Chapter 2

Number Sequencing Computer (NSC)

Yu Luo
Assistant Professor
Department of Electrical and Computer Engineering
Office: 239 Simrall Engineering Building
Email: yu.luo@ece.msstate.edu

Different Digit Systems

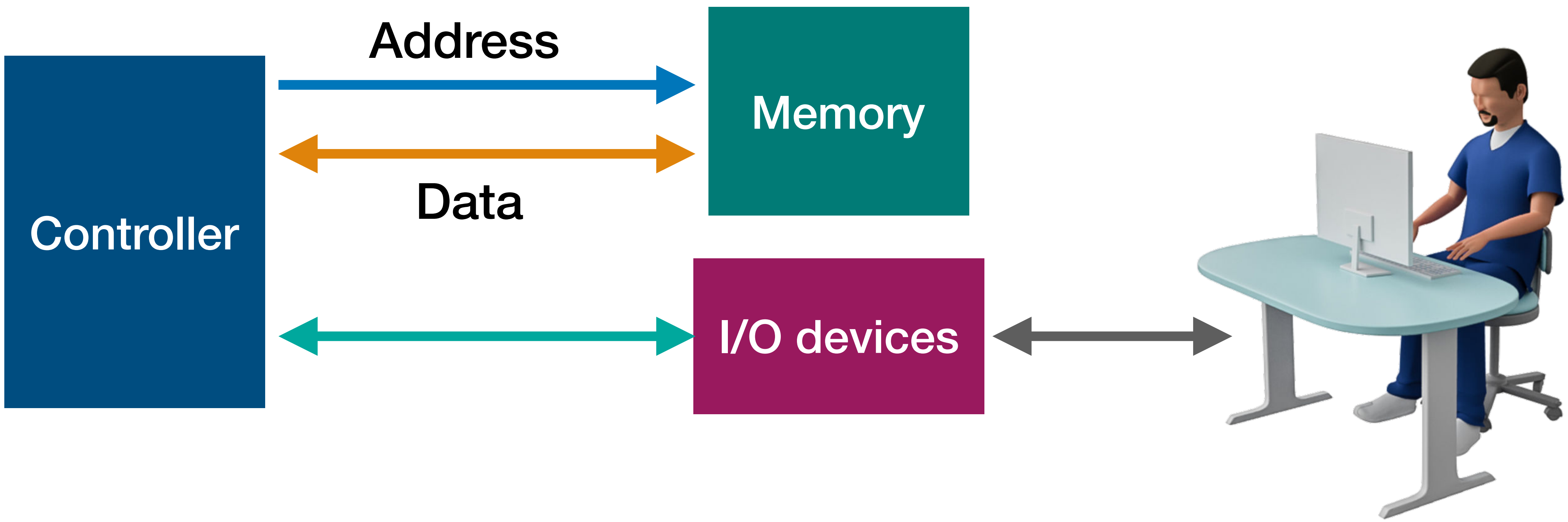
- Special purpose digit system
  A. Designed for **special** problem
  B. Different problems need different systems
  C. High-efficient, e.g., high end graphics card, calculator, communication chip

- General purpose digit system
  A. Can solve **multiple different** problems
  B. Low-efficient (relatively speed and high cost), e.g., CPU

Computer

- Computer is a general purpose digit system
- Operation can be specified via a program
    A. Program is a sequence of binary codes that represent instructions for the computer
    B. Program is stored in a Memory
- Changing the program changes the computer behavior to solve different problems
- External inputs (e.g., keyboard and mouse) can also change the behavior the computer.
- Computer can use input and output (I/O) to interact with user

Components of Computer System



- Controller: It **generates logic** to fetch or execute instructions
- Memory: It **stores** instructions and data
- I/O: It is used to interact with user

Number Sequencing Computer (NSC) — Problem Define

1. Assume format of a phone number is: $Y_1Y_2Y_3$-$Z_1Z_2Z_3Z_4$

2. The digital system has one external input called **LOC**.

3. If LOC = 0 , then the system displays full number: $Y_1Y_2Y_3$-$Z_1Z_2Z_3Z_4$.

4. If LOC = 1 , then the system displays only the digits $Z_1Z_2Z_3Z_4$.

NSC — Two Solutions

1. Design a special purpose digit system, such as Finite State Machine
   A. It only works for **one** number sequence
   B. If phone number is changed, the whole system needs to be **redesigned**
2. Computer system
   A. It works for **any** number sequence
   B. If phone number is changed, we need to just **reprogram** the system

