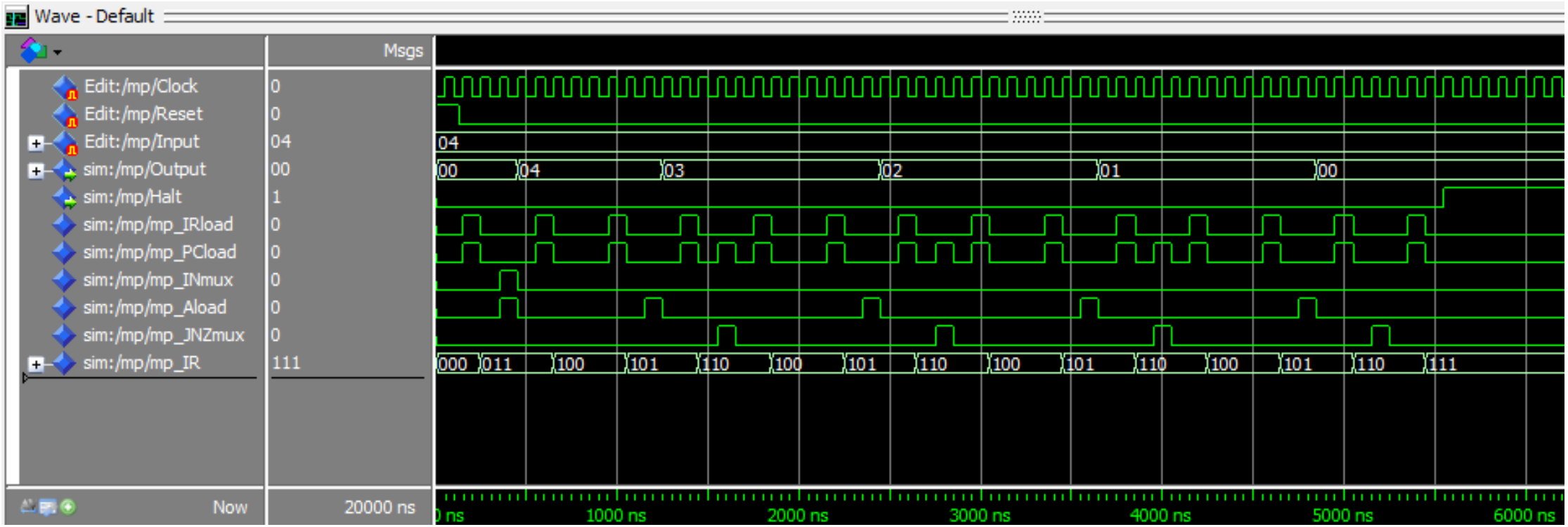
Lab 09. General CPU design 1

201810800 이혜인

- 주어진 VHDL code들을 사용해서 EC-1 microprocessor를 구현하고 “RTL viewer”와 “RTL simulation” 기능을 이용해서 구현된 결과를 분석하라.
  - You have to only perform “RTL simulation” since it is much easier for verification. (SDO 파일 없이, “gate_work” 대신 “work” library에서 entity load)
  - Use the Countdown program in “program.mif” & Verify with simulation the following algorithms.

- Result
  - RTL view capture & explanation
  - Simulation capture & detailed explanation. Does it behave as the program was intended?
  - Etc.
- Discussion
  - The problems met during simulation & verification
  - How they have been solved
  - The problems remained unsolved

Example simulation capture



- Input, output, 내부 signal 모두 관찰 필요

Lab 09. General CPU design 1

Result

1. VHDL Code – mp

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3
4  ENTITY mp IS PORT (
5     Clock, Reset: IN STD_LOGIC;
6     Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
8     Halt: OUT STD_LOGIC);
9  END mp;
10
11 ARCHITECTURE Structural OF mp IS
12    COMPONENT cu PORT (
13       clock, reset : IN STD_LOGIC;
14       IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
15       IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
16       Aneq0: IN STD_LOGIC;
17       halt: OUT STD_LOGIC);
18    END COMPONENT;
19
20    COMPONENT dp PORT (
21       Clock, Clear: IN STD_LOGIC;
22       Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23       IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
24       IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
25       Xneq0: OUT STD_LOGIC;
26       Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
27    END COMPONENT;
28
29    SIGNAL mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux: STD_LOGIC;
30    SIGNAL mp_IR: STD_LOGIC_VECTOR(7 DOWNTO 5);
31    SIGNAL mp_Aneq0: STD_LOGIC;
32 BEGIN
33     -- doing structural modeling for the microprocessor here
34    U0: cu PORT MAP(Clock, Reset, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Halt);
35    U1: dp PORT MAP(Clock, Reset, Input, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Output);
36 END Structural;
```

→ 다음은 주어진 md(Microprocessor Design) Code이다.

→ 외부 Input으로는 Clock과 Reset, 그리고 Input을 가지고 있다.

이때 Input은 연산을 할 값을 입력 받는다.

→ System Output으로는 Output과 Halt를 가지고 있다. 이때

Output은 연산을 한 결과이고, Halt는 State가 111이 되면 1이

되면서 프로그램의 종료를 알리는 역할을 한다.

1. VHDL Code – mp

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3
4  ENTITY mp IS PORT (
5     Clock, Reset: IN STD_LOGIC;
6     Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
8     Halt: OUT STD_LOGIC);
9  END mp;
10
11 ARCHITECTURE Structural OF mp IS
12    COMPONENT cu PORT (
13       clock, reset : IN STD_LOGIC;
14       IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
15       IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
16       Aneq0: IN STD_LOGIC;
17       halt: OUT STD_LOGIC);
18    END COMPONENT;
19
20    COMPONENT dp PORT (
21       Clock, Clear: IN STD_LOGIC;
22       Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23       IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
24       IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
25       Xneq0: OUT STD_LOGIC;
26       Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
27    END COMPONENT;
28
29    SIGNAL mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux: STD_LOGIC;
30    SIGNAL mp_IR: STD_LOGIC_VECTOR(7 DOWNTO 5);
31    SIGNAL mp_Aneq0: STD_LOGIC;
32 BEGIN
33     -- doing structural modeling for the microprocessor here
34    U0: cu PORT MAP(Clock, Reset, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Halt);
35    U1: dp PORT MAP(Clock, Reset, Input, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Output);
36 END Structural;
```

→ 다음 Code에서는 CU(Control Unit)와 DP(Datapath)를 각각 Component하였다.

→ 그 후 내부 Signal들을 이용하여 CU, DP와 MD를 연결하였다.

1. VHDL Code – mp

```
1  LIBRARY IEEE;
2  USE IEEE.std_logic_1164.all;
3
4  ENTITY mp IS PORT (
5     Clock, Reset: IN STD_LOGIC;
6     Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
7     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
8     Halt: OUT STD_LOGIC);
9  END mp;
10
11 ARCHITECTURE Structural OF mp IS
12    COMPONENT cu PORT (
13       clock, reset : IN STD_LOGIC;
14       IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
15       IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
16       Aneq0: IN STD_LOGIC;
17       halt: OUT STD_LOGIC);
18    END COMPONENT;
19
20    COMPONENT dp PORT (
21       Clock, Clear: IN STD_LOGIC;
22       Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
23       IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
24       IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
25       Xneq0: OUT STD_LOGIC;
26       Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
27    END COMPONENT;
28
29    SIGNAL mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux: STD_LOGIC;
30    SIGNAL mp_IR: STD_LOGIC_VECTOR(7 DOWNTO 5);
31    SIGNAL mp_Aneq0: STD_LOGIC;
32 BEGIN
33     -- doing structural modeling for the microprocessor here
34    U0: cu PORT MAP(Clock, Reset, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Halt);
35    U1: dp PORT MAP(Clock, Reset, Input, mp_IRload, mp_PCload, mp_INmux, mp_Aload, mp_JNZmux, mp_IR, mp_Aneq0, Output);
36 END Structural;
```

→ mp_IRload와 mp_PCload는 각각의 load값이 각각의 단계에서 0과 1 중 어떤 값인지 나타내는 Signal이고, mp_INmux는 외부 Input을 받을지 Decrement값을 받을지 보여주는 Signal이다.

→ 또한, mp_Aload는 A에 값이 저장되는지 보여주는 Signal이고, mp_JNZmux는 A에 있는 값이 0인지 아닌지 판단해서 해당 address로 이동하라고 명령해주는 Signal이다.

1. VHDL Code – dp(Datapath)

```vhdl
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

LIBRARY lpm;
USE lpm.lpm_components.ALL;

ENTITY dp IS PORT (
    Clock, Clear: IN STD_LOGIC;
    Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
    IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
    IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
    Xneq0: OUT STD_LOGIC;
    Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
END dp;

ARCHITECTURE Structural OF dp IS
    COMPONENT reg
    GENERIC (size: INTEGER := 4); -- the actual size is defined in the instantiation GENERIC MAP
    PORT (
       Clock, Clear, Load: IN STD_LOGIC;
       D: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
       Q: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
    END COMPONENT;

    COMPONENT increment
    GENERIC (size: INTEGER := 8); -- default number of bits
    PORT (
       A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
       F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
    END COMPONENT;

    COMPONENT decrement
    GENERIC (size: INTEGER := 8); -- default number of bits
    PORT (
       A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
       F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
    END COMPONENT;
```

→ 다음은 Datapath에 대한 Code이다.

```vhdl
    COMPONENT mux2
    GENERIC (size: INTEGER := 8);              -- default size
    PORT (
       S: IN STD_LOGIC;                       -- select line
       D1, D0: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);   -- data bus input
        Y: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));     -- data bus output
    END COMPONENT;

    SIGNAL dp_IR, dp_ROMQ: STD_LOGIC_VECTOR(7 DOWNTO 0);
    SIGNAL dp_JNZmux, dp_PC, dp_increment: STD_LOGIC_VECTOR(3 DOWNTO 0);
    SIGNAL dp_INmux, dp_decrement, dp_A: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
    -- doing structural modeling for the datapath here
    U0_IR: reg GENERIC MAP (8) PORT MAP(Clock, Clear, IRLoad, dp_ROMQ, dp_IR);
    U1_JNZmux: mux2 GENERIC MAP (4) PORT MAP(JNZmux,dp_IR(3 DOWNTO 0),dp_increment,dp_JNZmux);
    U2_PC: reg GENERIC MAP (4) PORT MAP(Clock, Clear, PCLoad, dp_JNZmux, dp_PC);
    U3_inc: increment GENERIC MAP (4) PORT MAP(dp_PC,dp_increment);

    U4_ROM: lpm_rom
       GENERIC MAP (
          lpm_widthad => 4,
          lpm_outdata => "UNREGISTERED",
          lpm_file => "program.mif", -- fill rom with content of file program.mif
          lpm_width   => 8)
       PORT MAP (address => dp_PC, inclock => Clock, q => dp_ROMQ);

    U5_INmux: mux2 GENERIC MAP (8) PORT MAP(INmux,Input,dp_decrement,dp_INmux);
    U6_A: reg GENERIC MAP (8) PORT MAP(Clock, Clear, ALoad, dp_INmux, dp_A);
    U7_dec: decrement PORT MAP(dp_A,dp_decrement);

    Xneq0 <= '1' WHEN dp_A /= "00000000" ELSE '0';
    IR <= dp_IR(7 DOWNTO 5);
    Output <= dp_A;
END Structural;
```

