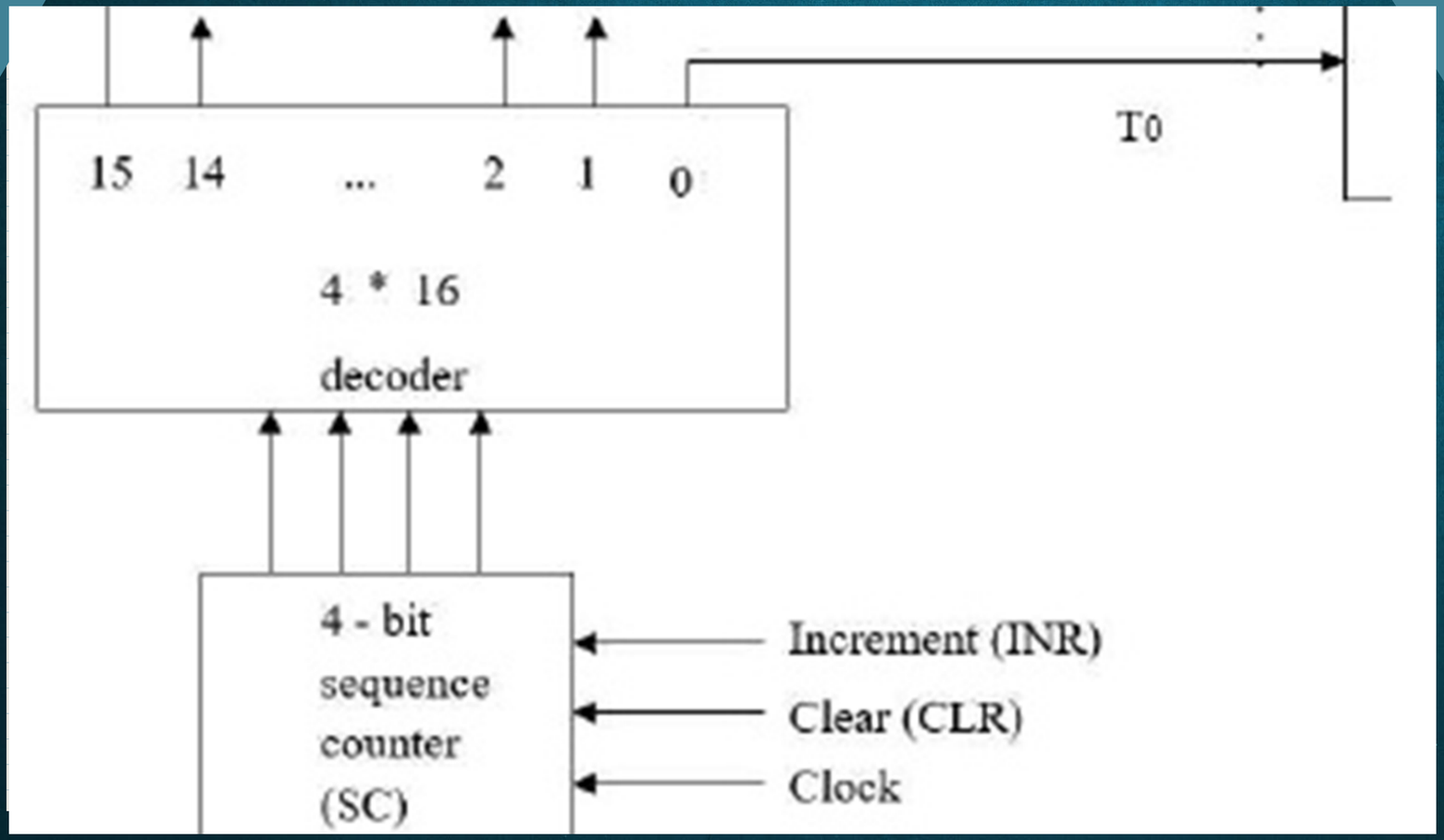
# OVER VIEW

# OVER VIEW

Control signals

```
-- Flip-Flops
signal I   : STD_LOGIC := '0';
signal E   : STD_LOGIC := '0';
signal R   : STD_LOGIC := '0';
signal IEN : STD_LOGIC := '0';
signal FGI : STD_LOGIC := '0';
signal FGO : STD_LOGIC := '0';
```

# COUNTER AND DECODER



# COUNTER

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity counter is
    Port (
        clk : in STD_LOGIC;
        sc  : in STD_LOGIC;
        T   : out STD_LOGIC_VECTOR(3 downto 0)
    );
end counter;

architecture Behavioral of counter is
    signal count : STD_LOGIC_VECTOR(3 downto 0) := "0000";
```

```vhdl
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if sc = '0' or count = "1111" then
                count <= (others => '0');
            else
                count <= std_logic_vector(unsigned(count) +
            end if;
        end if;
    end process;

    T <= count;
end Behavioral;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Decoder4_16 is
    port(
        I      : in  std_logic_vector(3 downto 0);
        Output: out std_logic_vector(15 downto 0)
    );
end Decoder4_16;

architecture arch_Decoder4_16 of Decoder4_16 is
begin
    process(I)
    begin
        Output <= (others => '0');
```

# DECODER

```vhdl
    case I is
        when "0000" => Output <= "0000000000000001";
        when "0001" => Output <= "0000000000000010";
        when "0010" => Output <= "0000000000000100";
        when "0011" => Output <= "0000000000001000";
        when "0100" => Output <= "0000000000010000";
        when "0101" => Output <= "0000000000100000";
        when "0110" => Output <= "0000000001000000";
        when "0111" => Output <= "0000000010000000";
        when "1000" => Output <= "0000000100000000";
        when "1001" => Output <= "0000001000000000";
        when "1010" => Output <= "0000010000000000";
        when "1011" => Output <= "0000100000000000";
        when "1100" => Output <= "0001000000000000";
        when "1101" => Output <= "0010000000000000";
        when "1110" => Output <= "0100000000000000";
        when "1111" => Output <= "1000000000000000";
        when others => Output <= (others => '0');
    end case;
end process;
end arch_Decoder4_16;
```

```
component counter is
        Port (
            clk : in STD_LOGIC;
            sc  : in STD_LOGIC;
            T   : out STD_LOGIC_VECTOR(3 downto 0)
        );
    end component;

    -- Decoder component declaration
    component Decoder4_16 is
        port(
            I     : in  std_logic_vector(3 downto 0);
            Output: out std_logic_vector(15 downto 0)
        );
    end component;
```

