Approaches to Digital System Design

- In Digital Devices, you learned how to create a logic network (Flip-flops + combinational gates) to solve a problem

  - The logic network was SPECIFIC to the problem. To solve a different problem, needed a different logic network

  - A high end graphics card is an example of a special purpose processing system.

- Another approach is to design a logic network that can used to solve many different problems

  - This **general purpose** logic network might not be as efficient (speed, cost) as a special purpose logic network, but hopefully can be used to solve multiple problems!

A Computer!!

- A Computer is a digital system whose operation can be specified via a Program.

  – Changing the program changes the computer behavior! (solves a different problem!!!).

- A Program is simply a sequence of binary codes that represent instructions for the computer. The Program is stored in a Memory.

- External inputs to the Computer can also alter the behavior the computer. The computer will have Outputs that can be set/reset via program instructions.

  – These external inputs/output are known as the I/O section of the computer.

Components of any Computer System

- Control – logic that controls fetching/execution of instructions
- Memory – area where instructions/data are stored
- Input/Output – external interaction with computer
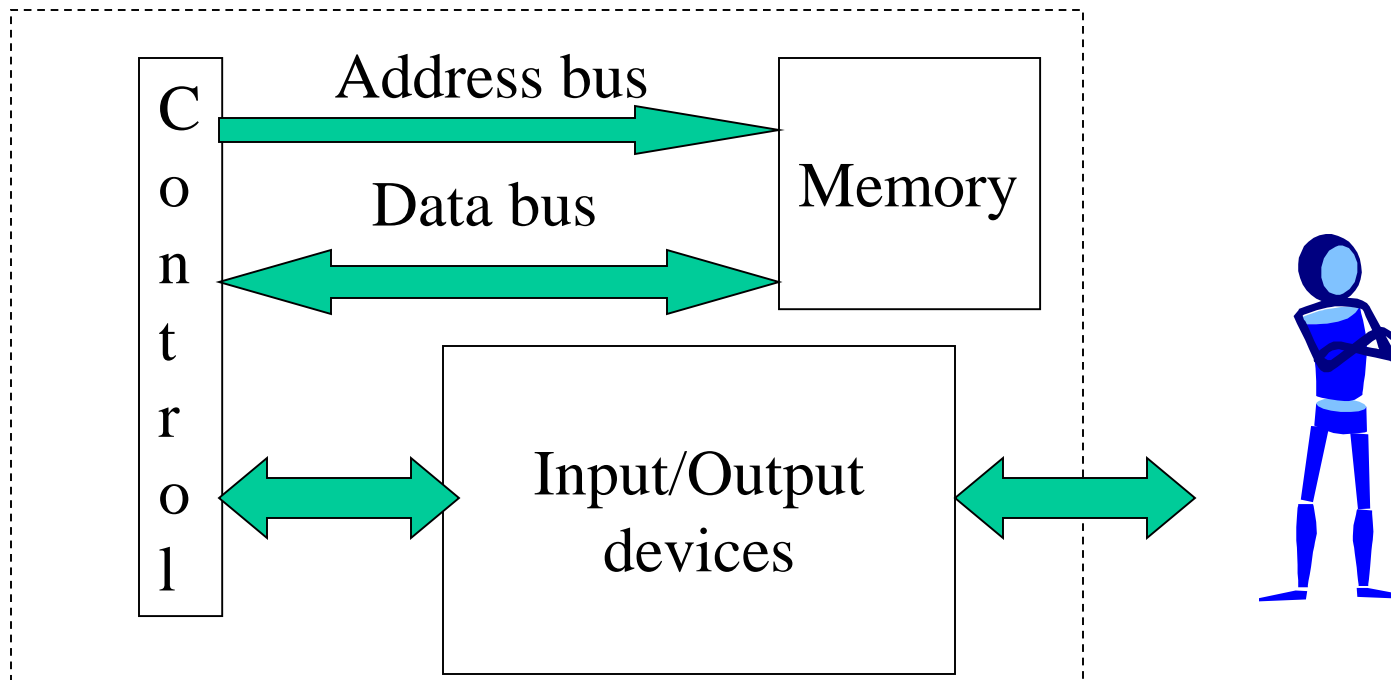
Problem Definition

Build a Digital System based upon your phone number, assumed to be of the form $Y_1 Y_2 Y_3$-$Z_1 Z_2 Z_3 Z_4$

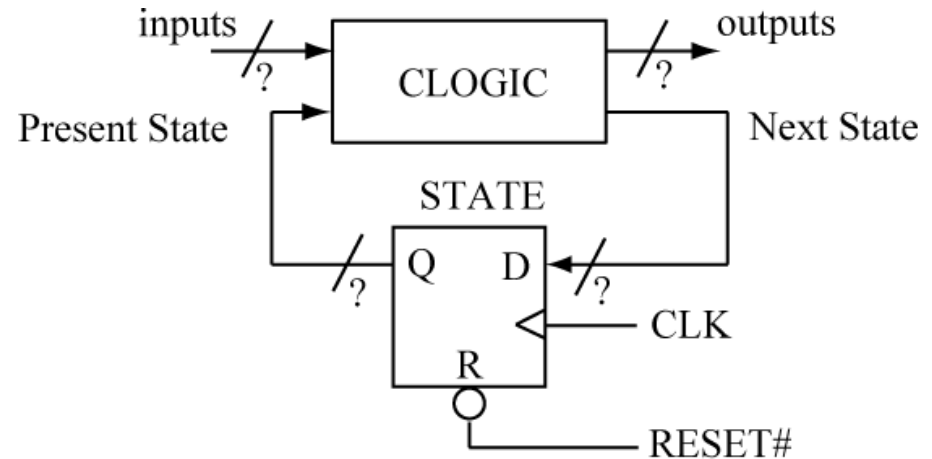The Digital System will have one external input called LOC.