1. VHDL Code – dp(Datapath)

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3
4   LIBRARY lpm;
5   USE lpm.lpm_components.ALL;
6
7   ENTITY dp IS PORT (
8      Clock, Clear: IN STD_LOGIC;
9      Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
10     IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
11     IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
12     Xneq0: OUT STD_LOGIC;
13     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
14  END dp;
15
16  ARCHITECTURE Structural OF dp IS
17     COMPONENT reg
18     GENERIC (size: INTEGER := 4); -- the actual size is defined in the instantiation GENERIC MAP
19     PORT (
20        Clock, Clear, Load: IN STD_LOGIC;
21        D: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
22        Q: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
23     END COMPONENT;
24
25     COMPONENT increment
26     GENERIC (size: INTEGER := 8); -- default number of bits
27     PORT (
28        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
29        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
30     END COMPONENT;
31
32     COMPONENT decrement
33     GENERIC (size: INTEGER := 8); -- default number of bits
34     PORT (
35        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
36        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
37     END COMPONENT;
```

→ Input으로는 Clock과 Clear, Input, IRload, PCload, INmux, Aload, JNZmux를 가지고 있다.

→ IRload와 PCload는 각각의 load값이 각각의 단계에서 0과 1 중 값을 가져 load의 여부를 결정해주는 Input이다. IRload와 PCload는 fetch단계에서만 1값을 가진다. (다만, JNZ에서 PCload가 1을 갖기도 한다.)

1. VHDL Code – dp(Datapath)

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3
4   LIBRARY lpm;
5   USE lpm.lpm_components.ALL;
6
7   ENTITY dp IS PORT (
8      Clock, Clear: IN STD_LOGIC;
9      Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
10     IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
11     IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
12     Xneq0: OUT STD_LOGIC;
13     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
14  END dp;
15
16  ARCHITECTURE Structural OF dp IS
17     COMPONENT reg
18     GENERIC (size: INTEGER := 4); -- the actual size is defined in the instantiation GENERIC MAP
19     PORT (
20        Clock, Clear, Load: IN STD_LOGIC;
21        D: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
22        Q: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
23     END COMPONENT;
24
25     COMPONENT increment
26     GENERIC (size: INTEGER := 8); -- default number of bits
27     PORT (
28        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
29        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
30     END COMPONENT;
31
32     COMPONENT decrement
33     GENERIC (size: INTEGER := 8); -- default number of bits
34     PORT (
35        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
36        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
37     END COMPONENT;
```

→ INmux는 외부 Input을 받을지 Decrement값을 받을지 결정하는 Input이고, Aload는 A에 값이 저장되는 여부를 결정하는 Input이고, JNZmux는 A에 있는 값이 0인지 아닌지 판단해서 해당 address로 이동하라고 해주는 Input이다.

1. VHDL Code – dp(Datapath)

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3
4   LIBRARY lpm;
5   USE lpm.lpm_components.ALL;
6
7   ENTITY dp IS PORT (
8      Clock, Clear: IN STD_LOGIC;
9      Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
10     IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
11     IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
12     Xneq0: OUT STD_LOGIC;
13     Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
14  END dp;
15
16  ARCHITECTURE Structural OF dp IS
17     COMPONENT reg
18     GENERIC (size: INTEGER := 4); -- the actual size is defined in the instantiation GENERIC MAP
19     PORT (
20        Clock, Clear, Load: IN STD_LOGIC;
21        D: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
22        Q: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
23     END COMPONENT;
24
25     COMPONENT increment
26     GENERIC (size: INTEGER := 8); -- default number of bits
27     PORT (
28        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
29        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
30     END COMPONENT;
31
32     COMPONENT decrement
33     GENERIC (size: INTEGER := 8); -- default number of bits
34     PORT (
35        A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
36        F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
37     END COMPONENT;
```

→ Output으로는 IR과 Xneq0, Output을 가지고 있다.

→ IR은 Instruction을 나타내는 총 8bit 중에서 MSB 3bit를 Status Signal 형태로 Control Unit에 제공해주는 Output이고(MSB 3bit는 Instruction을 가리킨다.), Xneq0는 A가 0과 같은 지 아닌지를 판단해 출력해주는 것이고, Output은 외부로 A의 값을 출력해주는 것이다.

1. VHDL Code – dp(Datapath)

```
1   LIBRARY IEEE;
2   USE IEEE.std_logic_1164.ALL;
3
4   LIBRARY lpm;
5   USE lpm.lpm_components.ALL;
6
7   ENTITY dp IS PORT (
8       Clock, Clear: IN STD_LOGIC;
9       Input: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
10      IRload, PCload, INmux, Aload, JNZmux: IN STD_LOGIC;
11      IR: OUT STD_LOGIC_VECTOR(7 DOWNTO 5);
12      Xneq0: OUT STD_LOGIC;
13      Output: OUT STD_LOGIC_VECTOR(7 DOWNTO 0));
14  END dp;
15
16  ARCHITECTURE Structural OF dp IS
17      COMPONENT reg
18      GENERIC (size: INTEGER := 4); -- the actual size is defined in the instantiation GENERIC MAP
19      PORT (
20          Clock, Clear, Load: IN STD_LOGIC;
21          D: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
22          Q: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
23      END COMPONENT;
24
25      COMPONENT increment
26      GENERIC (size: INTEGER := 8); -- default number of bits
27      PORT (
28          A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
29          F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
30      END COMPONENT;
31
32      COMPONENT decrement
33      GENERIC (size: INTEGER := 8); -- default number of bits
34      PORT (
35          A: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);
36          F: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));
37      END COMPONENT;
```

→ Generic를 이용하여 size가 4인 ROM을 생성하였다. 해당 Rom은 16개 word를 갖는다.

→ 다음 Code에서는 reg(Register), increment, decrement를 각각 Component하였다.

1. VHDL Code – dp(Datapath)

```
39    COMPONENT mux2
40    GENERIC (size: INTEGER := 8);             -- default size
41    PORT (
42       S: IN STD_LOGIC;                     -- select line
43       D1, D0: IN STD_LOGIC_VECTOR(size-1 DOWNTO 0);   -- data bus input
44        Y: OUT STD_LOGIC_VECTOR(size-1 DOWNTO 0));     -- data bus output
45    END COMPONENT;
46
47    SIGNAL dp_IR, dp_ROMQ: STD_LOGIC_VECTOR(7 DOWNTO 0);
48    SIGNAL dp_JNZmux, dp_PC, dp_increment: STD_LOGIC_VECTOR(3 DOWNTO 0);
49    SIGNAL dp_INmux, dp_decrement, dp_A: STD_LOGIC_VECTOR(7 DOWNTO 0);
50  BEGIN
51     -- doing structural modeling for the datapath here
52    U0_IR: reg GENERIC MAP (8) PORT MAP(Clock, Clear, IRLoad, dp_ROMQ, dp_IR);
53    U1_JNZmux: mux2 GENERIC MAP (4) PORT MAP(JNZmux,dp_IR(3 DOWNTO 0),dp_increment,dp_JNZmux);
54    U2_PC: reg GENERIC MAP (4) PORT MAP(Clock, Clear, PCLoad, dp_JNZmux, dp_PC);
55    U3_inc: increment GENERIC MAP (4) PORT MAP(dp_PC,dp_increment);
56
57    U4_ROM: lpm_rom
58       GENERIC MAP (
59          lpm_widthad => 4,
60          lpm_outdata => "UNREGISTERED",
61          lpm_file => "program.mif", -- fill rom with content of file program.mif
62          lpm_width   => 8)
63       PORT MAP (address => dp_PC, inclock => Clock, q => dp_ROMQ);
64
65    U5_INmux: mux2 GENERIC MAP (8) PORT MAP(INmux,Input,dp_decrement,dp_INmux);
66    U6_A: reg GENERIC MAP (8) PORT MAP(Clock, Clear, ALoad, dp_INmux, dp_A);
67    U7_dec: decrement PORT MAP(dp_A,dp_decrement);
68
69    Xneq0 <= '1' WHEN dp_A /= "00000000" ELSE '0';
70    IR <= dp_IR(7 DOWNTO 5);
71    Output <= dp_A;
72  END Structural;
```

→ 다음 Code에서는 mux2를 Component하였다.

→ 그 후 내부 Signal들을 이용하여 Component한 값들을 DP와 연결하였다.

→ 그리고 Altera에서 제공된 ROM을 이용하여 ROM을 만들었다.

→ Xneq0을 통해 A가 0이 아니면 1, 0이면 0을 내보냈고, IR은 dp_IR값 중에서 MSB 3bit만을 출력하였다. 그리고 dp_A에 있는 값을 Output을 통해 출력하였다.

1. VHDL Code – CU(Control Unit)

```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY cu IS PORT (
    clock, reset : IN STD_LOGIC;
    -- control signals
    IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
    -- status signals
    IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
    Aneq0: IN STD_LOGIC;
    -- control outputs
    halt: OUT STD_LOGIC);
END cu;

ARCHITECTURE FSM OF cu IS
    SIGNAL state: STD_LOGIC_VECTOR(2 DOWNTO 0);
BEGIN
    next_state_logic: PROCESS(reset, clock)
    BEGIN
        IF(reset = '1') THEN
            state <= "000";
        ELSIF(clock'EVENT AND clock = '1') THEN
            CASE state IS
            WHEN "000" => -- reset, start
                state <= "001";
            WHEN "001" => -- fetch
                state <= "010";
            WHEN "010" => -- s_decode
                CASE IR IS
                    WHEN "011" => state <= "011"; --s_input;
                    WHEN "100" => state <= "100"; --s_output;
                    WHEN "101" => state <= "101"; --s_dec;
                    WHEN "110" => state <= "110"; --s_jnz;
                    WHEN "111" => state <= "111"; --s_halt;
                    WHEN OTHERS => state <= "000"; --s_start;
                END CASE;
            WHEN "011" => -- s_input
                state <= "000";
            WHEN "100" => -- s_output =>
                state <= "000";
            WHEN "101" => -- s_dec =>
                state <= "000";
            WHEN "110" => -- s_jnz =>
                state <= "000";
            WHEN "111" => -- s_halt =>
                state <= "111";
            WHEN OTHERS =>
                state <= "000";
            END CASE;
        END IF;
    END PROCESS;
    output_logic: PROCESS(state)
    BEGIN
        CASE state IS
        WHEN "001" => --s_fetch =>
            IRload <= '1';
            PCload <= '1';
            INmux <= '0';
            Aload <= '0';
            JNZmux <= '0';
            halt <= '0';
        WHEN "011" => --s_input =>
            IRload <= '0';
            PCload <= '0';
            INmux <= '1';
            Aload <= '1';
            JNZmux <= '0';
            halt <= '0';
        WHEN "101" => --s_dec =>
            IRload <= '0';
            PCload <= '0';
            INmux <= '0';
            Aload <= '1';
            JNZmux <= '0';
            halt <= '0';
        WHEN "110" => --s_jnz =>
            IRload <= '0';
            IF (Aneq0 = '1') THEN
                PCload <= '1';
            ELSE
                PCload <= '0';
            END IF;
            INmux <= '0';
            Aload <= '0';
            JNZmux <= '1';
            halt <= '0';
        WHEN "111" => --s_halt =>
            IRload <= '0';
            PCload <= '0';
            INmux <= '0';
            Aload <= '0';
            JNZmux <= '0';
            halt <= '1';
        WHEN OTHERS =>
            IRload <= '0';
            PCload <= '0';
            INmux <= '0';
            Aload <= '0';
            JNZmux <= '0';
            halt <= '0';
        END CASE;
    END PROCESS;
END FSM;
```

→ 다음은 Control Unit에 대한 Code이다.

1. VHDL Code – CU(Control Unit)

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY cu IS PORT (
5     clock, reset : IN STD_LOGIC;
6     -- control signals
7     IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
8     -- status signals
9     IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
10    Aneq0: IN STD_LOGIC;
11    -- control outputs
12    halt: OUT STD_LOGIC);
13 END cu;
14
15 ARCHITECTURE FSM OF cu IS
16    SIGNAL state: STD_LOGIC_VECTOR(2 DOWNTO 0);
17 BEGIN
18    next_state_logic: PROCESS(reset, clock)
19    BEGIN
20       IF(reset = '1') THEN
21          state <= "000";
22       ELSIF(clock'EVENT AND clock = '1') THEN
23          CASE state IS
24          WHEN "000" => -- reset, start
25             state <= "001";
26          WHEN "001" => -- fetch
27             state <= "010";
28          WHEN "010" => -- s_decode
29             CASE IR IS
30                WHEN "011" => state <= "011"; --s_input;
31                WHEN "100" => state <= "100"; --s_output;
32                WHEN "101" => state <= "101"; --s_dec;
33                WHEN "110" => state <= "110"; --s_jnz;
34                WHEN "111" => state <= "111"; --s_halt;
35                WHEN OTHERS => state <= "000"; --s_start;
36             END CASE;
```

