Experiment 7 - Week 1

- Week 1
  - Test the sequencer and able to increment PC.
  - Develop the RTL statements for the CPU instructions. (submit as post-lab)

CPU Instruction (Fall 2025)

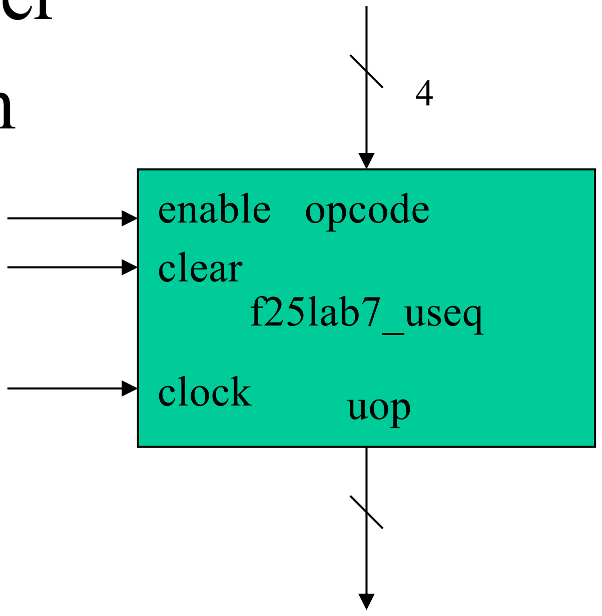
Instruction	Instruction code	Operation
NOP	00000000	No operation
LOADI Rn,X	0001000n X	Rn $\longleftarrow$ X
LOAD Rn,X	0010000n X	Rn $\longleftarrow$ M[X]
STORE X,Rm	0011000m X	M[X] $\longleftarrow$ Rm
MOVE Rn,Rm	0100000n	Rn $\longleftarrow$ Rm, m $\neq$ n
ADD Rn,Rm	0101000n	Rn $\longleftarrow$ Rn + Rm, m $\neq$ n
SUB Rn,Rm	0110000n	Rn $\longleftarrow$ Rn – Rm, m $\neq$ n
TESTNZ Rm	0111000m	Z $\longleftarrow$ not V, V = OR of the bits of Rm
TESTZ Rm	1000000m	Z $\longleftarrow$ V, V = OR of the bits of Rm
JUMP X	10010000 X	PC $\longleftarrow$ X
JUMPZ X	10100000 X	If (Z = 1) then PC $\longleftarrow$ X
LOADSP X	10110000 X	SP $\longleftarrow$ X
PEEP Rn	1100000n	Rn $\longleftarrow$ M[SP]
PUSH Rn	1101000n	M[--SP] $\longleftarrow$ Rn Pre-increment
POP Rn	1110000n	Rn $\longleftarrow$ M[SP++] Post-decrement
HALT	11110000	PC $\longleftarrow$ 0, stop microsequencer