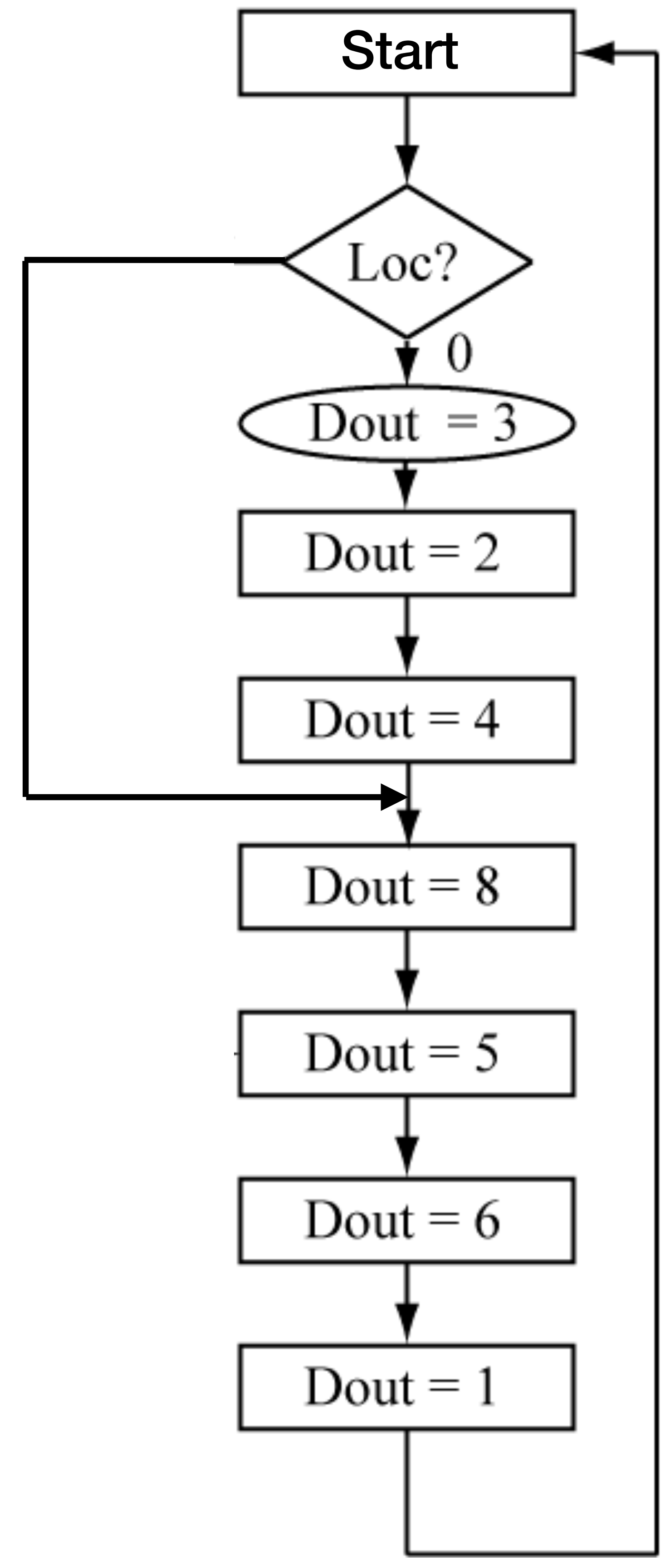
Which one you prefer?

I prefer **the first one because it is easy**

Unfortunately, we need to learn **the second one**

Algorithmic State Machine

Assume the phone number is: 324 - 8561



Pseudo Code for Operations

```
START:
    If LOC ==1, goto LOCAL

        Output = 3;
        Output = 2;
        Output = 4;

LOCAL:
        Output = 8;
        Output = 5;
        Output = 6;
        Output = 1;

        goto START;
```

Instructions We Need

START:

If LOC ==1, goto LOCAL

Output = 3;

Output = 2;

Output = 4;

LOCAL:

Output = 8;

Output = 5;

Output = 6;

Output = 1;

goto START;

1. JC instruction: jump conditionally
   - A. If LOC ==1, jump to LOCAL
   - B. If LOC ==0, fetch next instruction
2. OUT instruction: output phone number
3. JMP instruction: jump unconditionally

START and LOCAL are labels. They are **NOT** instructions !

Assembly Code for Phone Number Display

```
START:
    If LOC ==1, goto LOCAL

      Output = 3;
      Output = 2;
      Output = 4;

LOCAL:
      Output = 8;
      Output = 5;
      Output = 6;
      Output = 1;

      goto START;
```

```
START:
    JC    LOCAL

    OUT   3
    OUT   2
    OUT   4

LOCAL:
    OUT   8
    OUT   5
    OUT   6
    OUT   1

    JMP   START
```

A program written using the **native instructions** of the computer is called an **Assembly** Language Program

Machine Code

- Computer can only understand 1 and 0
- We need to convert assembly code to binary
- Machine code is the binary representation of an instruction
- Machine code has **two** parts
    A. Operation code: it tells what the **instruction** is
    B. Data or memory location: it tells the input/output data or memory location
- Assembly: it is the process of converting instructions to their machine code representation