→ Input으로는 Clock과 reset, Input, IRload, PCload, INmux, Aload, JNZmux, IR, Aneq0를 가지고 있다.

→ 대부분 Input을 Datapath와 동일하고, 다른 것은 IR, Aneq0이다. IR은 Datapath에서 내보낸 IR Output을 받는 Input이다. Aneq0는 Datapath에서 내보낸 Xneq0 Output을 받는 Input이다.

→ Output으로는 halt를 가지고 있다.

→ Halt는 해당 프로그램의 State가 111이면 프로그램이 종료되었음을 알리는 Output이다.

1. VHDL Code – CU(Control Unit)

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY cu IS PORT (
5     clock, reset : IN STD_LOGIC;
6     -- control signals
7     IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
8     -- status signals
9     IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
10    Aneq0: IN STD_LOGIC;
11    -- control outputs
12    halt: OUT STD_LOGIC);
13 END cu;
14
15 ARCHITECTURE FSM OF cu IS
16    SIGNAL state: STD_LOGIC_VECTOR(2 DOWNTO 0);
17 BEGIN
18    next_state_logic: PROCESS(reset, clock)
19    BEGIN
20       IF(reset = '1') THEN
21          state <= "000";
22       ELSIF(clock'EVENT AND clock = '1') THEN
23          CASE state IS
24          WHEN "000" => -- reset, start
25             state <= "001";
26          WHEN "001" => -- fetch
27             state <= "010";
28          WHEN "010" => -- s_decode
29             CASE IR IS
30                WHEN "011" => state <= "011"; --s_input;
31                WHEN "100" => state <= "100"; --s_output;
32                WHEN "101" => state <= "101"; --s_dec;
33                WHEN "110" => state <= "110"; --s_jnz;
34                WHEN "111" => state <= "111"; --s_halt;
35                WHEN OTHERS => state <= "000"; --s_start;
36          END CASE;
```

→ State는 총 7개로 3bit로 구성되어 있다.

→ Reset이 1이 되면 state는 초기 state인 000으로 Initialized한다.

→ State가 000이면, State 001로 가고, State가 001(fetch : MSB 3 bit를 받는다.)이면 State는 010으로 간다.

1. VHDL Code – CU(Control Unit)

```
1  LIBRARY IEEE;
2  USE IEEE.STD_LOGIC_1164.ALL;
3
4  ENTITY cu IS PORT (
5     clock, reset : IN STD_LOGIC;
6     -- control signals
7     IRload, PCload, INmux, Aload, JNZmux: OUT STD_LOGIC;
8     -- status signals
9     IR: IN STD_LOGIC_VECTOR(7 DOWNTO 5);
10    Aneq0: IN STD_LOGIC;
11    -- control outputs
12    halt: OUT STD_LOGIC);
13 END cu;
14
15 ARCHITECTURE FSM OF cu IS
16    SIGNAL state: STD_LOGIC_VECTOR(2 DOWNTO 0);
17 BEGIN
18    next_state_logic: PROCESS(reset, clock)
19    BEGIN
20       IF(reset = '1') THEN
21          state <= "000";
22       ELSIF(clock'EVENT AND clock = '1') THEN
23          CASE state IS
24          WHEN "000" => -- reset, start
25             state <= "001";
26          WHEN "001" => -- fetch
27             state <= "010";
28          WHEN "010" => -- s_decode
29             CASE IR IS
30                WHEN "011" => state <= "011"; --s_input;
31                WHEN "100" => state <= "100"; --s_output;
32                WHEN "101" => state <= "101"; --s_dec;
33                WHEN "110" => state <= "110"; --s_jnz;
34                WHEN "111" => state <= "111"; --s_halt;
35                WHEN OTHERS => state <= "000"; --s_start;
36             END CASE;
```

→ State가 010이면, State는 경우에 따라 해당 State로 이동한다. Input을 받는 경우는 State 011로, Output을 내보내는 경우는 State 100으로, decrement를 하는 경우는 State 101로, JNZ를 하는 경우는 State 110으로, 그리고 프로그램이 종료되어 halt인 경우는 State 111로 이동한다.

→ 다른 경우는 모두 000으로 이동한다.

1. VHDL Code – CU(Control Unit)

```
37          WHEN "011" => -- s_input
38             state <= "000";
39          WHEN "100" => -- s_output =>
40             state <= "000";
41          WHEN "101" => -- s_dec =>
42             state <= "000";
43          WHEN "110" => -- s_jnz =>
44             state <= "000";
45          WHEN "111" => -- s_halt =>
46             state <= "111";
47          WHEN OTHERS =>
48             state <= "000";
49          END CASE;
50       END IF;
51    END PROCESS;
53    output_logic: PROCESS(state)
54    BEGIN
55       CASE state IS
56       WHEN "001" => --s_fetch =>
57          IRload <= '1';
58          PCload <= '1';
59          INmux <= '0';
60          Aload <= '0';
61          JNZmux <= '0';
62          halt <= '0';
63       WHEN "011" => --s_input =>
64          IRload <= '0';
65          PCload <= '0';
66          INmux <= '1';
67          Aload <= '1';
68          JNZmux <= '0';
69          halt <= '0';
```

→ State가 011, 100, 101, 110인 경우에는 다시 초기 State인 000으로 가고, 111인 경우에는 프로그램이 종료된 상태이므로 계속해서 State 111에 머무른다.

→ 다른 경우는 모두 000으로 이동한다.

1. VHDL Code – CU(Control Unit)