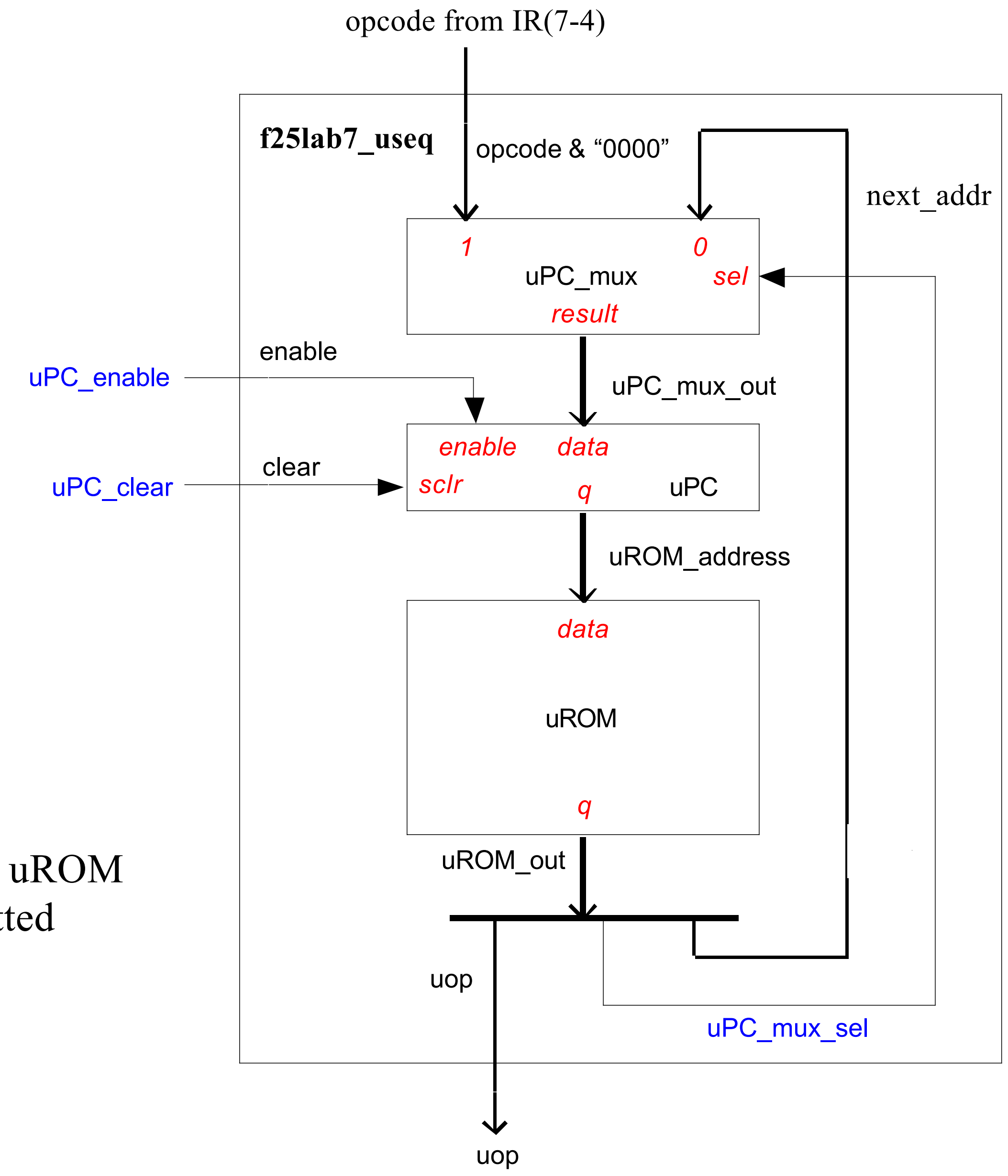
X = 8-bit data or address

m, n = 0 or 1, m $\neq$ n

Part 1A - Microsequencer

- Clock - system clock
- Enable - enables microsequencer
- Clear - reset the uPC register in the microsequencer to 0
- Opcode - 4-bit opcode
- Requires two clock cycle