```
-- Instantiate the 4-bit counter
Timing_Generator: counter
    port map(
        clk => clk,
        sc => SC,
        T => T_counter_4bit
    );
    -- Instantiate the decoder
    Decoder_inst: Decoder4_16
        port map(
            I => T_counter_4bit,
            Output => T
        );
```

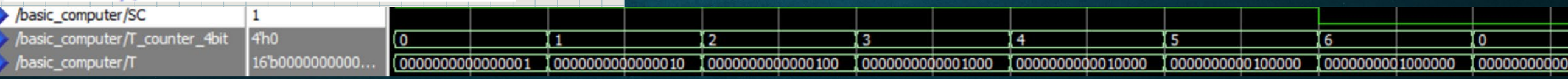
```
    -- Timing and Control
    signal SC    : unsigned(3 downto 0) := "0000";
    signal D     : STD_LOGIC_VECTOR(7 downto 0);

begin
```

# COUNTER AND DECO

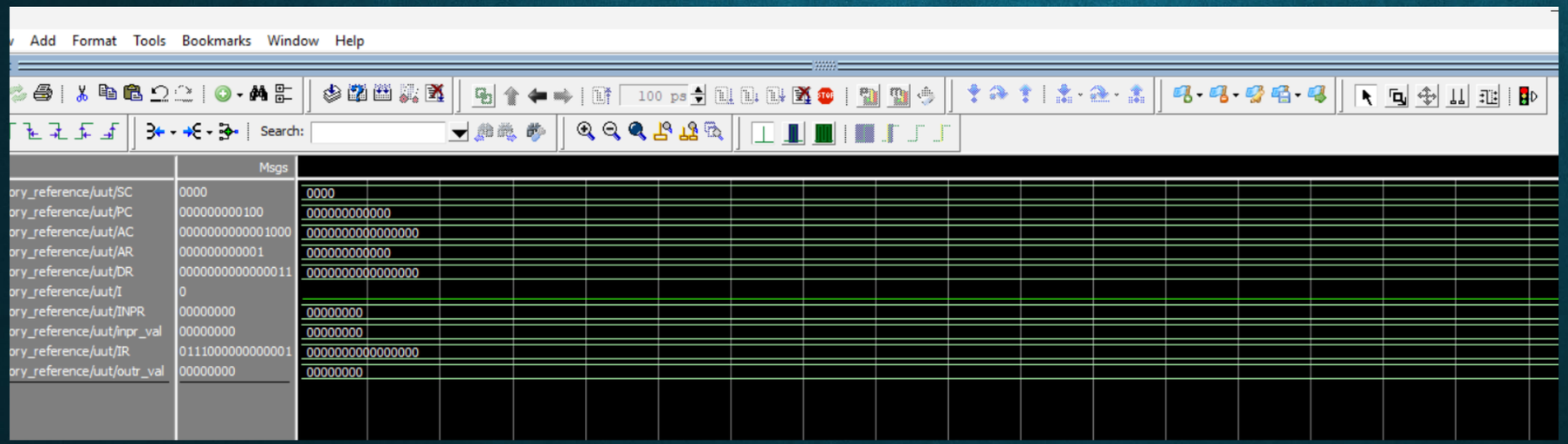
# REPLACE

# THIS WITH SIGNAL; SC



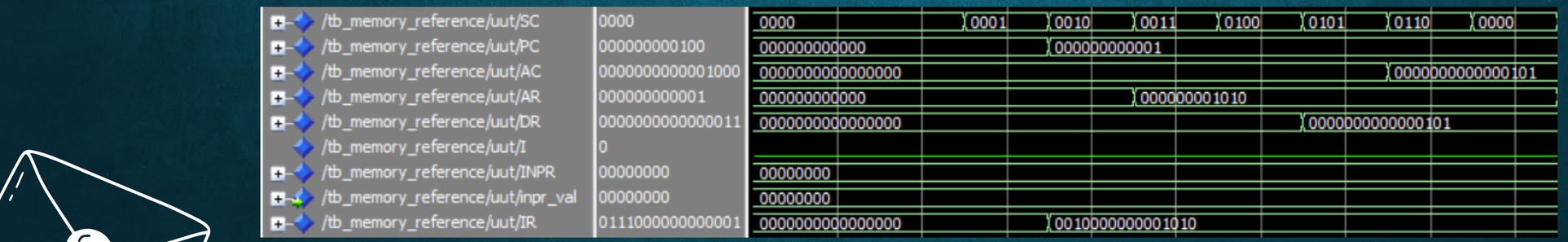
```
if reset = '1' then
    SC <= '0';
    PC <= "000000000010";
    AC <= (others => '0');
    AR <= (others => '0');
    IR <= (others => '0');
    DR <= (others => '0');
    TR <= (others => '0');
    OUTR_out <= (others => '0');
    OUTR_in <= (others => '0');
    INPR_out <= (others => '0');
    INPR_in <= (others => '0');
    temp <= (others => '0');
    I <= '0'; E <= '0'; R <= '0';
    IEN <= '0'; FGI <= '0'; FGO <= '0';
    D <= (others => '0');
    run <= true;
```

# RESET



# NORMAL INSTRUCTION CYC

```
elsif T(0) = '1' then
    AR <= PC;
    SC<='1';
elsif T(1) = '1' then
    IR <= RAM(to_integer(unsigned(AR)));
    PC <= std_logic_vector(unsigned(PC) + 1);
    SC<='1';
elsif T(2) = '1' then
    D <= IR(14 downto 12);
    AR <= IR(11 downto 0);
    I <= IR(15);
    SC<='1';
```



# Memory reference instructions

## AND

```
when "0101" =>
    if D(0) = '1' then AC <= AC and DR; -- AND
```

## ADD

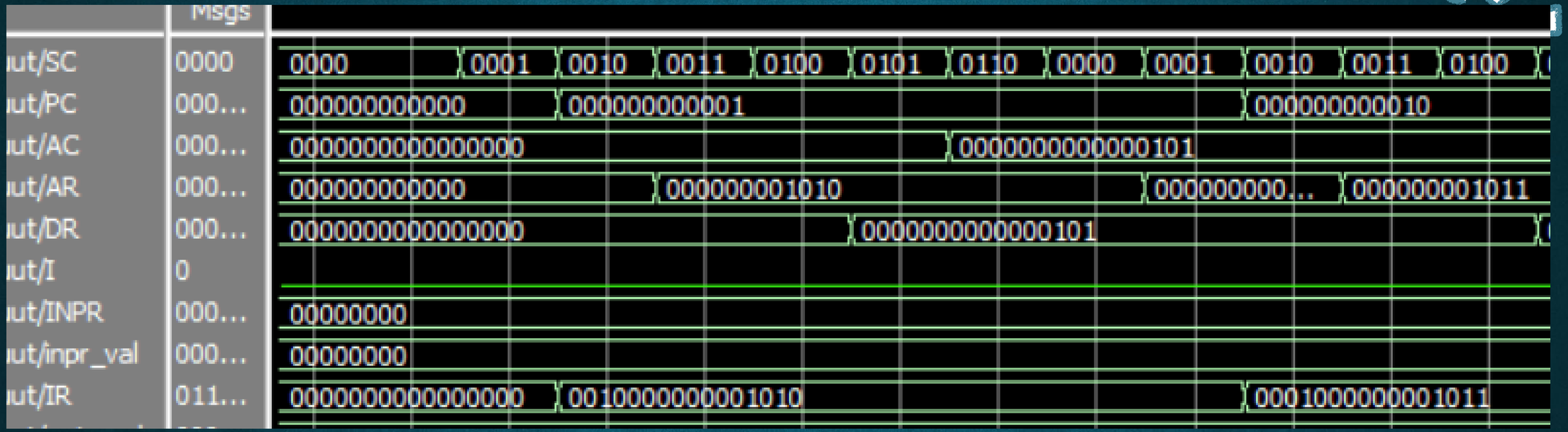
```
if D(1) = '1' then -- ADD
 temp_add := unsigned('0' & AC) + unsigned('0' & DR);
 AC <= std_logic_vector(temp_add(15 downto 0));
 E <= temp_add(16);
```

## LDA