```
37          WHEN "011" => -- s_input
38             state <= "000";
39          WHEN "100" => -- s_output =>
40             state <= "000";
41          WHEN "101" => -- s_dec =>
42             state <= "000";
43          WHEN "110" => -- s_jnz =>
44             state <= "000";
45          WHEN "111" => -- s_halt =>
46             state <= "111";
47          WHEN OTHERS =>
48             state <= "000";
49          END CASE;
50       END IF;
51    END PROCESS;
53    output_logic: PROCESS(state)
54    BEGIN
55       CASE state IS
56       WHEN "001" => --s_fetch =>
57          IRload <= '1';
58          PCload <= '1';
59          INmux <= '0';
60          Aload <= '0';
61          JNZmux <= '0';
62          halt <= '0';
63       WHEN "011" => --s_input =>
64          IRload <= '0';
65          PCload <= '0';
66          INmux <= '1';
67          Aload <= '1';
68          JNZmux <= '0';
69          halt <= '0';
```

→ 다음은 각각 State 별로 Input, Output값을 나타낸 것이다.

→ 이는 해당 표를 참고하면 동일하다는 것을 알 수 있다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

1. VHDL Code – CU(Control Unit)

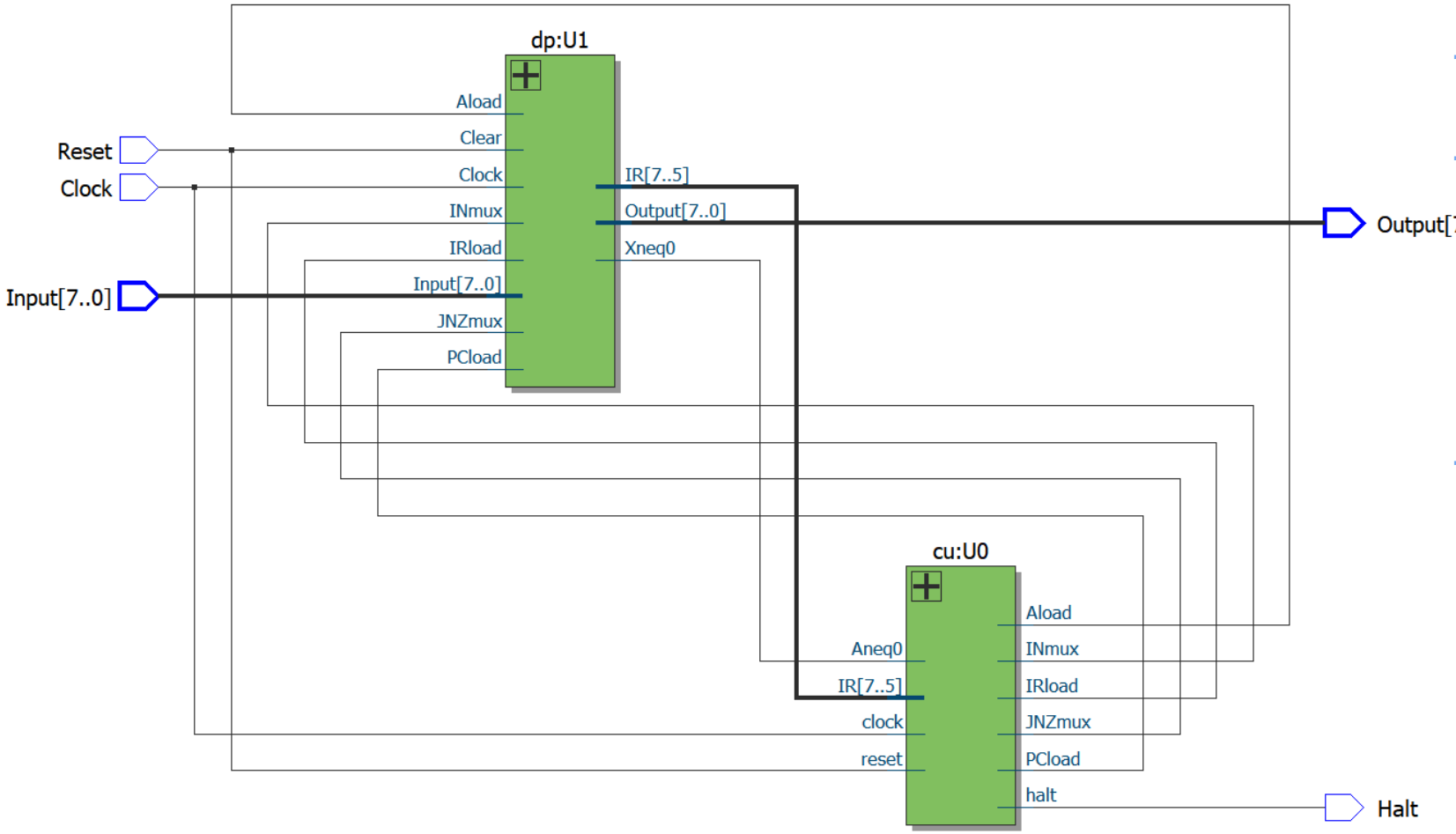
```
70          WHEN "101" => --s_dec =>
71             IRload <= '0';
72             PCload <= '0';
73             INmux <= '0';
74             Aload <= '1';
75             JNZmux <= '0';
76             halt <= '0';
77          WHEN "110" => --s_jnz =>
78             IRload <= '0';
79             IF (Aneq0 = '1') THEN
80                PCload <= '1';
81             ELSE
82                PCload <= '0';
83             END IF;
84             INmux <= '0';
85             Aload <= '0';
86             JNZmux <= '1';
87             halt <= '0';