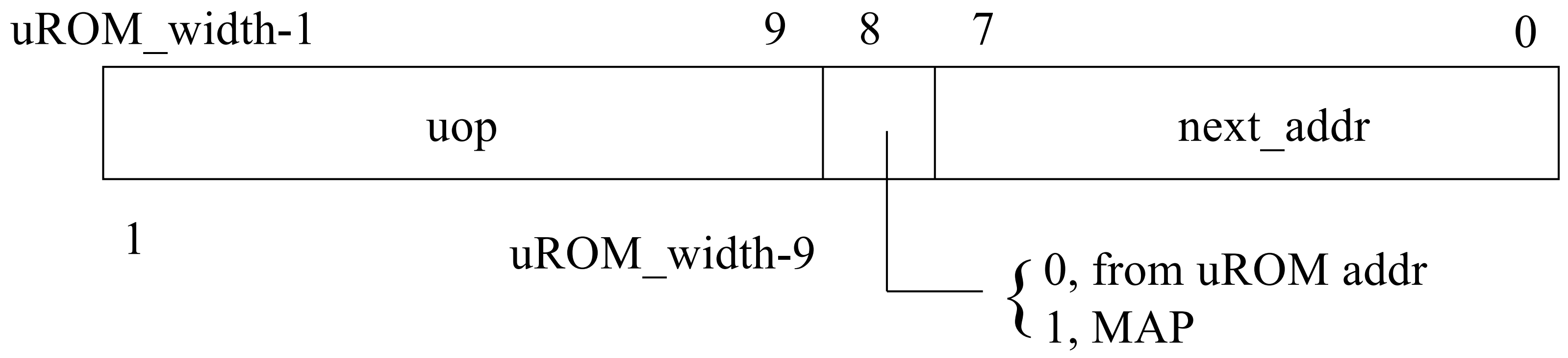


Clock signals for uROM and uPC are omitted

Micro-ROM

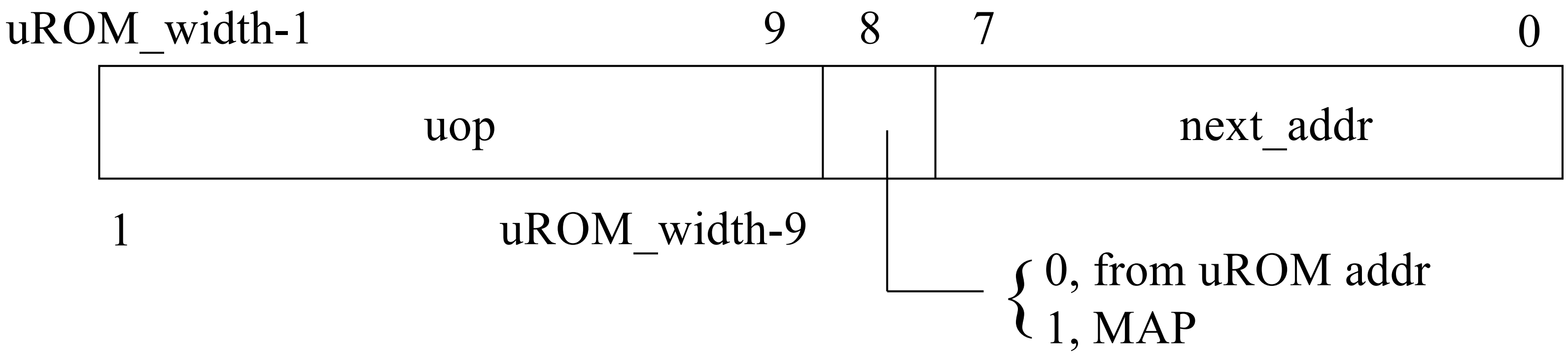
- 256 words
- Width of each micro-ROM word is uROM_width
- uROM_file is the name of the mif with the content of the micro-ROM

Micro-ROM format



Component Declaration

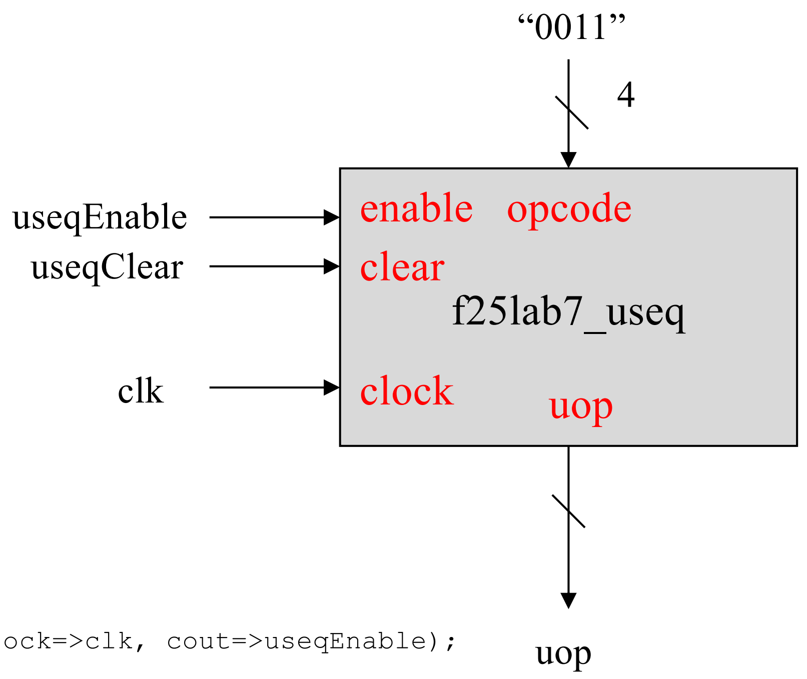
```
component f25lab7_useq
generic (uROM_width: integer;      -- uROM width
         uROM_file: string);       -- uROM file path
port (opcode: in std_logic_vector(3 downto 0);
      uop: out std_logic_vector(1 to (uROM_width-9));
      debug_map_addr: out std_logic_vector(8 downto 0);   -- for debugging
      enable, clear, clock: in std_logic);
end component;
```