```
elsif D(2) = '1' then AC <= DR; -- LDA
```

## STA

```
if D(0)='1' or D(1)='1' or D(2)='1' or D(6)='1' then
    DR <= RAM(to_integer(unsigned(AR)));
elsif D(3) = '1' then -- STA
    RAM(to_integer(unsigned(AR))) <= AC;
    SC <= "0000";
```


Msgs
ut/SC
0000
0001
0010
0011
0100
0101
0110
0000
0001
0010
0011
0100
ut/PC
000...
000000000000
000000000001
000000000010
ut/AC
000...
0000000000000000
0000000000000101
ut/AR
000...
000000000000
000000001010
000000000...
000000001011
ut/DR
000...
0000000000000000
0000000000000101
ut/I
0
ut/INPR
000...
00000000
ut/inpr_val
000...
00000000
ut/IR
011...
0000000000000000
0010000000001010
0001000000001011

# MEMORY REFERENCE INSTRUCTIO

## BUN

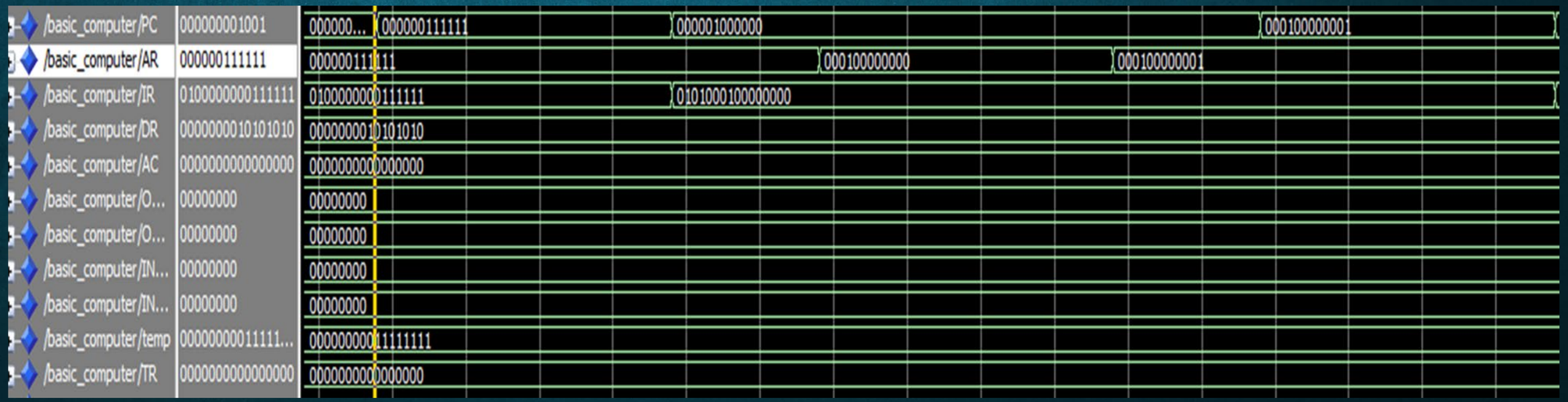
```
when "100" => -- BUN
        PC <= AR;
        SC<='0';
```

# MEMORY REFERENCE INSTRUCTI

## BSA

```
when "101" => -- BSA
    RAM(to_integer(unsigned(AR))) <= "0000" & PC ;
    AR <= std_logic_vector(unsigned(AR) + 1);
    SC<='1';

    PC <= AR;
    SC<='0';
```

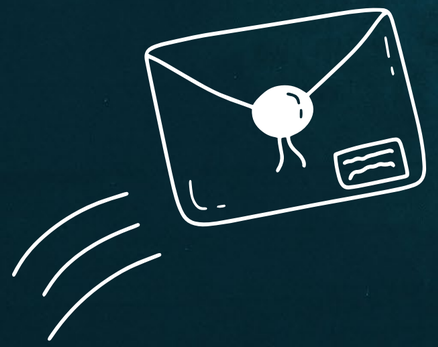


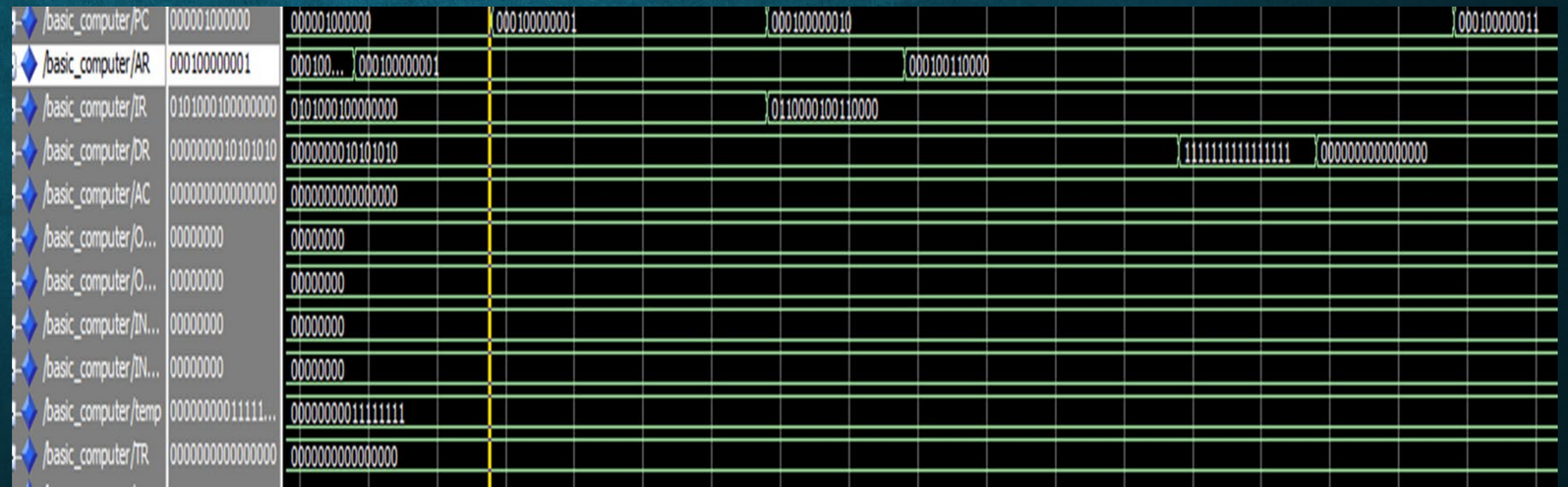
# MEMORY REFERENCE INSTRUCTIO

## ISZ

```
elsif D(6) = '1' then -- ISZ
    DR <= std_logic_vector(unsigned(DR) + 1);
end if;