Example of a Machine Code

- Assume a machine code has **6 bits**
- We let the **first 2 bits** (Bit 4 and Bit 5) represent the operation code
- We let the **last 4 bits** (Bit 0 to Bit 3) represent the data or memory location

Machine code: 0b $\underbrace{01}_{\text{operation code}}$ $\underbrace{1011}_{\text{Data or memory location}}$

Next, we allocation each instruction a unique operation code

Instruction Table

Machine code: 0b 01 1011

operation code — Data or memory location

Next, we allocation each instruction a unique operation code

JMP:	00	00 xxxx	Memory location
JC:	01	01 xxxx	Memory location
OUT:	10	10 xxxx	Data

Convert Program to Machine Code

JMP:	00
JC:	01
OUT:	10

Instruction	Memory Location	Machine Code
START: JC LOCAL	0x00	Memory address of LOCAL 0x01 0100
OUT 3	0x01	0x10 0011
OUT 2	0x02	0x10 0010
OUT 4	0x03	0x10 0100
LOCAL: OUT 8	0x04	0x10 1000
OUT 5	0x05	0x10 0101
OUT 6	0x06	0x10 0110
OUT 1	0x07	0x10 0001
JMP START	0x08	Memory address of START 0x00 0000

Length of Machine Code

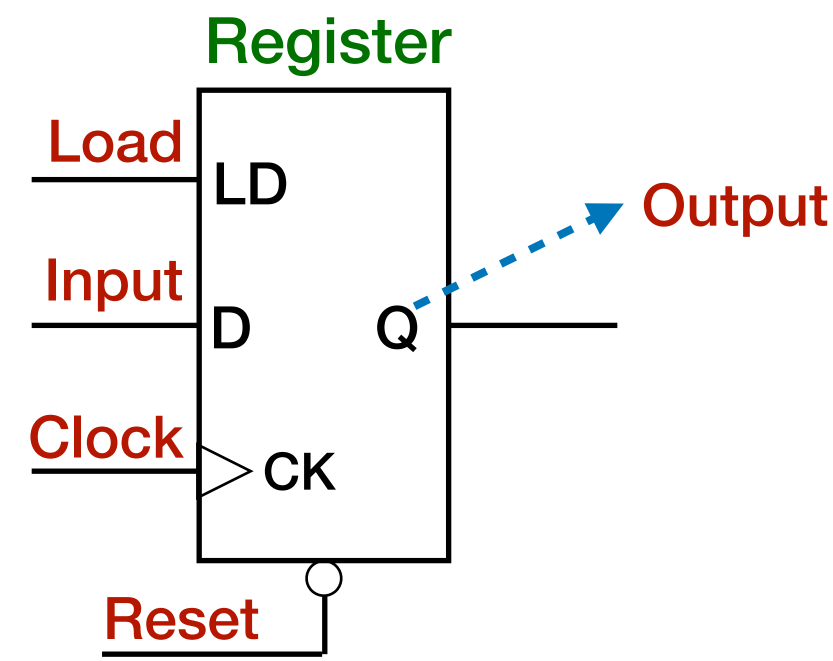
Machine code: 0b $\underbrace{01}_{\text{operation code}}$ $\underbrace{1011}_{\text{Data or memory location}}$

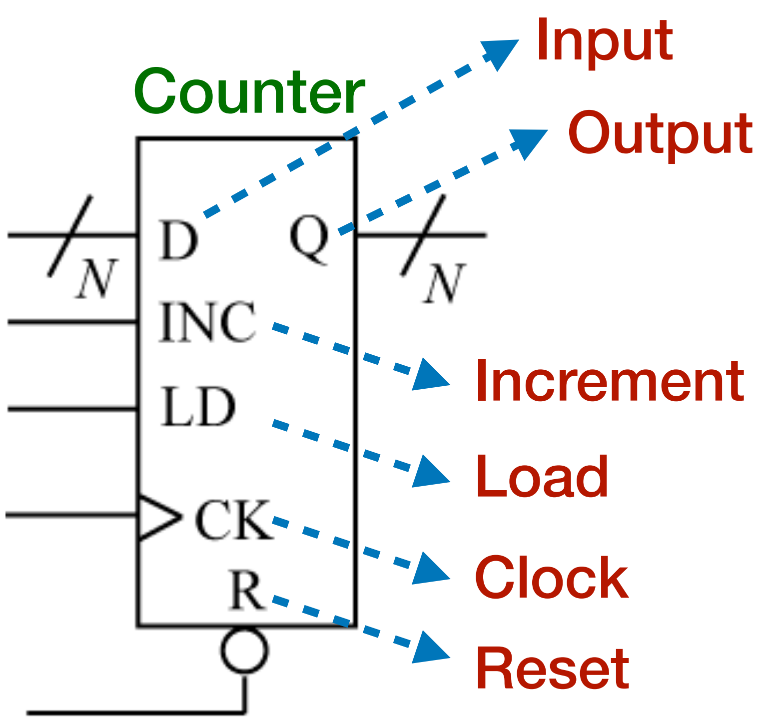
2 bits operation code can only represent **4 different** instructions