88          WHEN "111" => --s_halt =>
89             IRload <= '0';
90             PCload <= '0';
91             INmux <= '0';
92             Aload <= '0';
93             JNZmux <= '0';
94             halt <= '1';
95          WHEN OTHERS =>
96             IRload <= '0';
97             PCload <= '0';
98             INmux <= '0';
99             Aload <= '0';
100            JNZmux <= '0';
101            halt <= '0';
102         END CASE;
103      END PROCESS;
104   END FSM;
```

→ 각각 State 별로 Input, Output값을 나타낸 것이다.

→ 이는 해당 표를 참고하면 동일하다는 것을 알 수 있다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

2. RTL Viewer

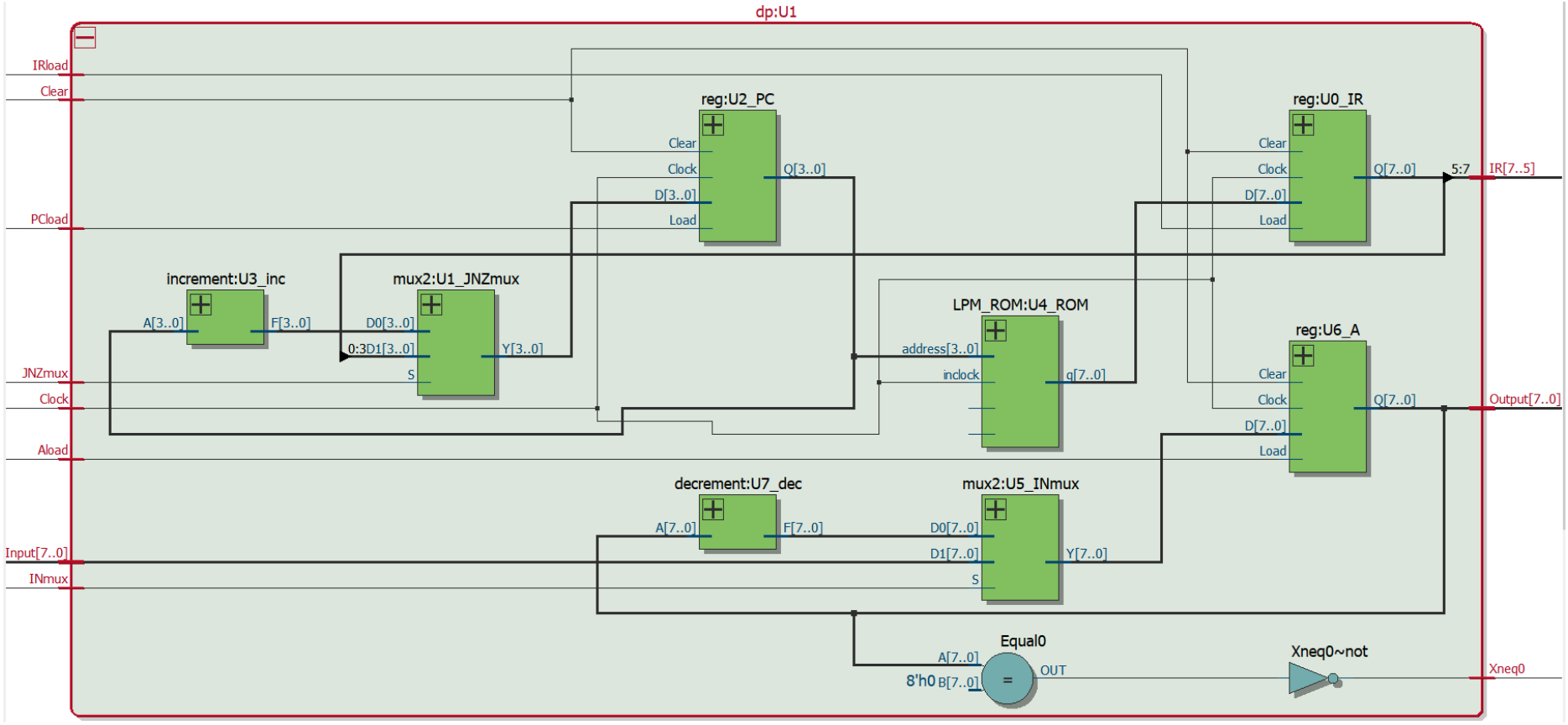


→ 다음은 해당 Code의 RTL Viewer이다.

→ 해당 Code는 DP(Datapath)와 CU(Control Unit)으로 이루어져 있는 것을 확인할 수 있다.

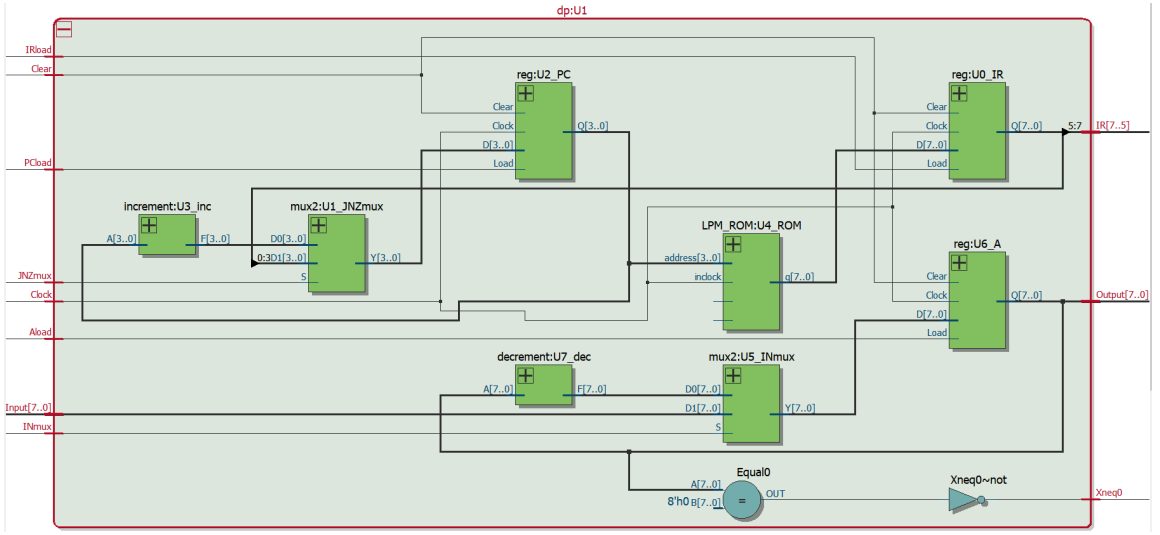
→ Input은 Reset, Clock, Input(8bit) 3가지이고, System Output은 Output(8bit)와 Halt 2가지이다.

2. RTL Viewer



→ 다음은 dp(Datapath)의 RTL Viewer이다.

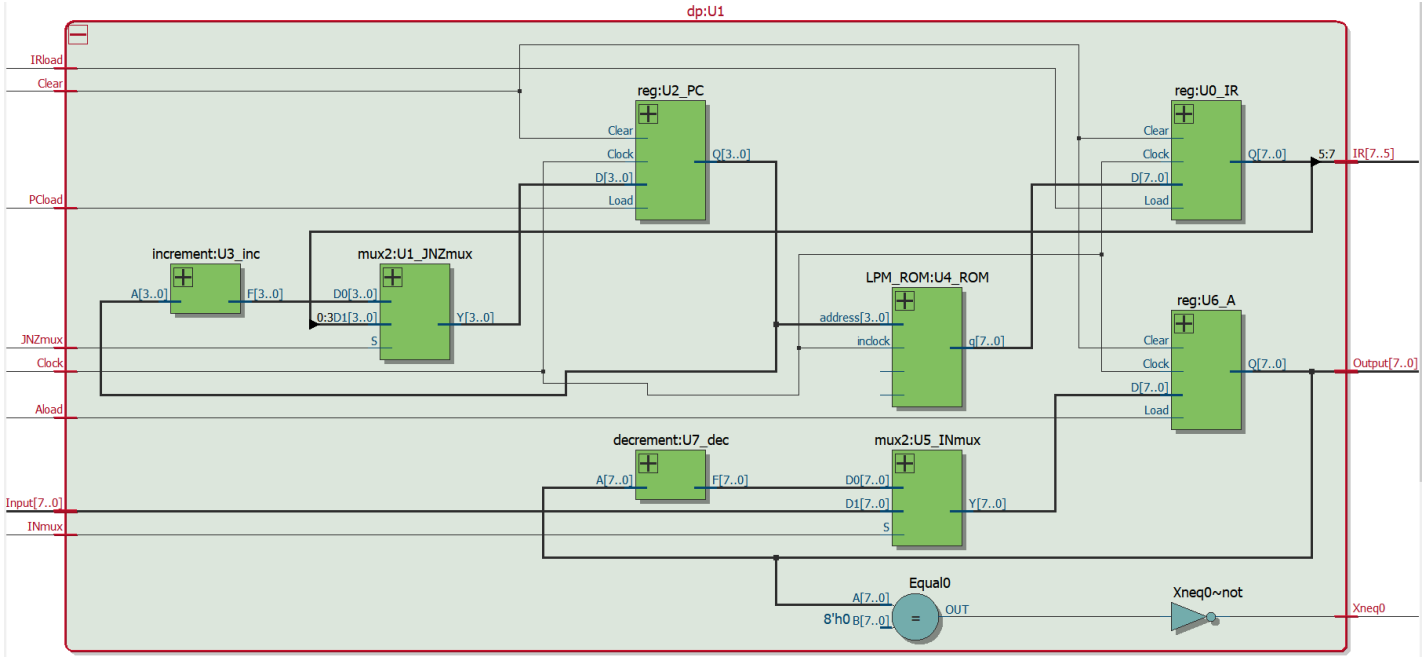
2. RTL Viewer



→ Decrement를 먼저 보면, 이는 일종의 ALU로 볼 수 있다. 그리고 Decrement를 한 다음 해당 값과 외부 Input 중 하나를 선택하여 A register에 저장하고, 해당 결과가 다시 feedback된다. 그 과정에서 A가 0과 동일한지 여부를 판단하여 Xneq0를 통해 출력된다.

→ 다음 Increment를 보면 이 값은 Pc의 output이 들어와서 1을 증가시키고, MUX를 통해서 해당 값과 IR의 Output 중 선택하여 PC에 값을 보낸다. PC에서는 1이 증가한 값이 나오면서 Increment로 다시 feedback이 되고, 해당 값이 ROM으로 들어가면서 ROM에서 address가 명시된 곳의 Instruction이 나와서 IR에 반영한다.

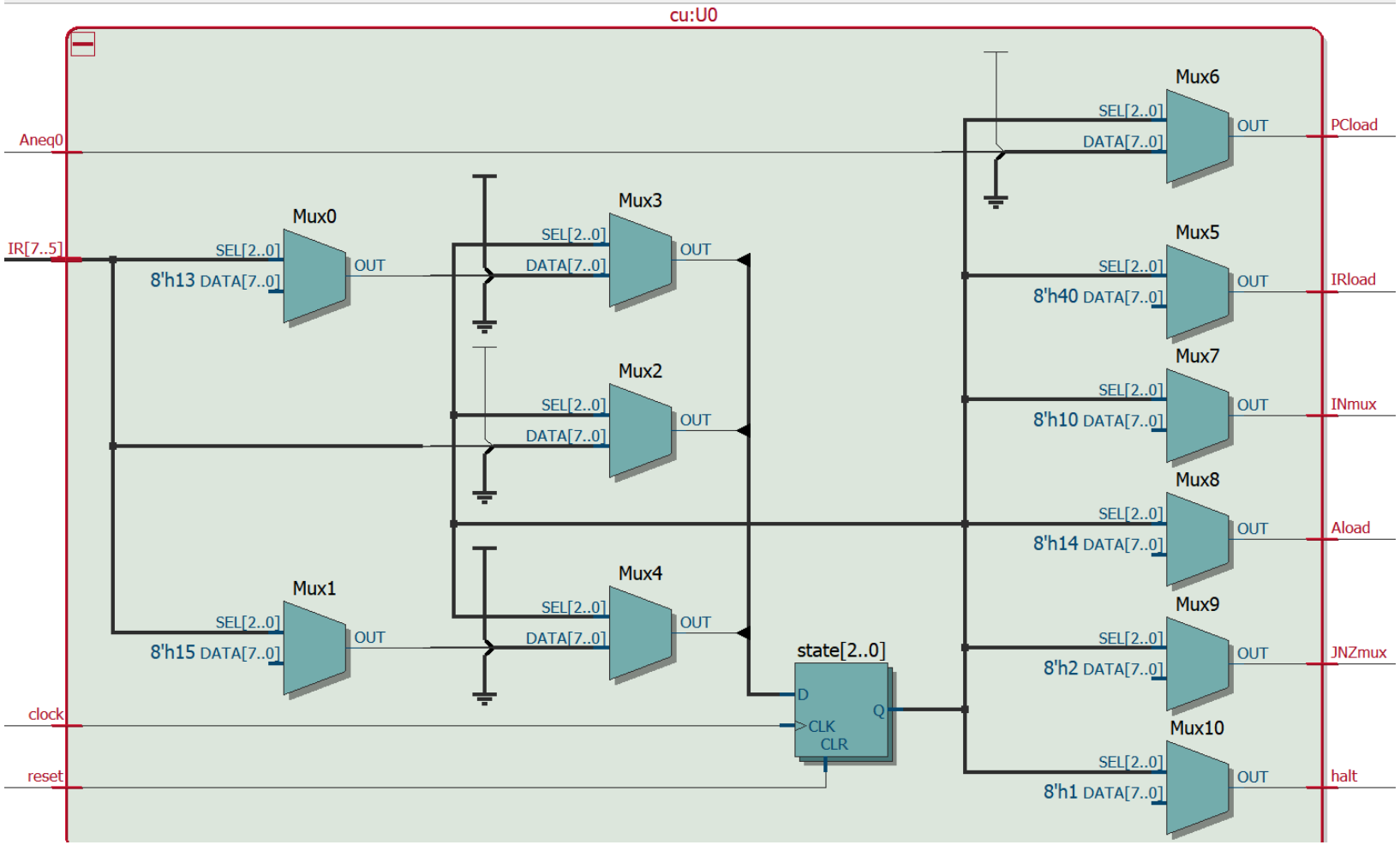
2. RTL Viewer



→ IR과 PC register는 각각 8bit와 4bit를 Input과 Output으로 가지는데, ROM의 address가 4bit여야 하므로 PC는 4bit로 구성된 것이고, ROM에서 나오는 Instruction이 8bit이므로 IR이 8bit로 구성된 것이다.

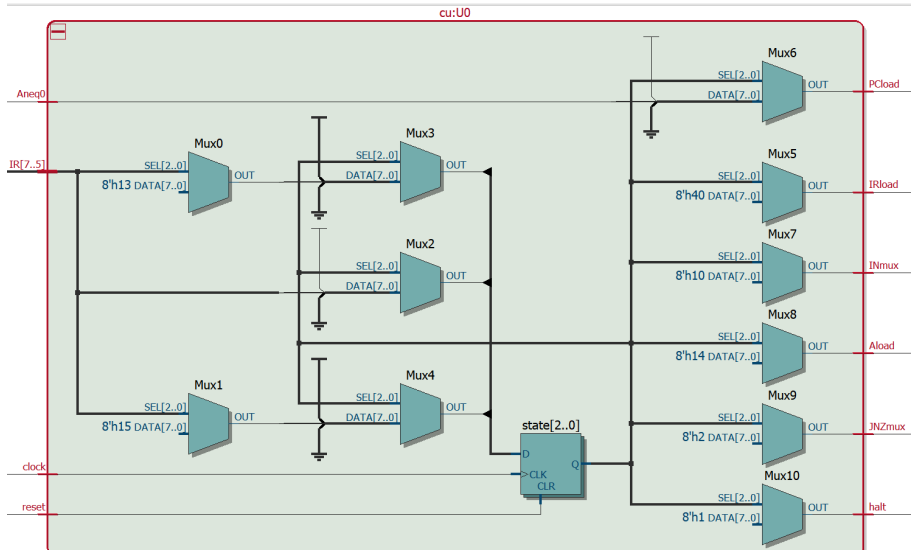