-- T6: Finalize ISZ
when "0110" =>
    if D(6) = '1' then
        RAM(to_integer(unsigned(AR))) <= DR;
        if DR = x"0000" then
            PC <= std_logic_vector(unsigned(PC) + 1);
        end if;
    end if;
    SC <= "0000";
```





# REGISTER REFERENCE

## CLEAR AC

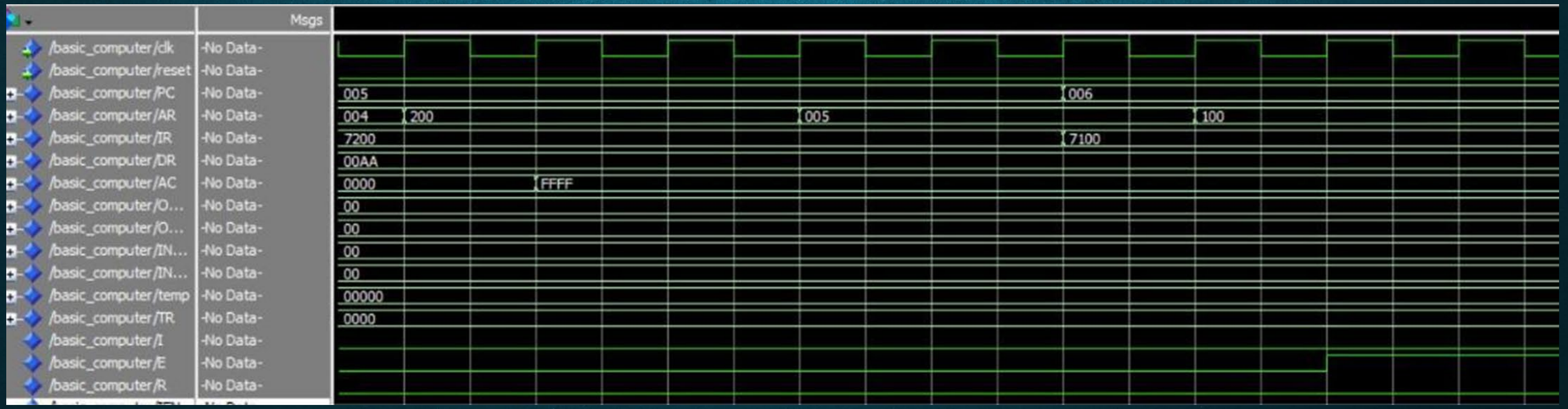
```
-- Register Reference and I/O Instructions
elsif T(3) = '1' then
    if I = '0' and D = "111" then
        -- Register reference instructions
        if IR(11) = '1' then
            AC <= (others => '0'); -- CLA
```

# COMPLEMENT AC

```
elsif IR(9) = '1' then
    AC <= not AC; -- CMA
```

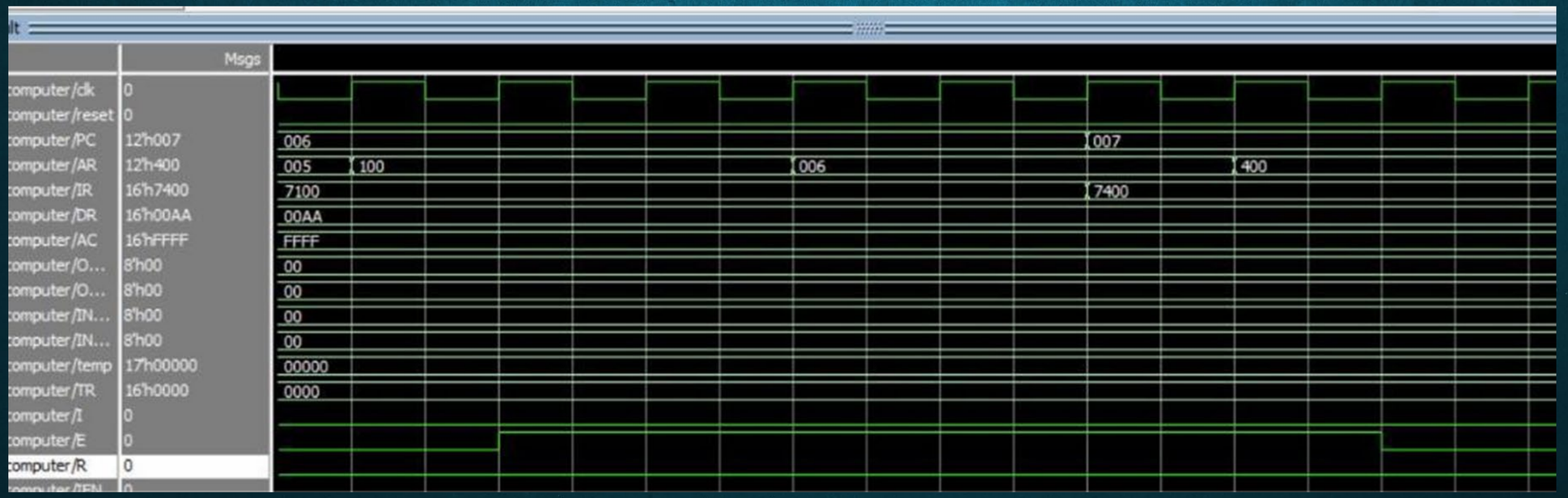
# COMPLEMENT E

```
elsif IR(8) = '1' then
    E <= not E; -- CME
```



# CLEAR E

```
elsif IR(10) = '1' then
    E <= '0'; -- CLE
```

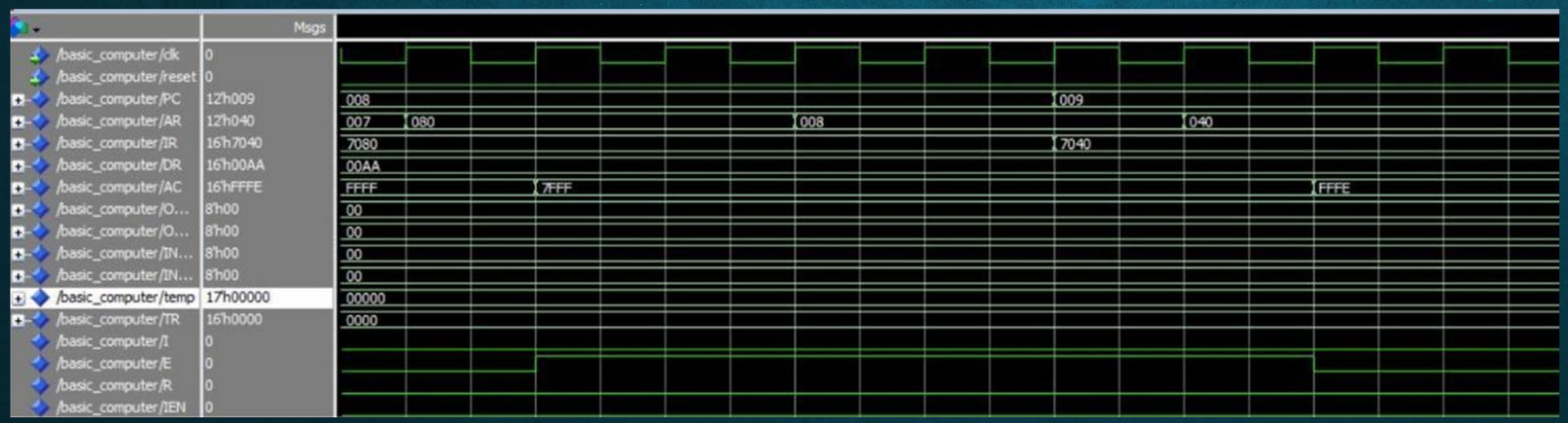


# CIRCULATE RIGHT AC