Have more instructions? —> Extend the code to 4-bit, 8-bit, …

- A PIC33/PIC24 instruction is **24 bits** wide (3 bytes)
    A. 8 bits opcode
    B. 16 bits data or memory locations

Counter and Register

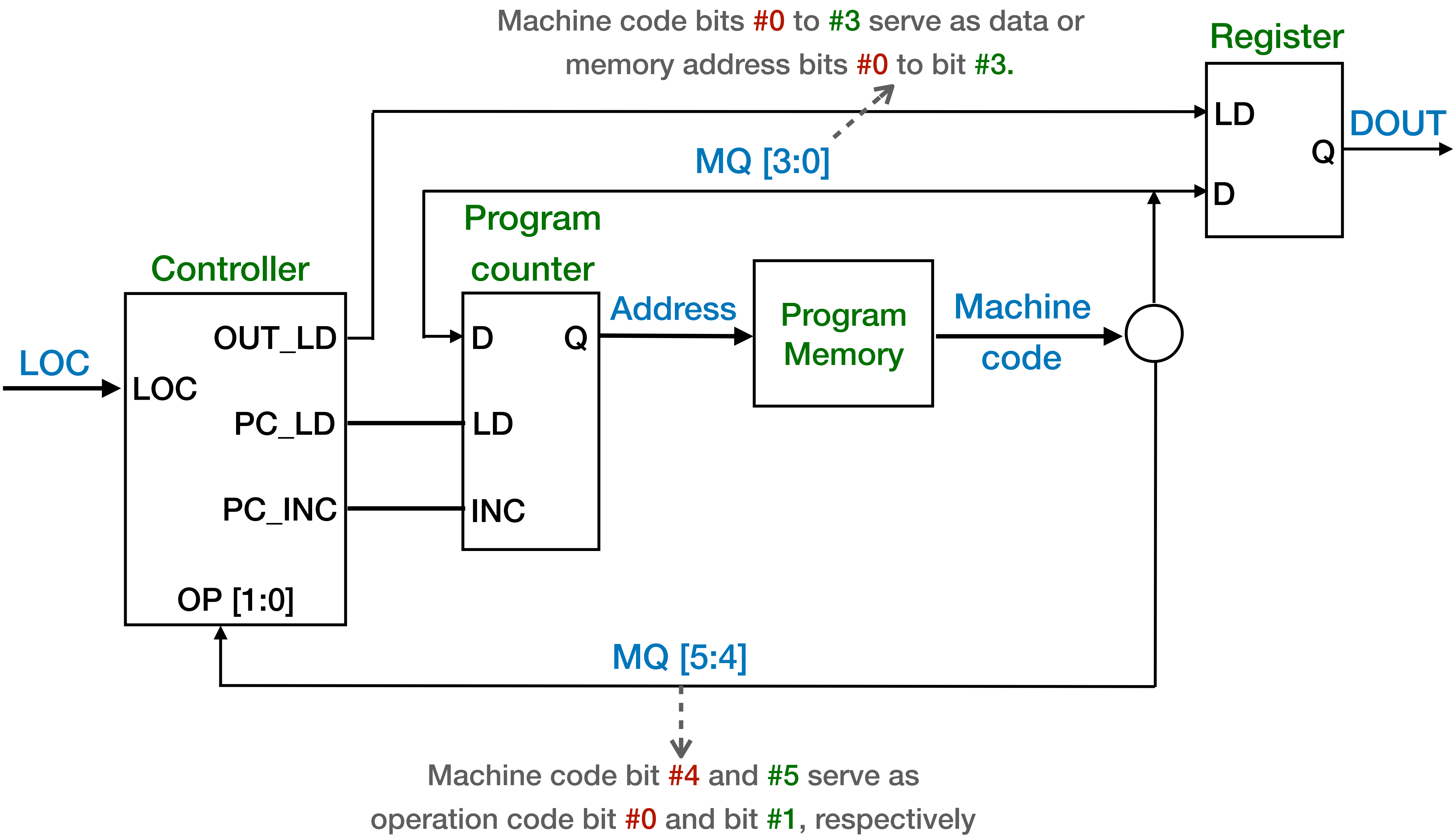




- If LD = 0 —> Output remains unchanged
- If LD = 1 —> Output = Input
- If R = 0 —> Output = 0

- Increment mode: If INC = 1 and LD = 0 —> Output = Output +1
- Load mode: If INC = 0 and LD = 1 —> Output = Input
- Rest: If R = 0 —> Output = 0

NSC Implementation (1)