→ 그리고 IR에서 나온 값 중 MSB 3bit를 IR System Output을 통해 나가면서 CU의 IR Input으로 가고, Output은 전체 System의 Output이 된다. Xneq0는 CU의 Aneq0로 가게 된다.

2. RTL Viewer



→ 다음은 cu(Control Unit)의 RTL Viewer이다.

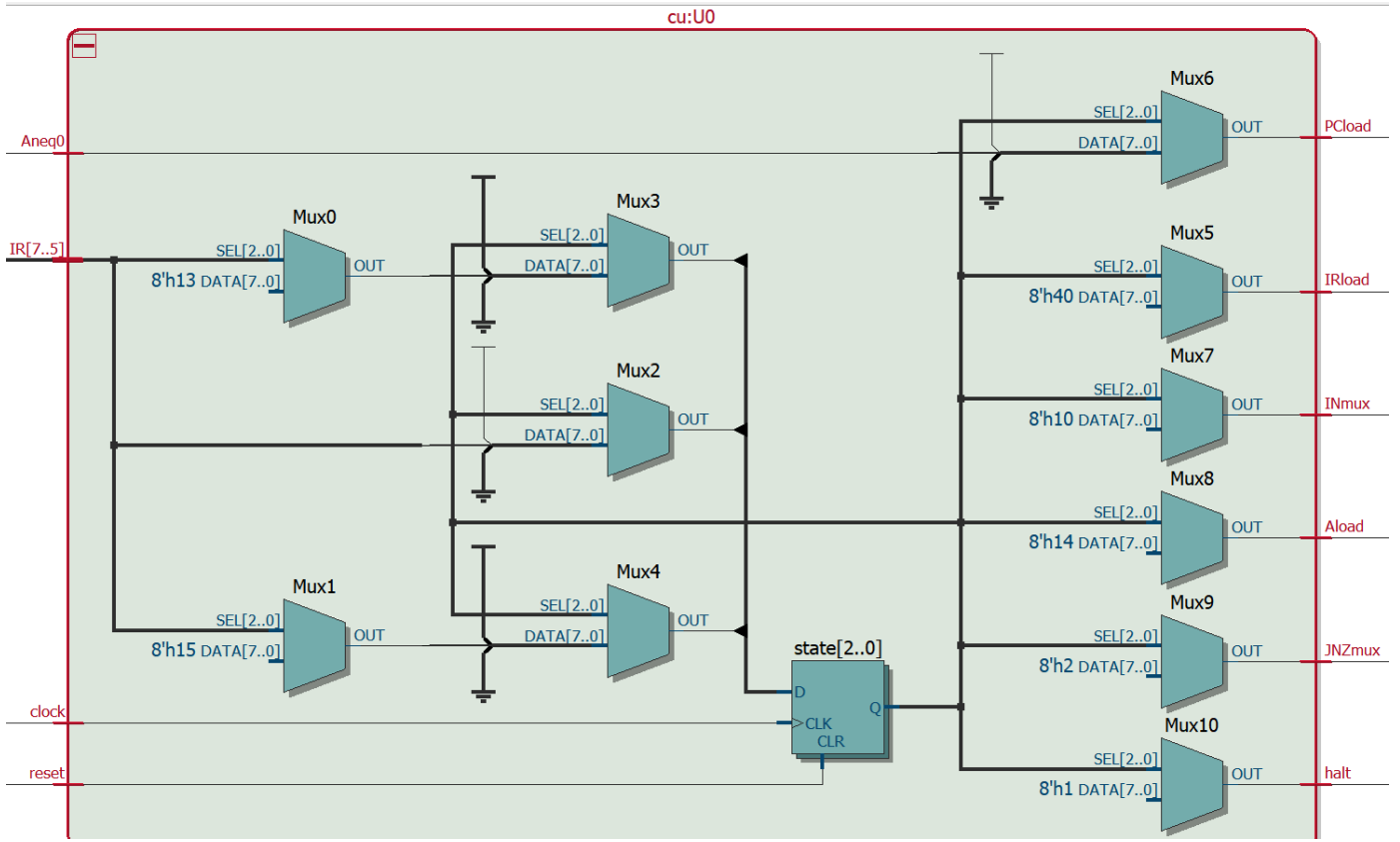
2. RTL Viewer



→ CU는 State Machine이다. 해당 State는 총 7개여서 3개의 flipflop으로 구성되었고, 여러 MUX를 통해 Combinational Logic이 구현되었다.

→ IR에서 MSB 3 bit를 받아온 값(Instruction)은 MUX를 통해 State로 가서 해당 Instruction(State)에 맞는 결과를 실행시킨다. State 011일 때는 Input을 A에 저장하고, State 100일 때는 Output을 출력해준다. State 101일 때는 decrement를 하고, State 110일 때는 A가 0이 아닌 상태이므로 JNZ를 한다. 마지막으로 State가 111이면 프로그램이 종료됨을 알리고 계속해서 State 111에 머무르게 된다.

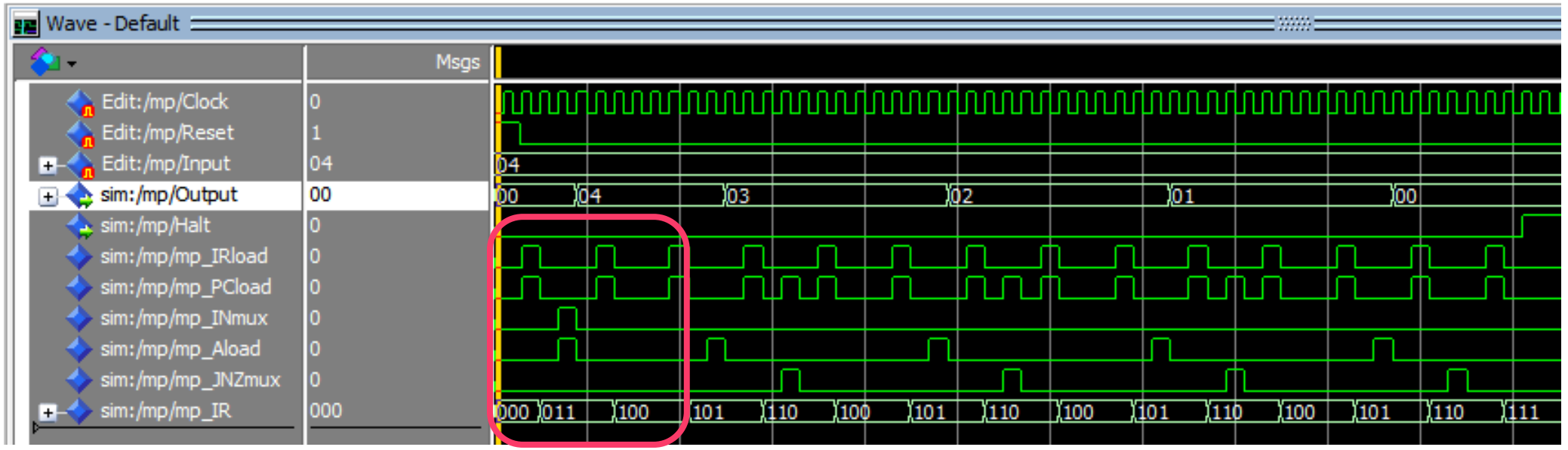
2. RTL Viewer



→ 각각의 결과를 실행시킨 다음, 해당 결과물들을 내부 Signal들을 통해 각각 연결해준다. Aload, INmux, Irload, JNZmux, PCload는 각각에 대응되는 dp의 Input으로 연결해주고, halt는 System Output을 통해 출력된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

3. Simulation



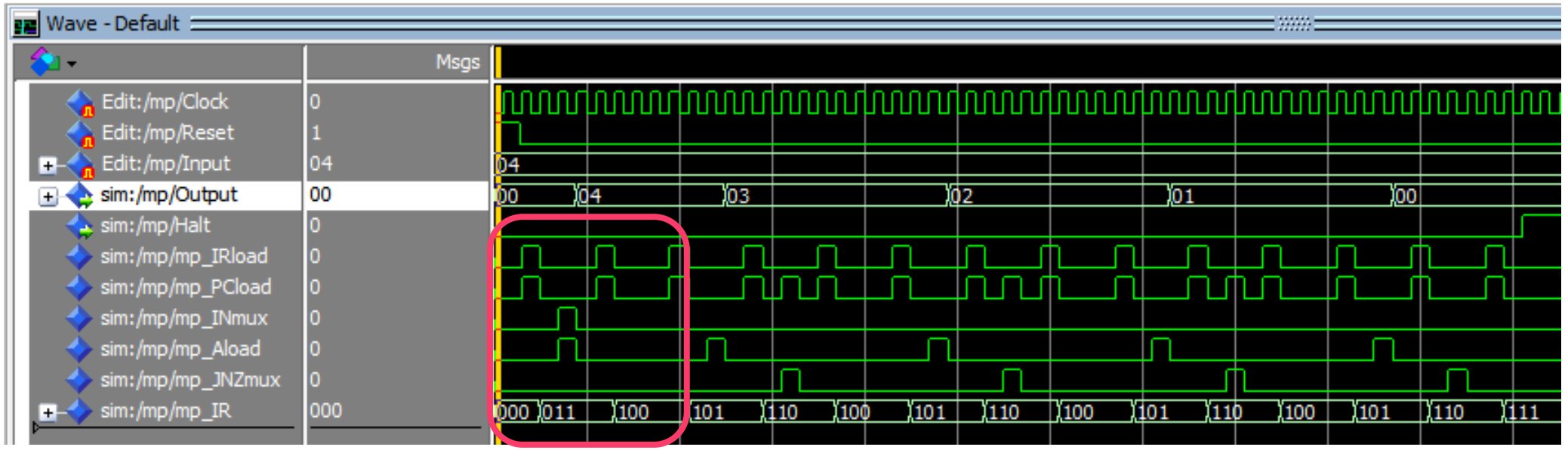
→ 다음은 위의 Example Simulation Capture와 동일하게 진행한 Simulation이다.

→ Reset을 통해 초기 상태를 000으로 한 후, IRload와 PCload가 1이 되었으므로 fetch 단계가 되고,

다음에 전부 0이어서 Decode 단계를 거쳐, INmux와 Aload가 1인 Input 단계가 된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

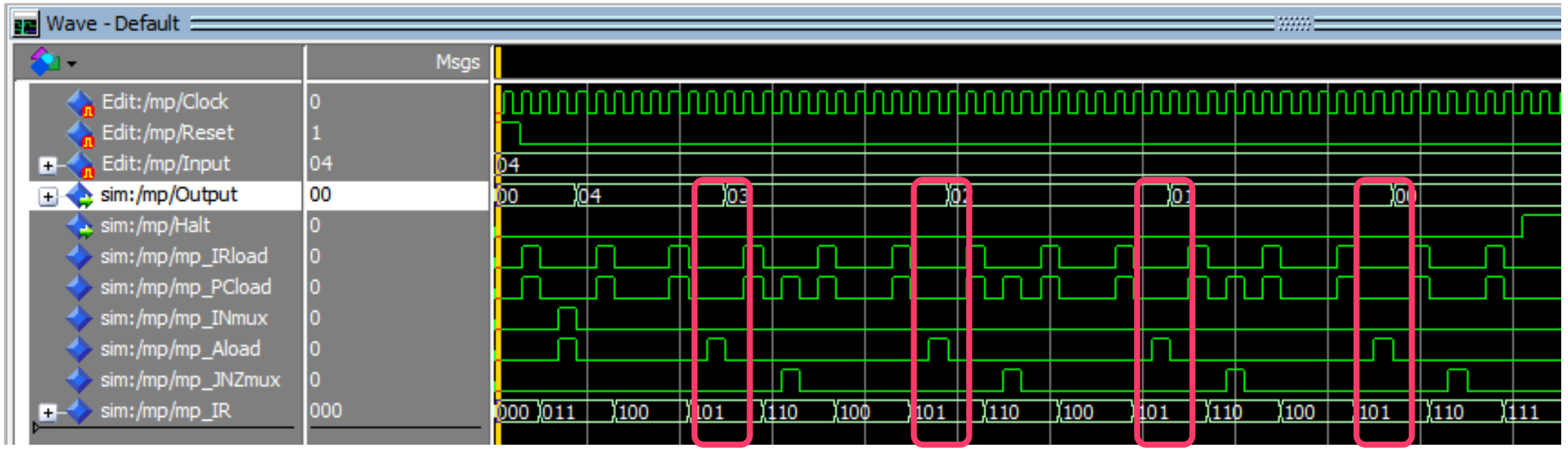
3. Simulation



→ 이때, mp_IR(State)은 011이다. Input 단계 후에 전부 0이므로 Output 단계가 되어서 04를 Output 으로 출력하게 되는데 이 때 mp_IR(State)은 100이다. 그러고 다시 IRload와 PCload가 1이 되어 fetch 단계가 된다. 해당 Simulation은 위의 과정을 반복하면서 진행된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

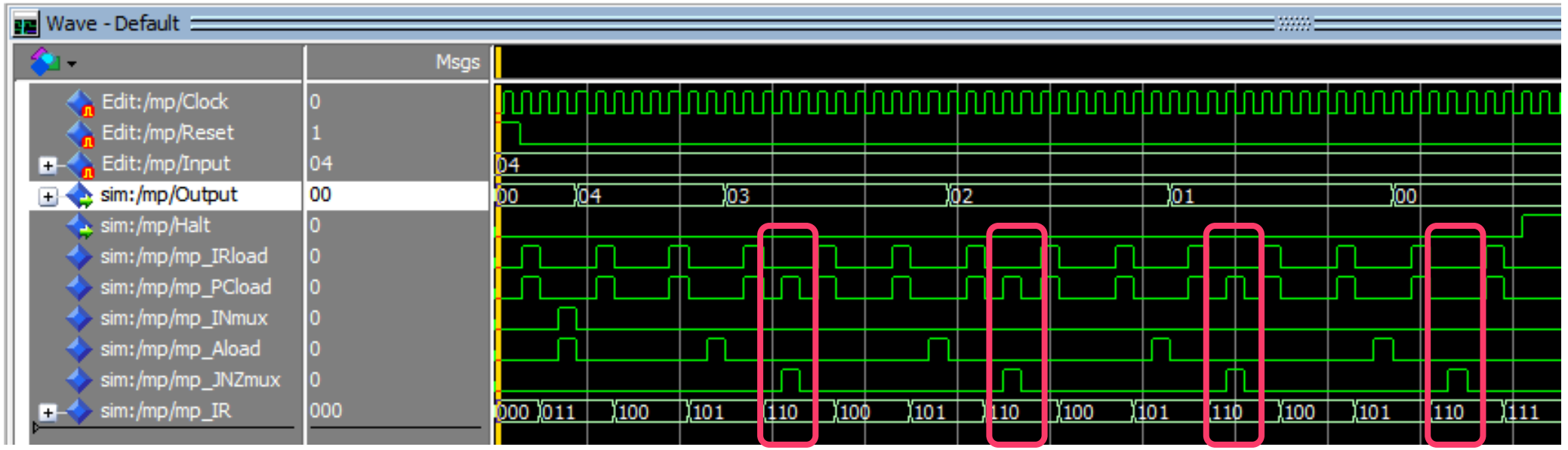
3. Simulation



→ 이와 같이 Aload값만 1이 되는 경우는 Decrement 단계로 mp_IR(State) 101에 해당한다. 다음 단계에서는 위의 Simulation과 같이 값을 1을 줄여주는 작업을 진행하게 된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

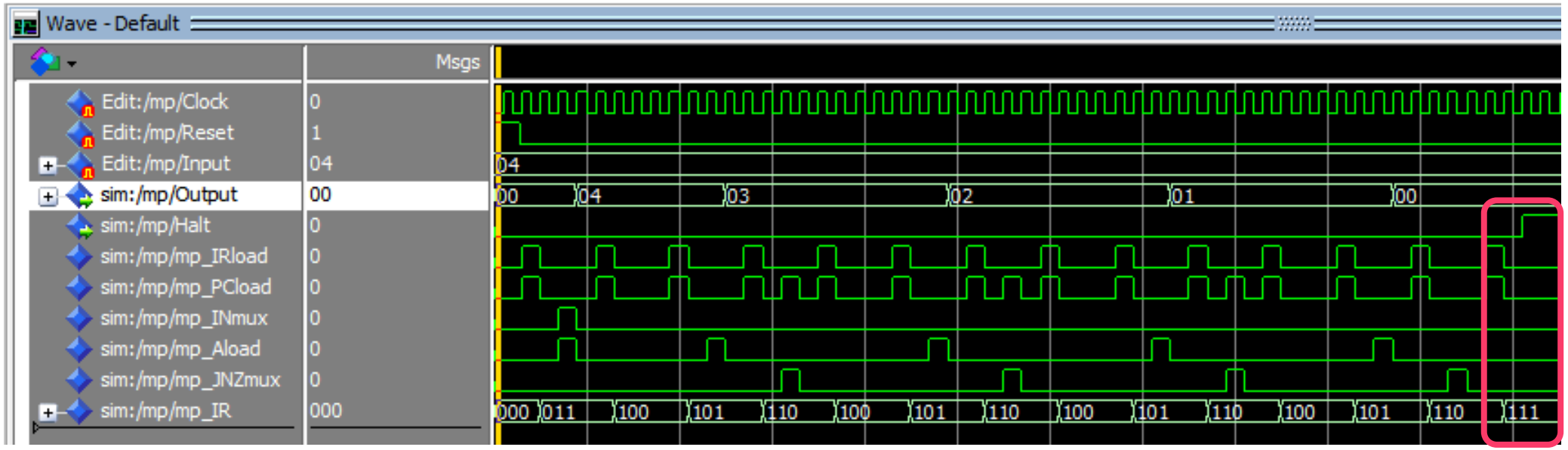