```
if IR(7)   = '1' then -- SHR
     E <= AC(0); AC <= E & AC(15 downto 1);
end if;
```

# CIRCULATE LEFT AC

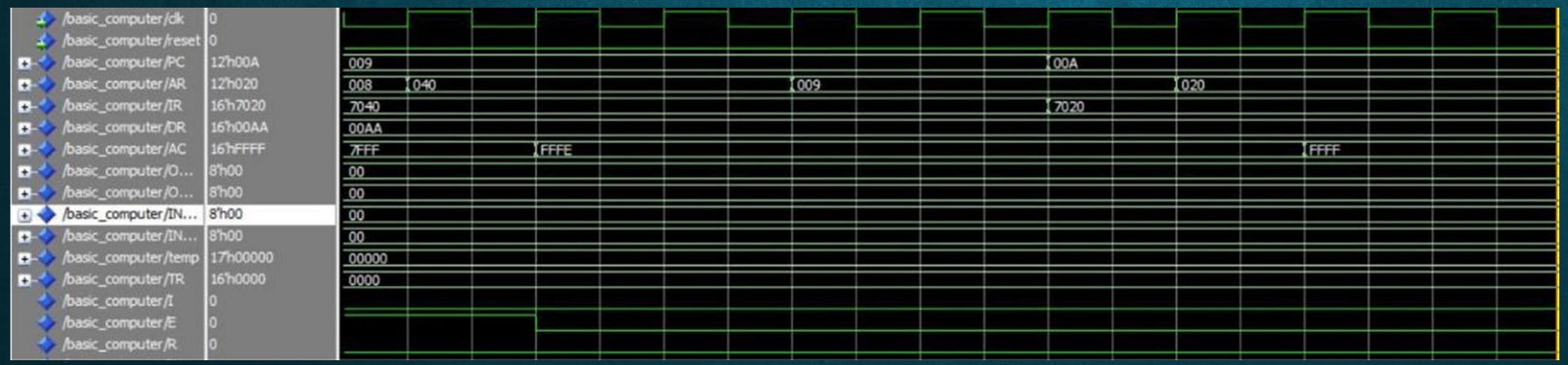
```
        E <= AC(0); AC <= E & AC(15 downto 1);
    end if;
    if IR(6)  = '1' then -- SHL
        E <= AC(15); AC <= AC(14 downto 0) & E;
    end if;
```



# INCREMENT

## AC

```
elsif IR(5) = '1' then
    AC <= std_logic_vector(unsigned(AC) + 1); -- INC
```

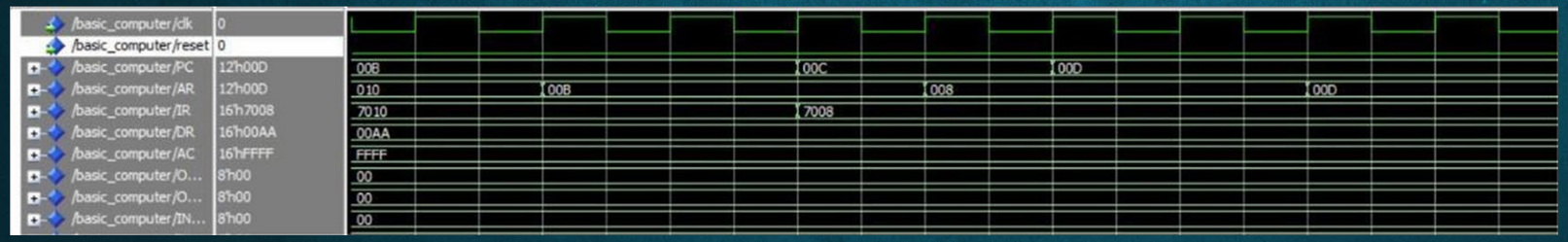


# SKIP IF AC (+)

```
elsif IR(4) = '1' then
    if AC(15) = '0' and AC /= x"0000" then
        PC <= std_logic_vector(unsigned(PC) + 1); -- SPA
    end if;
```

# SKIP IF AC(

```
elsif IR(3) = '1' then
    if AC(15) = '1' then
        PC <= std_logic_vector(unsigned(PC) + 1); -- SNA
    end if;
```

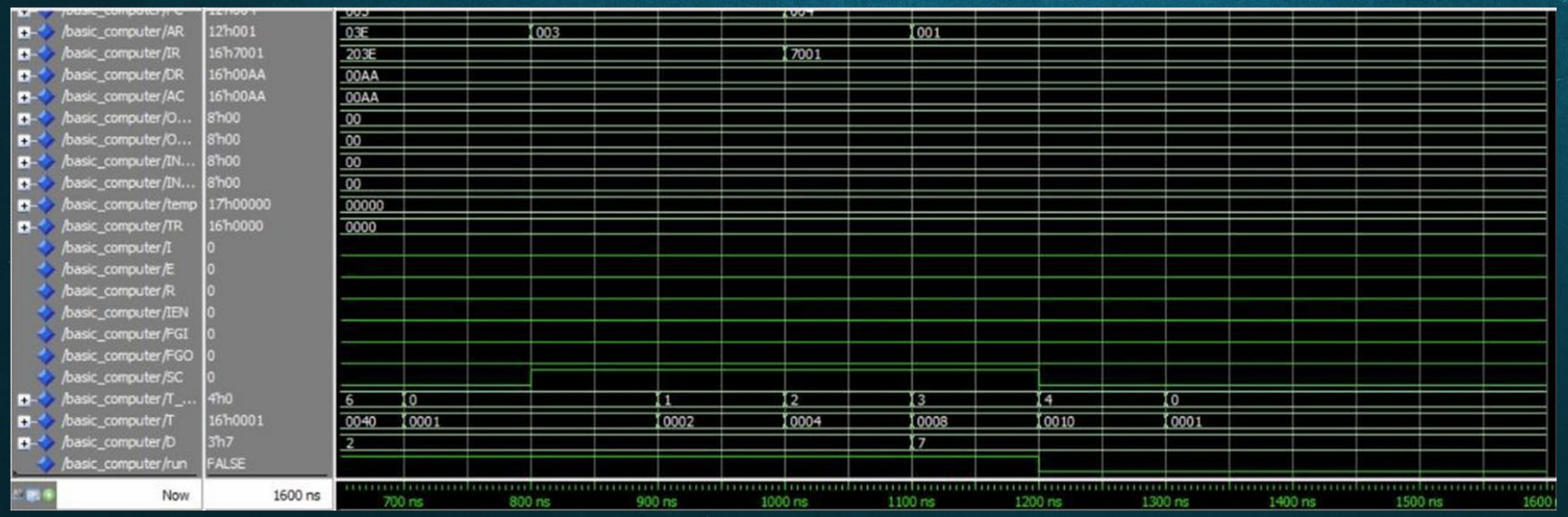


# SKIP IF E=0

```
elsif IR(1) = '1' then
    if E = '0' then
        PC <= std_logic_vector(unsigned(PC) + 1); -- SZE
    end if;
```

# HLT

```
elsif IR(0) = '1' then
    run <= false; -- HLT
end if;
```



# REGISTER REFERE
# INSTRUCTIONS