Part 1A – Test Microsequencer

Create a project for lab7 Part 1A with following code:

```
library ieee; use ieee.std_logic_1164.all;
library lpm; use lpm.lpm_components.all;
entity lab7Part1A is
port (clk, clear: in std_logic;
      useqEnOut: out std_logic;
      ctrlSignals: out std_logic_vector(0 to 7));
end lab7Part1A;
architecture structural of lab7Part1A is

-- add component declaration of f25lab7_useq

  signal useqEnable, useqClear: std_logic;
  signal uop: std_logic_vector(0 to 7);

begin
  delay: lpm_counter generic map(lpm_width=>2) port map(clock=>clk, cout=>useqEnable);
  useqClear <= clear;
  useqEnOut <= useqEnable;
  labelG: for i in 0 to 7 generate
    ctrlSignals(i) <= uop(i) and useqEnable;
  end generate;

  -- add port map statement of f25lab7_useq, use the uROM Test file in the next slide

end structural;
```



Part 1A – uROM Test File

```
-- uROM Test File 1A
-- 8-bit control signals

WIDTH=17;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
  [0..FF]: 00000000000000000;
--       uop
--     01234567M01234567
  00: 00000000000000001; -- control signal = 00000000, next address = 0x01

  01: 00000001000000010; -- control signal = 00000001, next address = 0x02
  02: 00000010000000011; -- control signal = 00000010, next address = 0x03
  03: 00000011000000100; -- control signal = 00000011, next address = 0x04
  04: 00000100100000000; -- control signal = 00000100, use MAP, go to address 0x30

  30: 00000101000110001; -- control signal = 00000101, next address = 0x31
  31: 00000110000110010; -- control signal = 00000110, next address = 0x32
  32: 00000111000110011; -- control signal = 00000111, next address = 0x33
  33: 00001000000000001; -- control signal = 00001000, next address = 0x01

END;
```

control signal

mux select

next address

- Assign
  - clk = key3, clear = sw0, useqEnOut = ledg7
  - ctrlSignals(0) = ledr7, ctrlSignals(1) = ledr6, …, ctrlSignals(7) = ledr0
- When useqEnOut goes high, you should see

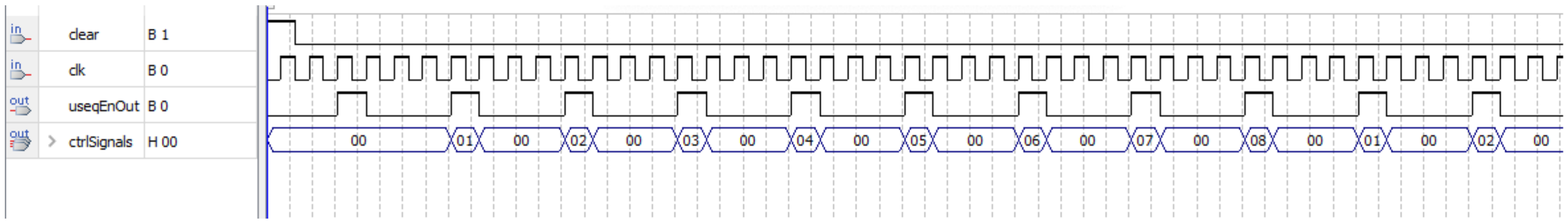
  ledr(7-0) =

  "00000000",

  "00000001", "00000010", "00000011", "00000100"

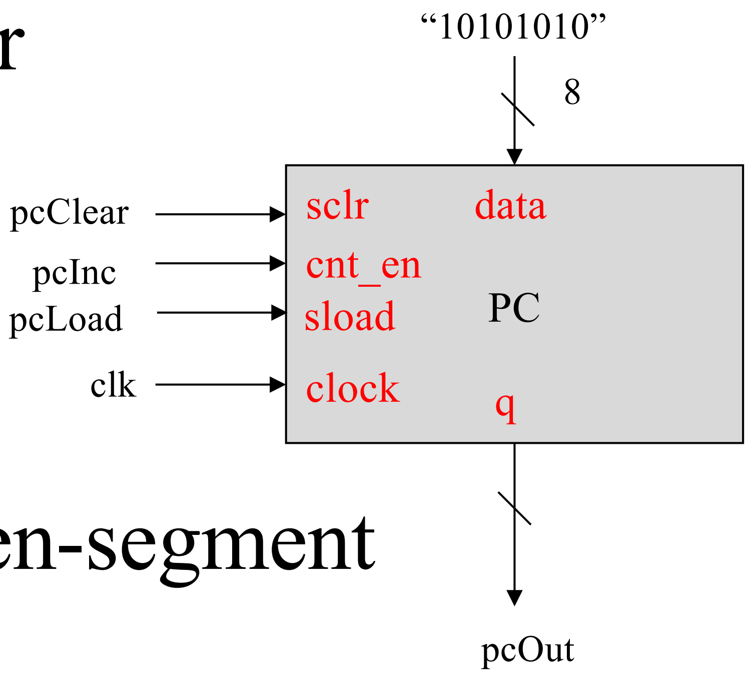
  "00000101", "00000110", "00000111", "00001000"