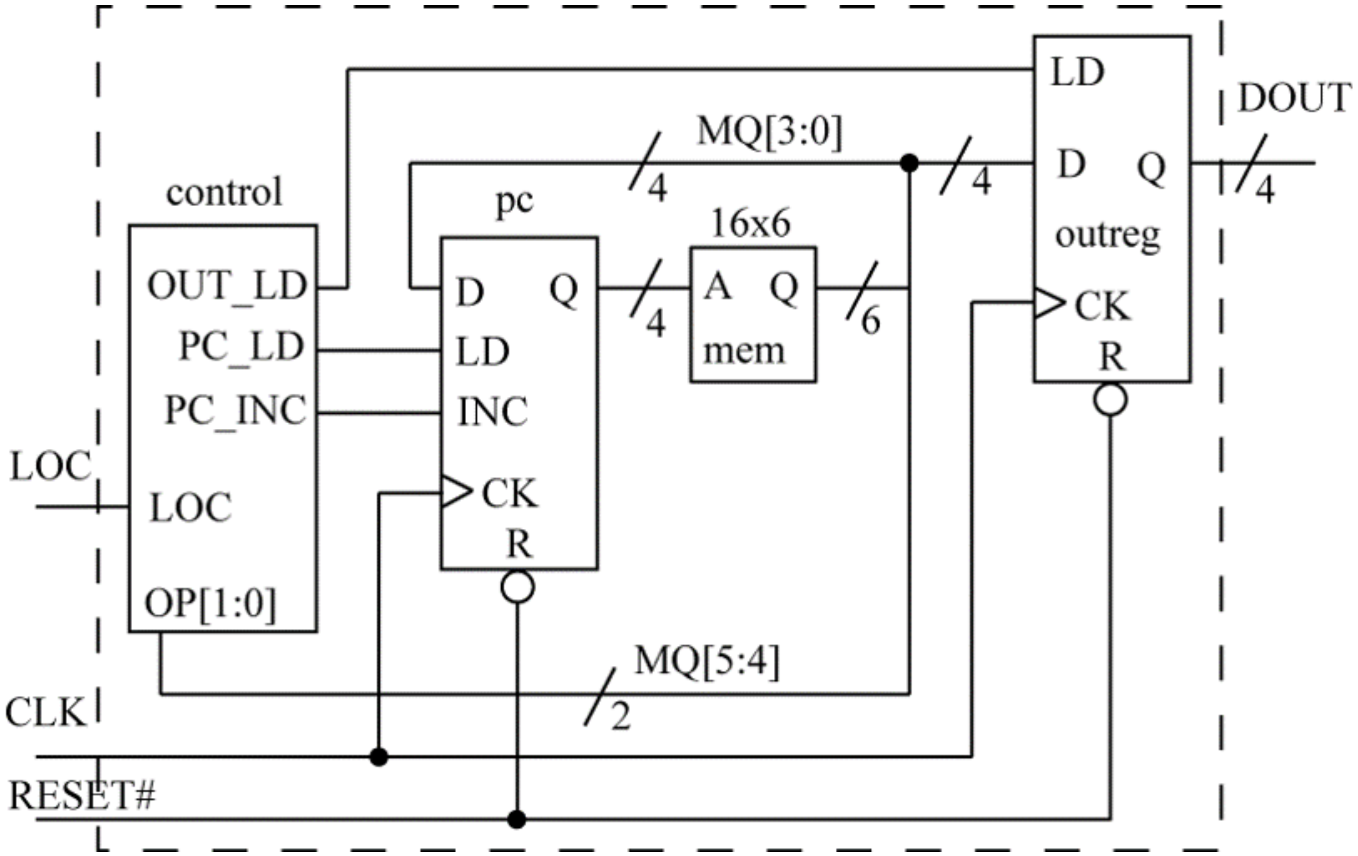
NSC Implementation (2)

- Controller: It based on inputs: LOC and operation code (OP[0:1]), controls OUT_LD, PL_LD, and PC_INC

- Program counter:
  A. If INC = 1 and LD = 0 —> Output = Output +1 —> The address of next instruction fetched from the program memory = current address + 1
  B. If INC = 0 and LD = 1 —> Output = Input —> The address of next instruction fetched from the program memory = input = MQ [3:0]

- Register:
  A. If LD = 0 —> DOUT remains unchanged
  B. If LD = 1 —> DOUT = input = MQ [3:0]

NSC Example (1)



- PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
- PC_INC = !PC_LD
- OUT_LD = ~OP[0] & ~OP[1]

Address	Machine code
0	101000
1	100101
2	000001
3	010000
4	011001
5	000101

- **If the current address is 0, LOC = 1, and DOUT = 0b1110, what is the value of DOUT and the next address to execute after running this instruction?**

A. DOUT =
B. Next address =

Evaluating the output of the control logic where `OP[1:0]` = 0b10 and `LOC` = 1 gives:

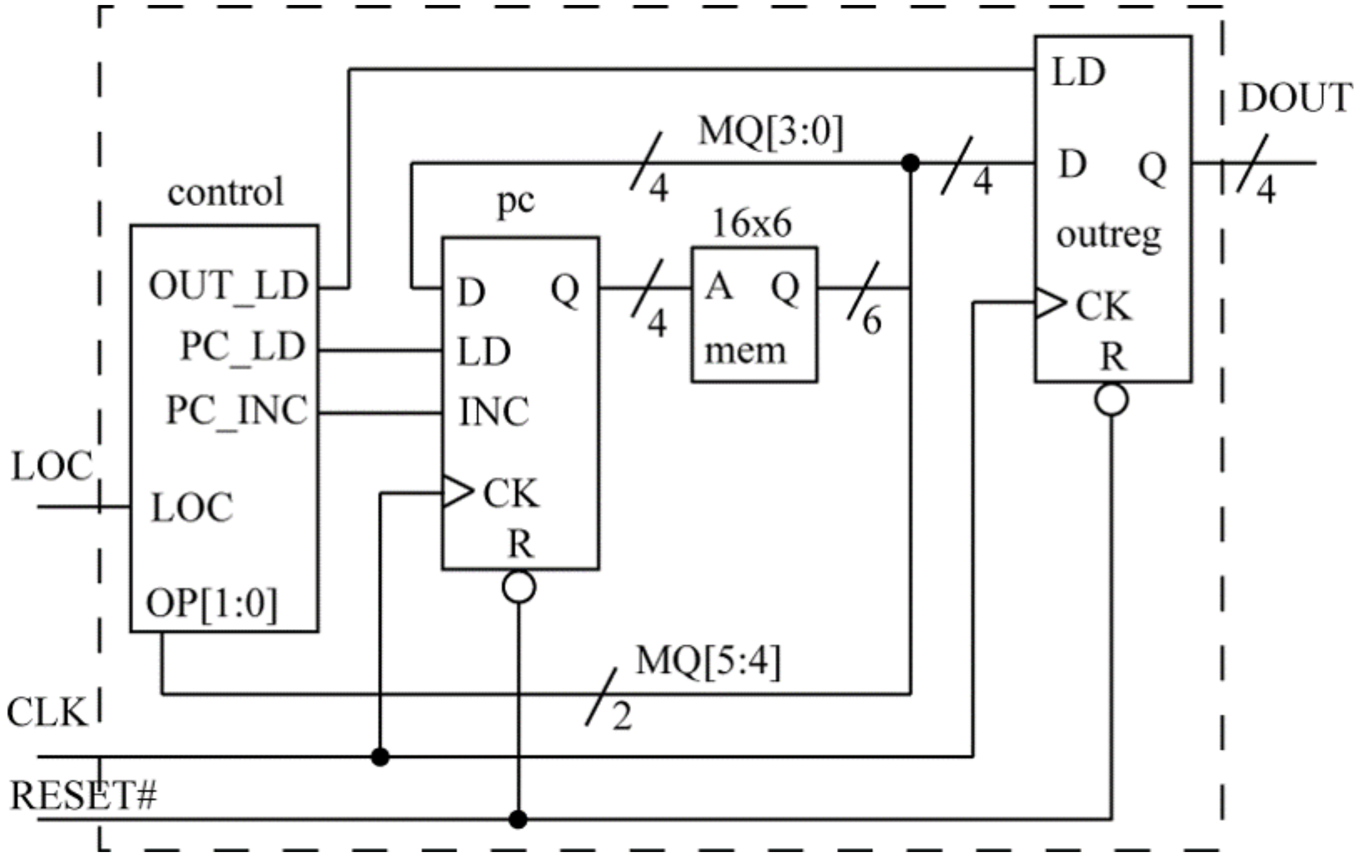
```
PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
      =  0    |   ~1    & 1      & ~0
      = 0

PC_INC = !PC_LD
       = 1

OUT_LD = ~OP[0] & ~OP[1]
       =   ~0   & ~1
       = 0
```

Therefore, the new address is 1 since `PC_INC` is true and the new `DOUT` is 0b1110 since `OUT_LD` is false.

NSC Example (2)



- PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
- PC_INC = !PC_LD
- OUT_LD = ~OP[0] & ~OP[1]

Address	Machine code
0	101000
1	100101
2	000001
3	010000
4	011001
5	000101

- If the current address is 1, LOC = 0, and DOUT = 0b0111, what is the value of DOUT and the next address to execute after running this instruction?