```
-- Register Reference and I/O Instructions
elsif T(3) = '1' then
    if I = '0' and D = "111" then
        -- Register reference instructions
        if IR(11) = '1' then
            AC <= (others => '0'); -- CLA
        elsif IR(10) = '1' then
            E <= '0'; -- CLE
        elsif IR(9) = '1' then
            AC <= not AC; -- CMA
        elsif IR(8) = '1' then
            E <= not E; -- CME
        elsif IR(5) = '1' then
            AC <= std_logic_vector(unsigned(AC) + 1); -- INC
        elsif IR(6) = '1' then
            E <= AC(15);
            AC <= AC(14 downto 0) & '0'; -- SHL
        elsif IR(7) = '1' then
            E <= AC(0);
            AC <= '0' & AC(15 downto 1); -- SHR
```

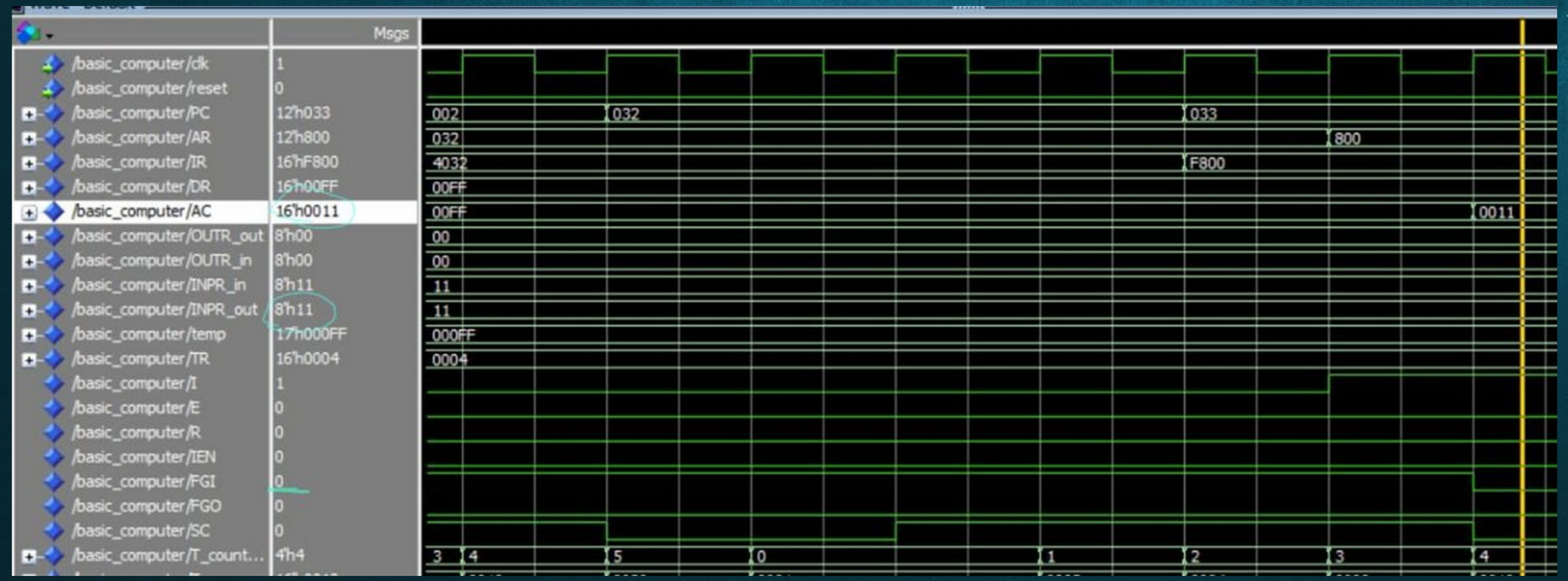
```
        elsif IR(4) = '1' then
            if AC(15) = '0' and AC /= x"0000" then
                PC <= std_logic_vector(unsigned(PC) + 1); -- SPA
            end if;
        elsif IR(3) = '1' then
            if AC(15) = '1' then
                PC <= std_logic_vector(unsigned(PC) + 1); -- SNA
            end if;
        elsif IR(2) = '1' then
            if AC = x"0000" then
                PC <= std_logic_vector(unsigned(PC) + 1); -- SZA
            end if;
        elsif IR(1) = '1' then
            if E = '0' then
                PC <= std_logic_vector(unsigned(PC) + 1); -- SZE
            end if;
        elsif IR(0) = '1' then
            run <= false; -- HLT
        end if;
```

# INPUT OUTPUT INSTRUCTIONS

1 )
INP

| | | | | | |
|---|---|---|---|---|---|
| 30 | INP | F800 | AC(7:0) ← INPR; FGI ← 0 | T3 | AC lower 8 bits updated, FGI cleared |

```
-- I/O instructions
if IR(11) = '1' then
    AC <= AC(15 downto 8) & INPR_out;
    FGI <= '0'; -- INP
```

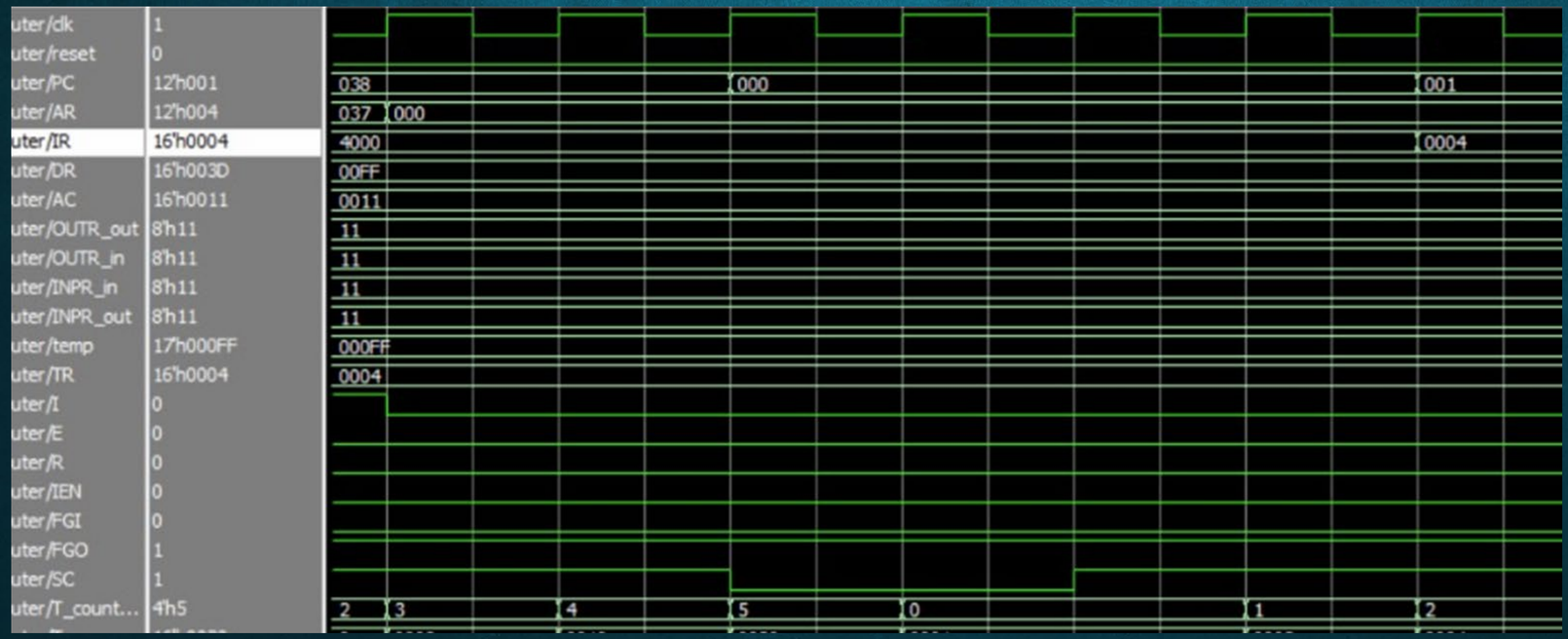



# INPUT OUTPUT INSTRUCTIONS

2 )
OUT

| | | | | | |
|---|---|---|---|---|---|
| 31 | OUT | F400 | OUTR ← AC(7:0); FGO ← 0 | T3 | OUTR updated, FGO cleared |