3. Simulation



→ 이와 같이 JNZmux가 1이 되면서 A가 0인지 아닌지를 검사하고, A가 0이 아니면 PCload 값에 다음과 같이 1을 반환해주었고, A가 0이 되는 마지막 부분에서 PCload가 0이 되었다. 해당 단계의 mp_IR(State)는 110이다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

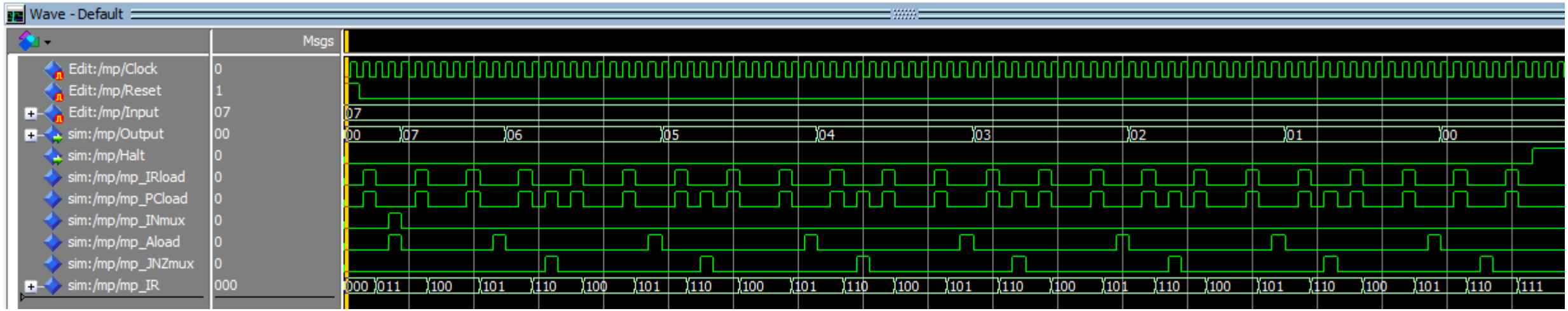
3. Simulation



→ State 110에서 JNZmux를 제외한 모든 값이 1이므로 A가 0값이 되었다는 것을 알게 되었다. 따라서 A가 0이므로 mp_IR(State)는 111이 되고, 자동으로 Halt가 1이 되면서 프로그램이 종료하게 된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

3. Simulation

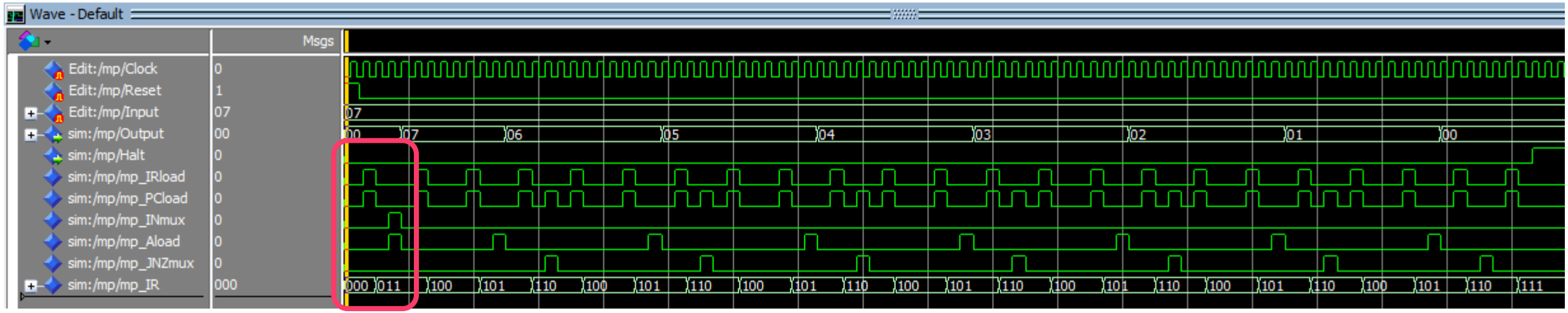


→ 다음은 동일한 Simulation을 적절한 수 로 바꿔서 진행하였다.

→ 해당 Simulation은 Input이 7인 경우이다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

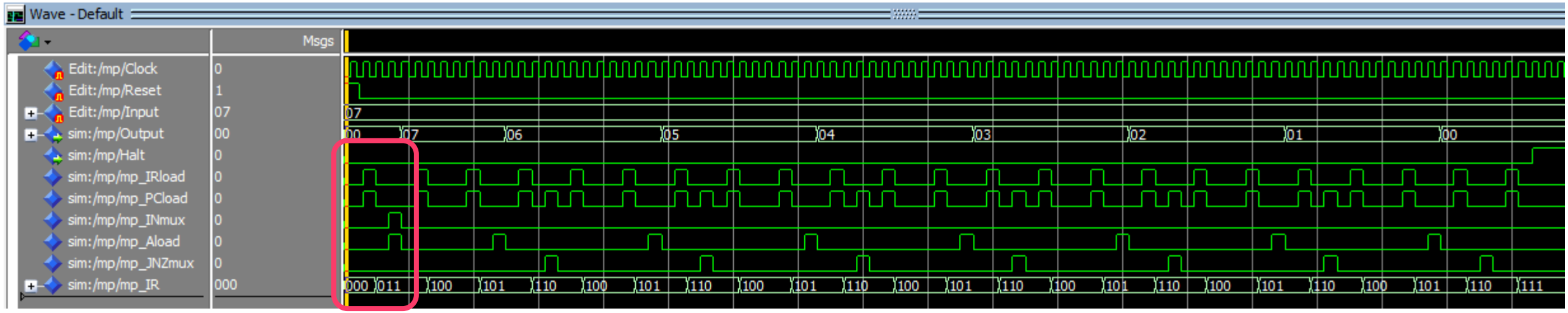
3. Simulation



→ 해당 Simulation도 앞의 Simulation과 동일하게 Reset을 통해 초기 상태를 000으로 한 후, IRload와 PCload가 1이 되었으므로 fetch 단계가 되고, 다음에 전부 0이어서 Decode 단계를 거쳐, INmux와 Aload가 1인 Input 단계가 된다. 이때, mp_IR(State)은 011이다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

3. Simulation

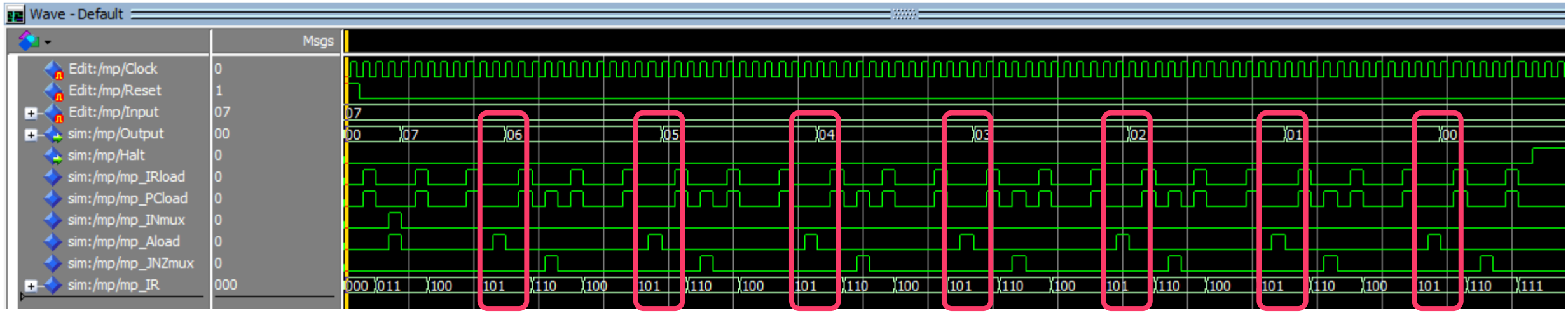


→ 그리고 Input 단계 후에 전부 0이므로 Output 단계가 되어서 07을 Output으로 출력하게 되는데 이 때 mp_IR(State)은 100이다. 그러고 다시 IRload와 PCload가 1이 되어 fetch 단계가 된다.

→ 해당 Simulation도 마찬가지로 위의 과정을 반복하면서 진행된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

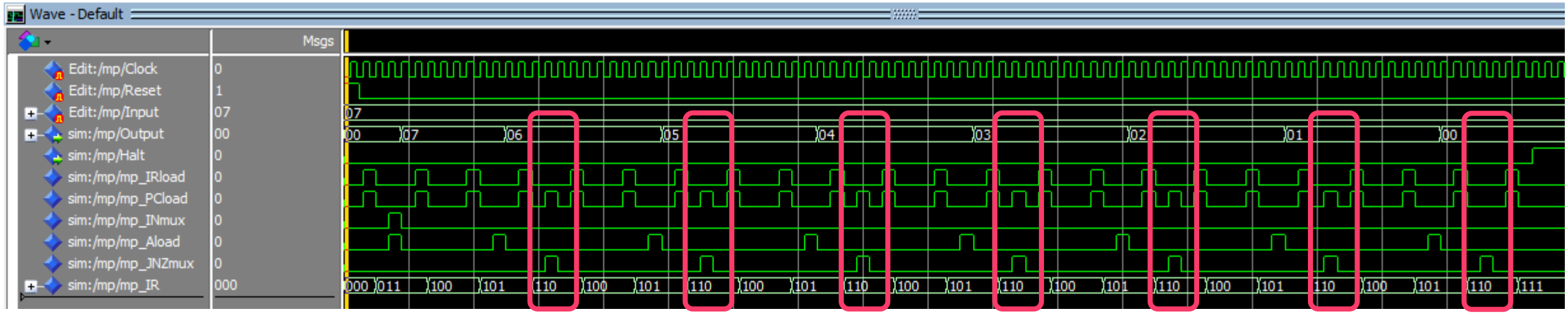
3. Simulation



→ 이 Simulation에서도 Aload값만 1이 되는 경우가 존재한다. 해당 부분은 Decrement 단계로 mp_IR(State)는 101에 해당한다. 다음 단계에서는 위의 Simulation과 같이 값을 1을 줄여주는 작업을 진행하게 된다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

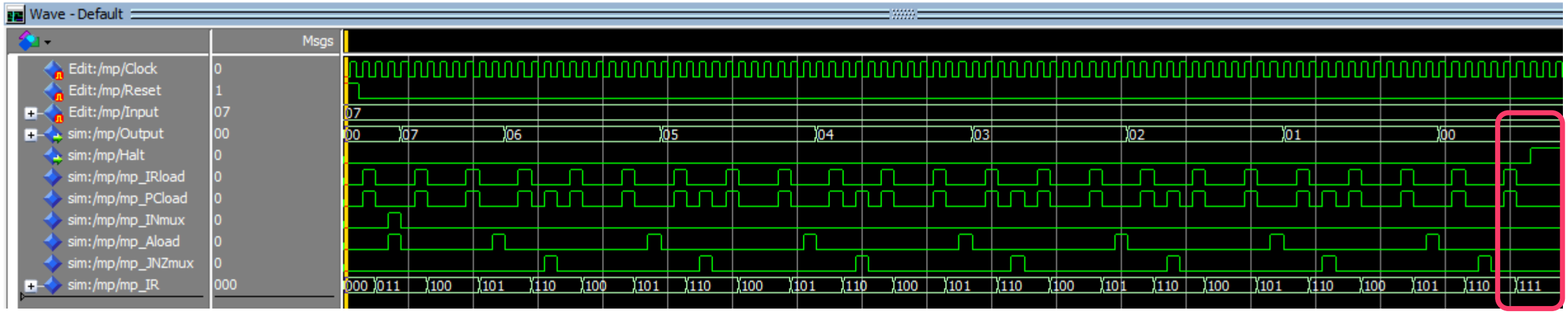
3. Simulation



→ 이 Simulation에서도 JNZmux가 1이 되는 경우가 존재한다. 이 때, A가 0인지 아닌지를 검사하고, A가 0이 아니면 PCload 값에 다음과 같이 1을 반환해주었고, A가 0이 되는 마지막 부분에서 PCload가 0이 되었다. 해당 단계의 mp_IR(State)는 110이다.