repeat



Part 1B -Use Microsequencer to clear, load and increment PC

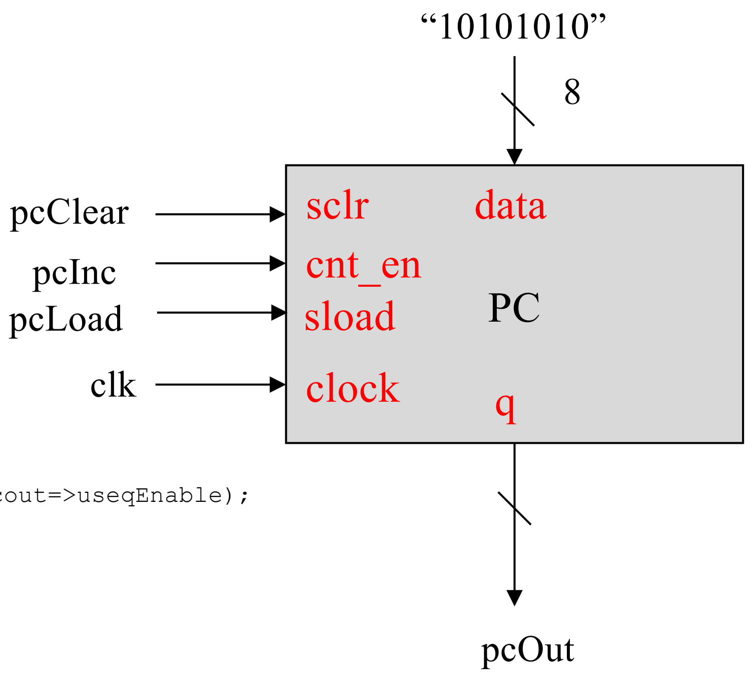
- Add PC using lpm_counter
- Use uop(0) as pcInc
- Use uop(1) as pcLoad
- Use uop(2) as pcClear
- Display PC using two seven-segment displays



Part 1B

Create a project for lab7 Part 1B with following code:

```
library ieee; use ieee.std_logic_1164.all;
library lpm; use lpm.lpm_components.all;
entity lab7Part1B is
port (clk, clear: in std_logic;
      useqEnOut: out std_logic;
      pcSeg1, pcSeg2: out std_logic_vector(0 to 6);
      ctrlSignals: out std_logic_vector(0 to 7));
end lab7Part1B;
architecture structural of lab7Part1B is

-- add component declaration of f25lab7_useq
-- add component declaration for seven-segment decoder

  signal useqEnable, useqClear: std_logic;
  signal uop: std_logic_vector(0 to 7);
  signal pcInc, pcLoad, pcClear: std_logic;

begin
  delay: lpm_counter generic map(lpm_width=>2) port map(clock=>clk, cout=>useqEnable);
  useqClear <= clear;
  useqEnOut <= useqEnable;
  labelG: for i in 0 to 7 generate
    ctrlSignals(i) <= uop(i) and useqEnable;
  end generate;
  pcInc <= uop(0) and useqEnable;
  pcLoad <= uop(1) and useqEnable;
  pcClear <= uop(2) and useqEnable;

  -- add port map statement of f25lab7_useq, use the uROm Test file in the next slide
  -- add pc register, seven-segment display

end structural;
```