```
elsif IR(10) = '1' then
    OUTR_in <= AC(7 downto 0);
    FGO <= '0'; -- OUT
```

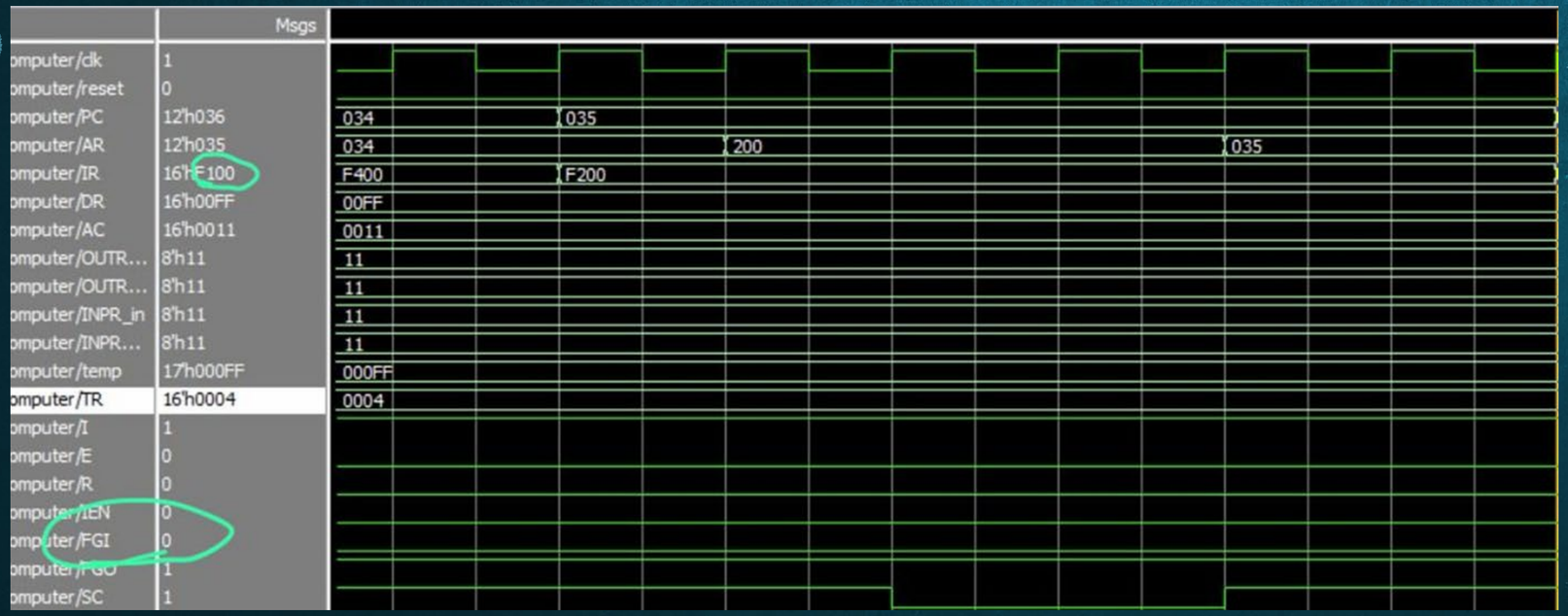



# INPUT OUTPUT INSTRUCTIONS

## 3 ) SKI

| | | | | | |
|---|---|---|---|---|---|
| 32 | SKI | F200 | if FGI=1, PC++ | T3 | PC incremented if FGI=1 |

```
if IR(9)  = '1' and FGI = '1' then PC <= std_logic_vector(unsigned(PC)+1); end if;
if IR(8)  = '1' and FGO = '1' then PC <= std_logic_vector(unsigned(PC)+1); end if;
```




IR 9 =1

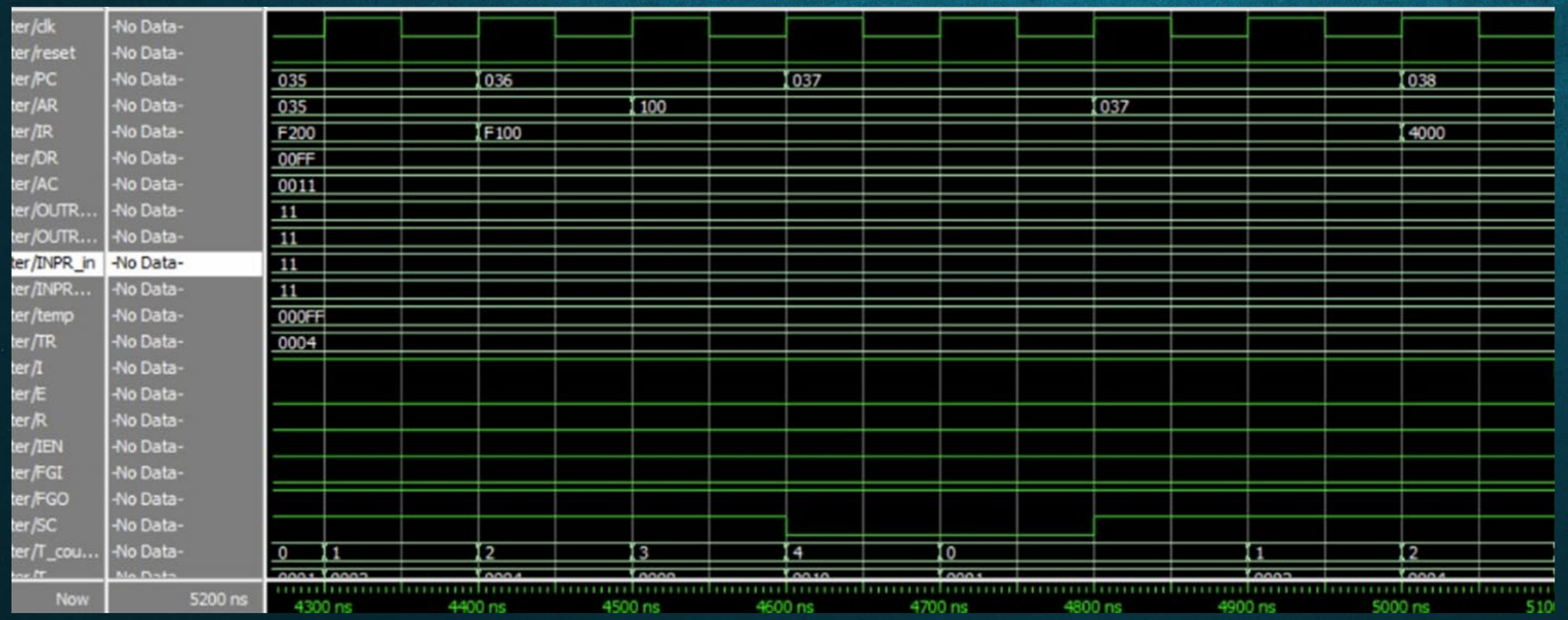
BUT FGI NOT EQUAL 1 NO SKI

# INPUT OUTPUT INSTRUCTIONS

## 4 ) SKO

| | | | | | |
|---|---|---|---|---|---|
| 33 | SKO | F100 | if FGO=1, PC++ | T3 | PC incremented if FGO=1 |