Control Word	State $Q_2Q_1Q_0$	*IRload*	*PCload*	*INmux*	*Aload*	*JNZmux*	*Halt*
0	000 *Start*	0	0	0	0	0	0
1	001 *Fetch*	1	1	0	0	0	0
2	010 *Decode*	0	0	0	0	0	0
3	011 *Input*	0	0	1	1	0	0
4	100 *Output*	0	0	0	0	0	0
5	101 *Dec*	0	0	0	1	0	0
6	110 *Jnz*	0	IF ($A \neq 0$) THEN 1 ELSE 0	0	0	1	0
7	111 *Halt*	0	0	0	0	0	1

3. Simulation



→ 이 Simulation에서 마지막 부분에 State 110에서 JNZmux를 제외한 모든 값이 1이 되는 경우가 존재한다. 이 때, A는 0값이라는 것을 알게 되었으므로 mp_IR(State)는 111이 되고, 자동으로 Halt가 1이 되면서 프로그램이 종료하게 된다.

Lab 09. General CPU design 1

Discussion

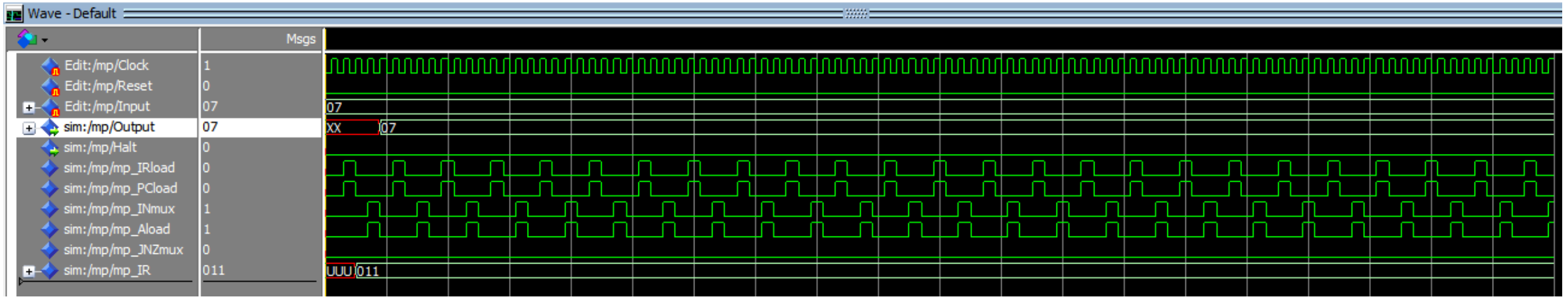
- Discussion

→ Simulation을 진행하면서 이미 Coding이 된 코드를 가지고 진행하였기 때문에 큰 어려움을 겪진 않았다. 다만, Reset을 앞부분에 1로 해주지 않거나 너무 큰 값으로 Simulation을 진행해서 결과를 보지 못하는 등의 문제로 인해 여러 번 Simulation을 진행하게 되었다.

→ 해당 문제는 적당한 값을 Input으로 주고, Reset 앞부분을 반드시 1로 해주며 문제를 해결하게 되었다.

→ 해당 문제를 해결하는 데 어려움은 없었지만, 다음과 같은 호기심이 생기게 되었다.

Discussion

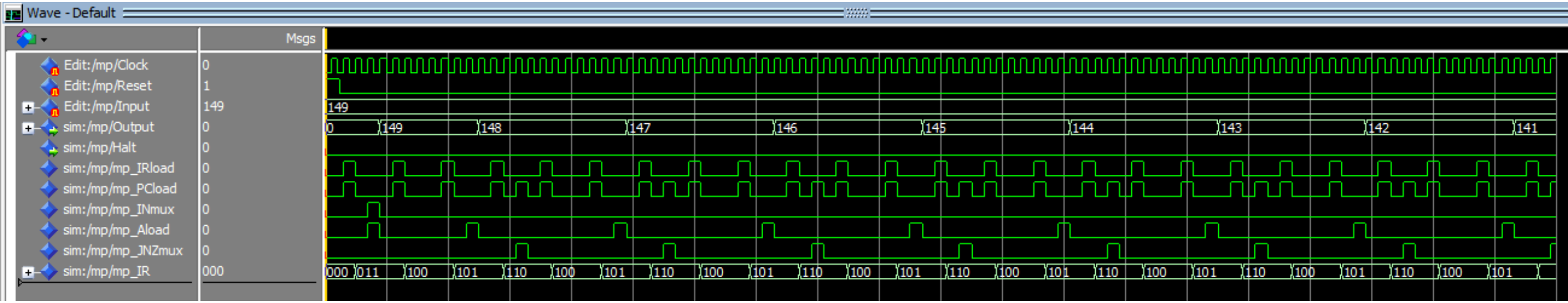


→ 다음은 Reset 앞에 1을 해주지 않아 오류가 난 코드이다.

→ 해당 오류가 왜 일어났는지에 대해서는 자세하게 모르지만, Reset을 할 때는 항상 주의하여 Simulation을 진행하여야 한다. 이를 통해 해당 오류가 나는 이유가 무엇인지 궁금해지게 되었다.

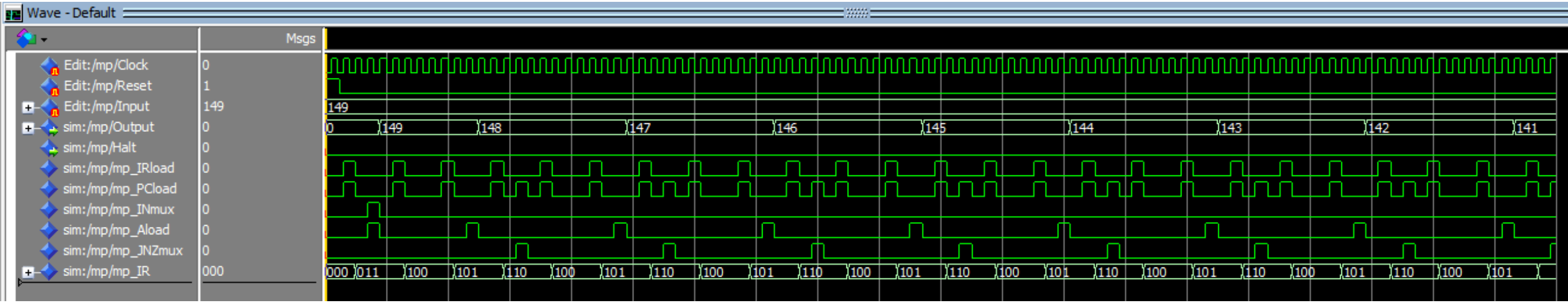
→ 하지만, 검색을 통해서도 해당 문제가 왜 일어나는지 알 수 없었다.

Discussion



→ 다음은 너무 큰 값으로 Simulation을 진행하여서 끝까지 결과가 나오지 않고, 중간에 Simulation이 종료된 경우이다. 해당 경우에는 Clock을 더 짧게 하여 해결하는 방법이 존재하지만, 이미 Simulation을 하면서 충분히 작은 Clock을 주어서 더 이상은 무리라고 판단하였다.

Discussion



→ 이와 같은 문제가 있는 경우에 끝까지 결과를 확인하기 위해서는 어떻게 Simulation을 진행하여 해결하는지 궁금하다.

- Discussion

→ 다음과 같은 호기심을 남겨두고 해당 실습을 마무리 지었다.

→ 이번 실습은 VHDL Code가 주어진 상태에서 내가 이해를 하고, Simulation을 진행하여 이를 분석하는 실습이라 비교적 어렵지 않았던 것 같다.하지만, 내가 직접 해당 Design의 VHDL Code를 구현하기는 아직 많은 어려움이 있다고 느껴진다.

→ 또한, 이번 수업시간에 언급되었듯이 직접 KIT를 통해 실습을 진행하면, 어떤 결과가 나오게 될지도 궁금해졌다.