Part 1B – uROM Test File

```
-- uROM Test File 1B
-- 8-bit control signals

WIDTH=17;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
  [0..FF]: 00000000000000000;
--      uop
--    01234567M01234567
  00: 00000000000000001; -- control signal = 00000000, next address = 0x01

  01: 00100001000000010; -- control signal = 00100001, next address = 0x02, clear PC, PC = 0x00
  02: 10000010000000011; -- control signal = 10000010, next address = 0x03, inc PC, PC = 0x01
  03: 10000011000000100; -- control signal = 10000011, next address = 0x04, inc PC, PC = 0x02
  04: 00000100100000000; -- control signal = 00000100, use MAP, go to address 0x30

  30: 01000101000110001; -- control signal = 01000101, next address = 0x31, load PC, PC = 0xAA
  31: 00000110000110010; -- control signal = 00000110, next address = 0x32
  32: 10000111000110011; -- control signal = 10000111, next address = 0x33, inc PC, PC = 0xAB
  33: 00001000000000001; -- control signal = 00001000, next address = 0x01,
END;
```

🟨 control signal

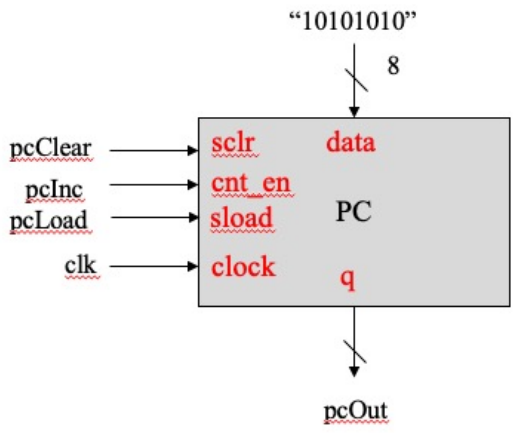
🟥 mux select

🟩 next address

Part 1B – uROM Test File

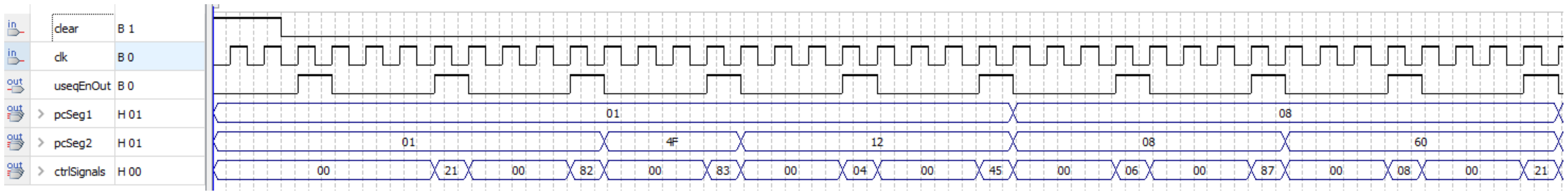
```
-- uROM Test File 1B
-- 8-bit control signals

WIDTH=17;
DEPTH=256;
ADDRESS_RADIX=HEX;
DATA_RADIX=BIN;

CONTENT BEGIN
  [0..FF]: 00000000000000000;
--      uop
--    01234567M01234567
  00: 00000000000000001; -- control signal = 00000000, next address = 0x01

  01: 00100001000000010; -- control signal = 00100001, next address = 0x02, clear PC, PC = 0x00
  02: 10000010000000011; -- control signal = 10000010, next address = 0x03, inc PC, PC = 0x01
  03: 10000011000000100; -- control signal = 10000011, next address = 0x04, inc PC, PC = 0x02
  04: 00000100100000000; -- control signal = 00000100, use MAP, go to address 0x30

  30: 01000101000110001; -- control signal = 01000101, next address = 0x31, load PC, PC = 0xAA
  31: 00000110000110010; -- control signal = 00000110, next address = 0x32
  32: 10000111000110011; -- control signal = 10000111, next address = 0x33, inc PC, PC = 0xAB
  33: 00001000000000001; -- control signal = 00001000, next address = 0x01,
END;
```

- uop(0) = pcInc
- uop(1) = pcLoad
- uop(2) = pcClear



- Assign
  - clk = key3, clear = sw0, useqEnOut = ledg7
  - ctrlSignals(0) = ledr7, ctrlSignals(1) – ledr6, …, ctrlSignals(7) = ledr0
  - pcSeg1 = hex0, pcSeg2 = hex1
- You should see

  pcSeg2,pcSeg1 = 0x00, 0x01, 0x02, 0xAA, 0xAB, …



Assignments

- Week 1
  - Test the sequencer and able to increment PC.
  - Develop the RTL statements for the CPU instructions. (submit as post-lab)