A. DOUT =
B. Next address =

Evaluating the output of the control logic where `OP[1:0]` = 0b10 and `LOC` = 0 gives:

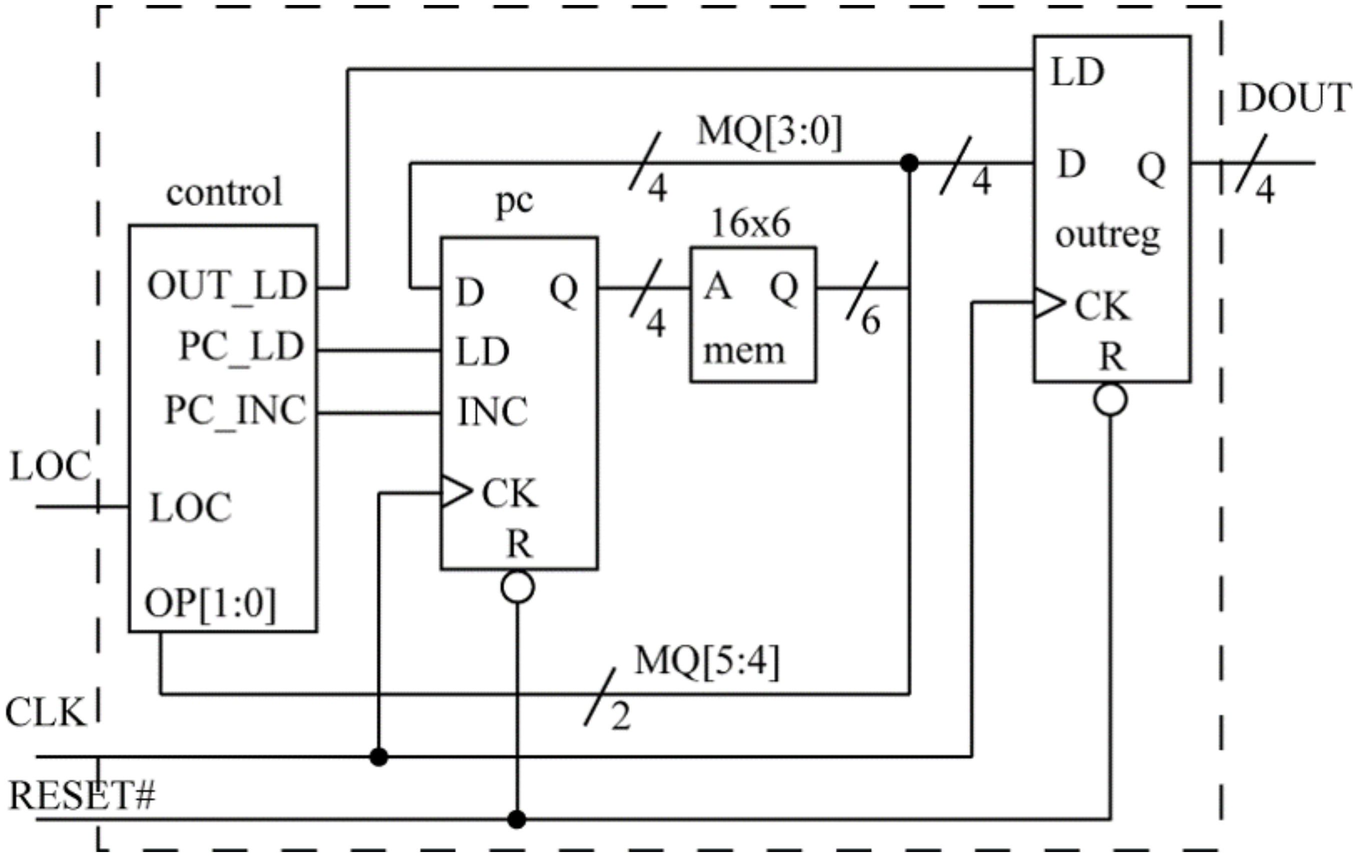
```
PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
      =  0    |   ~0  &  1    & ~0
      = 1

PC_INC = !PC_LD
       = 0

OUT_LD = ~OP[0] & ~OP[1]
       =    ~0  & ~1
       = 0
```

Therefore, the new address is 0b0101 (the lower four bits of the instruction) since `PC_LD` is true. The `DOUT` value is still 0b0111, because `OUT_LD` is false.

NSC Example (3)



- PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
- PC_INC = !PC_LD
- OUT_LD = ~OP[0] & ~OP[1]

Address	Machine code
0	101000
1	100101
2	000001
3	010000
4	011001
5	000101

- **If the current address is 2, LOC = 1, and DOUT = 0b1100, what is the value of DOUT and the next address to execute after running this instruction?**

A. DOUT =

B. Next address =

Evaluating the output of the control logic where `OP[1:0]` = 0b00 and `LOC` = 1 gives:

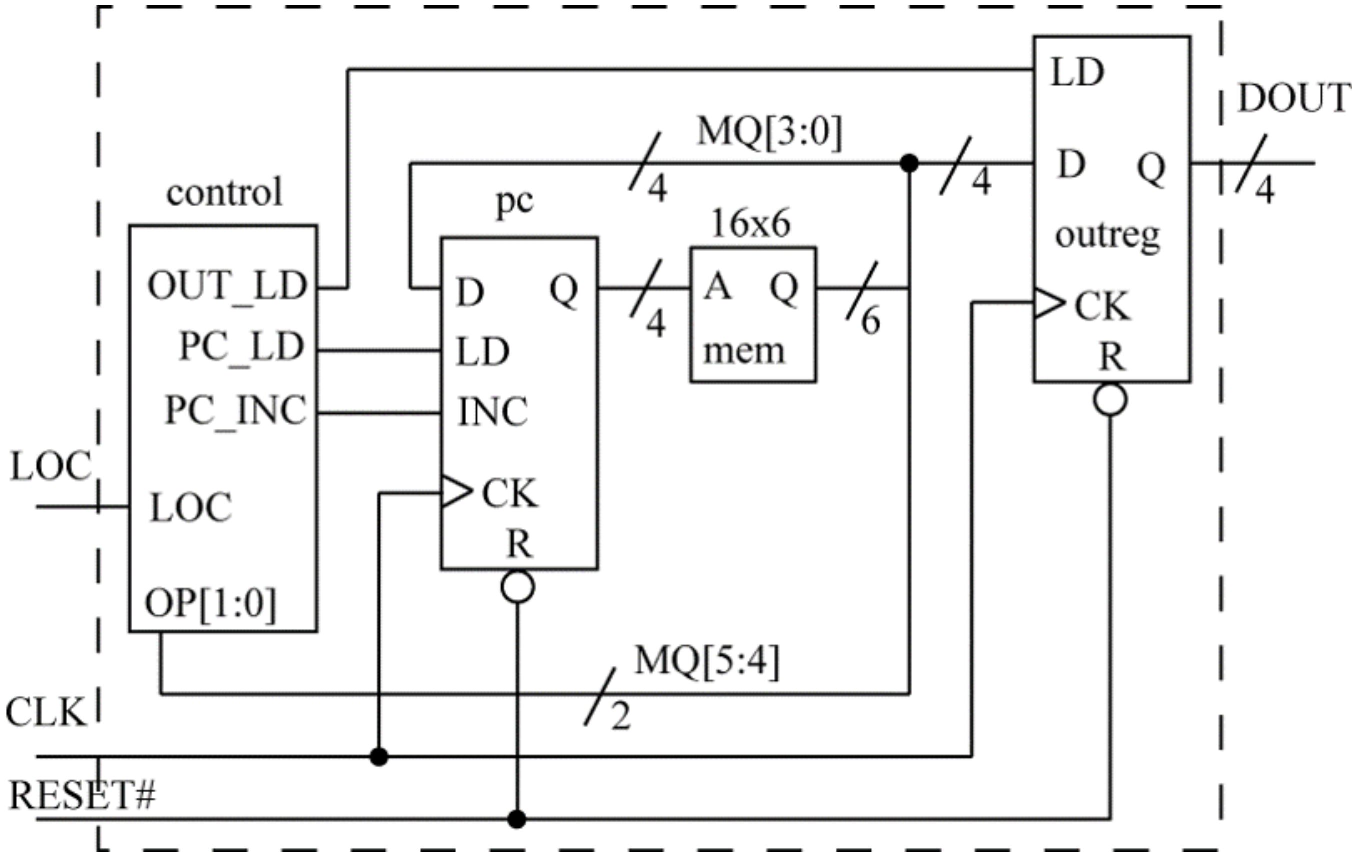
```
PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
      =  0    |   ~1  & 0      & ~0
      = 0

PC_INC = !PC_LD
       = 1

OUT_LD = ~OP[0] & ~OP[1]
       =   ~0   & ~0
       = 1
```

Therefore, the new address is 3 since `PC_INC` is true. The `DOUT` value is 0b0001 (the lower four bits of the instruction) because `OUT_LD` is true.

NSC Example (4)



- PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
- PC_INC = !PC_LD
- OUT_LD = ~OP[0] & ~OP[1]

Address	Machine code
0	101000
1	100101
2	000001
3	010000
4	011001
5	000101

- If the current address is 3, LOC = 1, and DOUT = 0b1001, what is the value of DOUT and the next address to execute after running this instruction?

A. DOUT =
B. Next address =

Evaluating the output of the control logic where `OP[1:0]` = 0b01 and LOC = 1 gives:

```
PC_LD = OP[0] | (~LOC & OP[1] & ~OP[0])
      =  1    |     ~1  & 0     & ~1
      = 1

PC_INC = !PC_LD
       = 0

OUT_LD = ~OP[0] & ~OP[1]
       =    ~1  & ~0
       = 0
```

Therefore, the new address is 0 (the lower four bits of the instruction) since `PC_LD` is true. The `DOUT` value is still 0b1001, because `OUT_LD` is false.