```
if IR(9)  = '1' and FGI = '1' then PC <= std_logic_vector(unsigned(PC)+1); end if;
if IR(8)  = '1' and FGO = '1' then PC <= std_logic_vector(unsigned(PC)+1); end if;
```



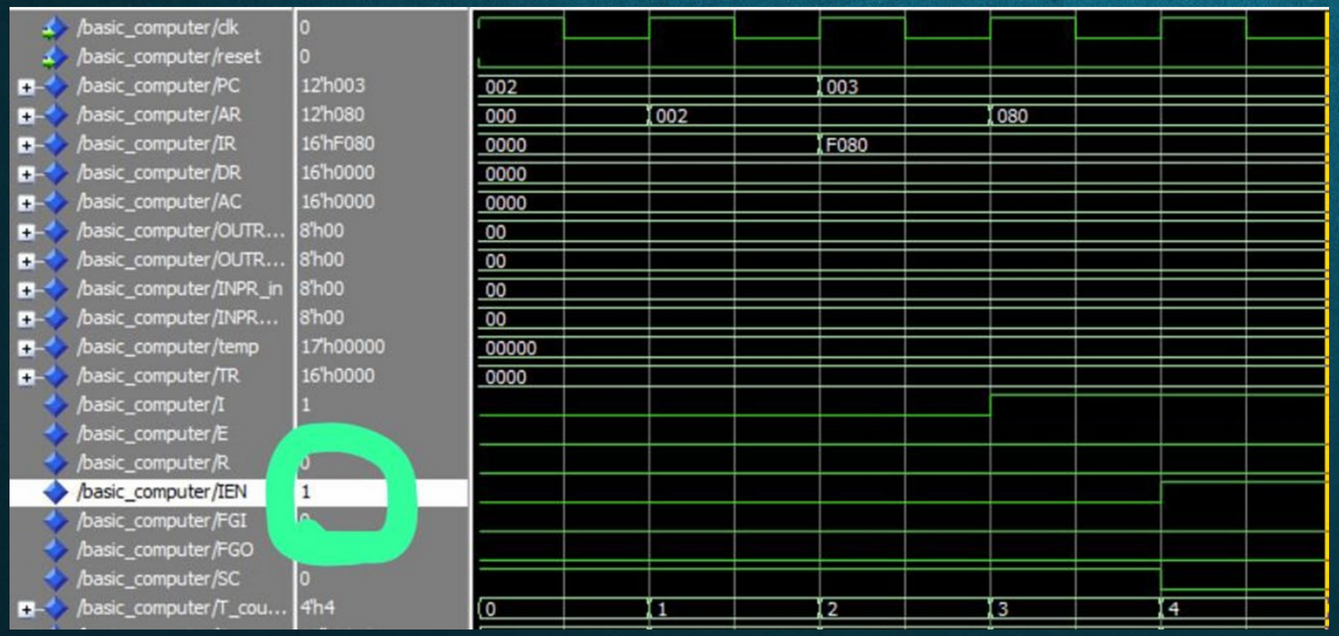

IR 8=1
FGO=1
SO PC=PC+1=037

# INPUT OUTPUT INSTRUCTIONS

5 )
I->ON

| | | | | |
|---|---|---|---|---|
| ION | F080 | IEN ← 1 | T3 | Interrupts enabled |

```
elsif IR(7) = '1' then
    IEN <= '1'; -- ION
```



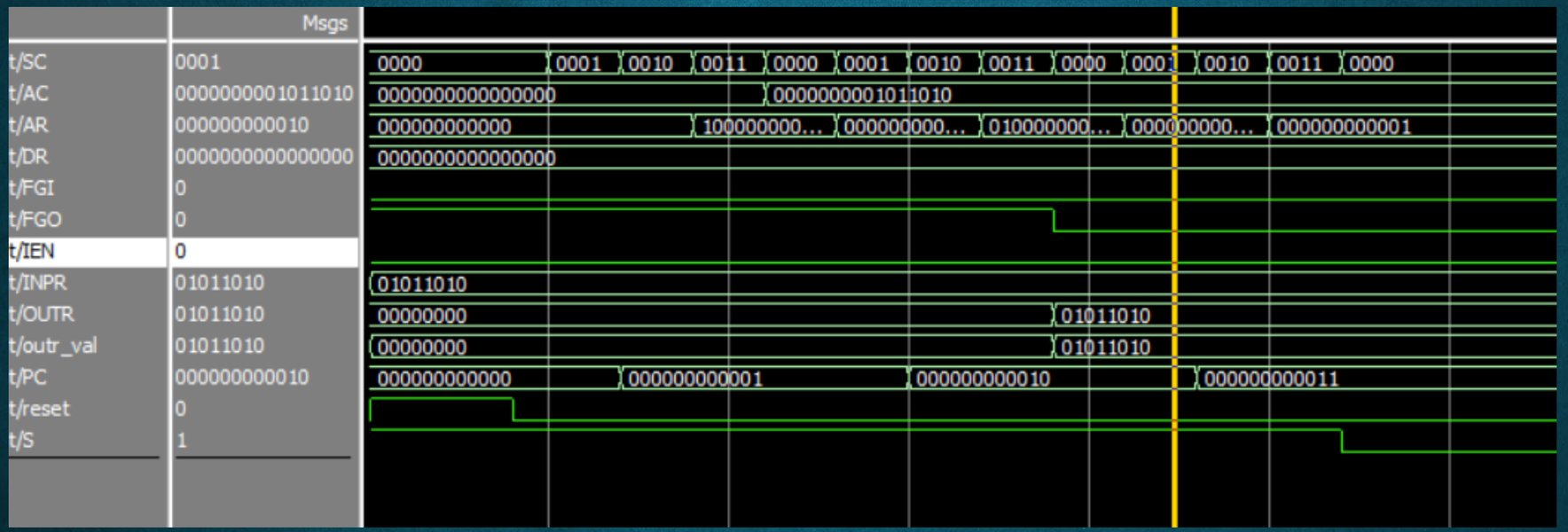

6)
I->OF

# INPUT OUTPUT INSTRUCTIONS

**IOF** $pB_6$: $IEN \leftarrow 0$

```
elsif IR(6) = '1' then
    IEN <= '0'; -- IOF
end if;
```

# INTERRUPT AND I/O FLAG HANDLING LOGIC



# INTERRUPT AND I/O FLAG HANDLING LOGIC

## Interrupt Check

```
if IR(7)   = '1' then IEN <= '1'; end if; -- ION
if IR(6)   = '1' then IEN <= '0'; end if; -- IOF
```


# THANK YOU

ESLAM AHMED

MOHAMED EMAD

BASIL MAGDY