If LOC is true, then the system will display only the digits $Z_1 Z_2 Z_3 Z_4$.

If LOC is false, then the system will display all seven digits.
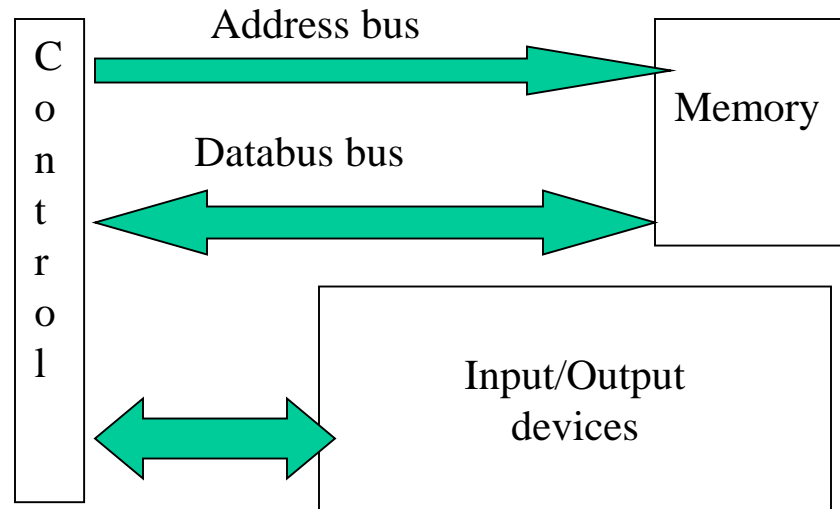
Two Approaches for Solving this Problem

Finite State Machine

Will only work for one number sequence

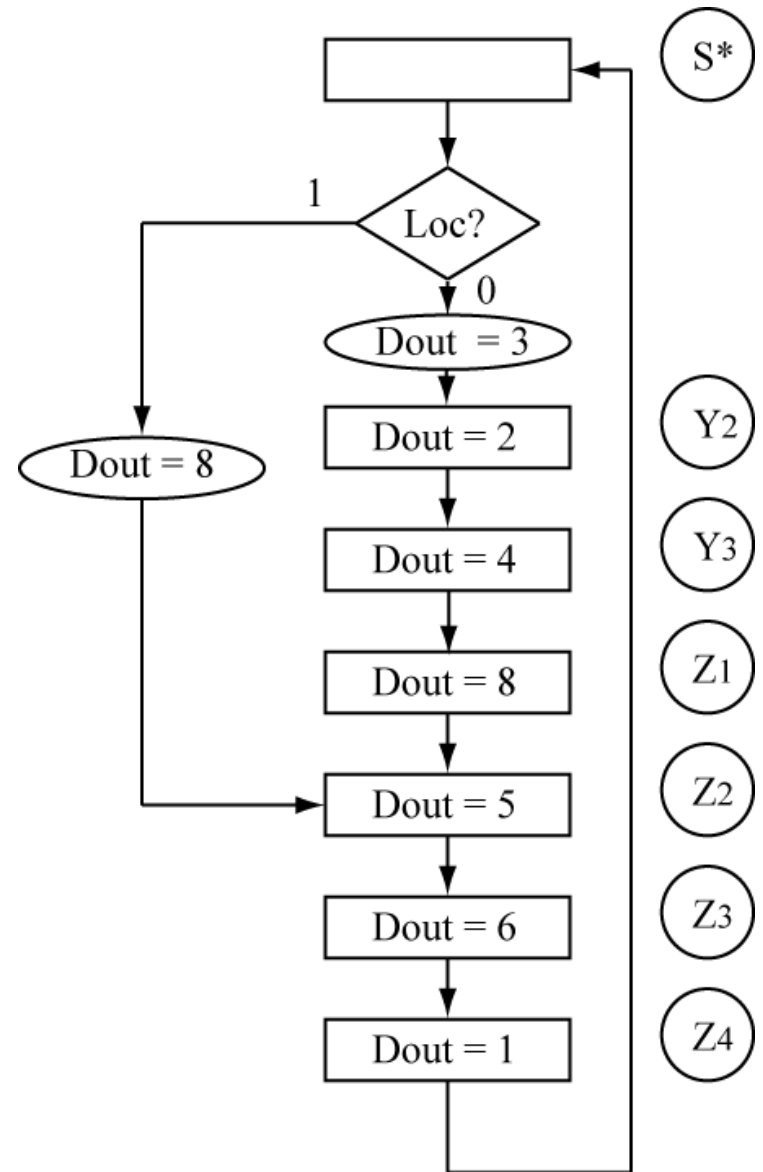inputs / ? → CLOGIC → / ? outputs

Present State

Next State

STATE

Q   D   ← / ?

CLK

R

RESET#

Computer System

Will only work for any number sequence, change program to change sequence

Control

Address bus

Memory

Databus bus

Input/Output devices

ASM chart for 324 8561

Odd only affects sequence in State S* (Reset state)

V 0.7

FSM Implementation
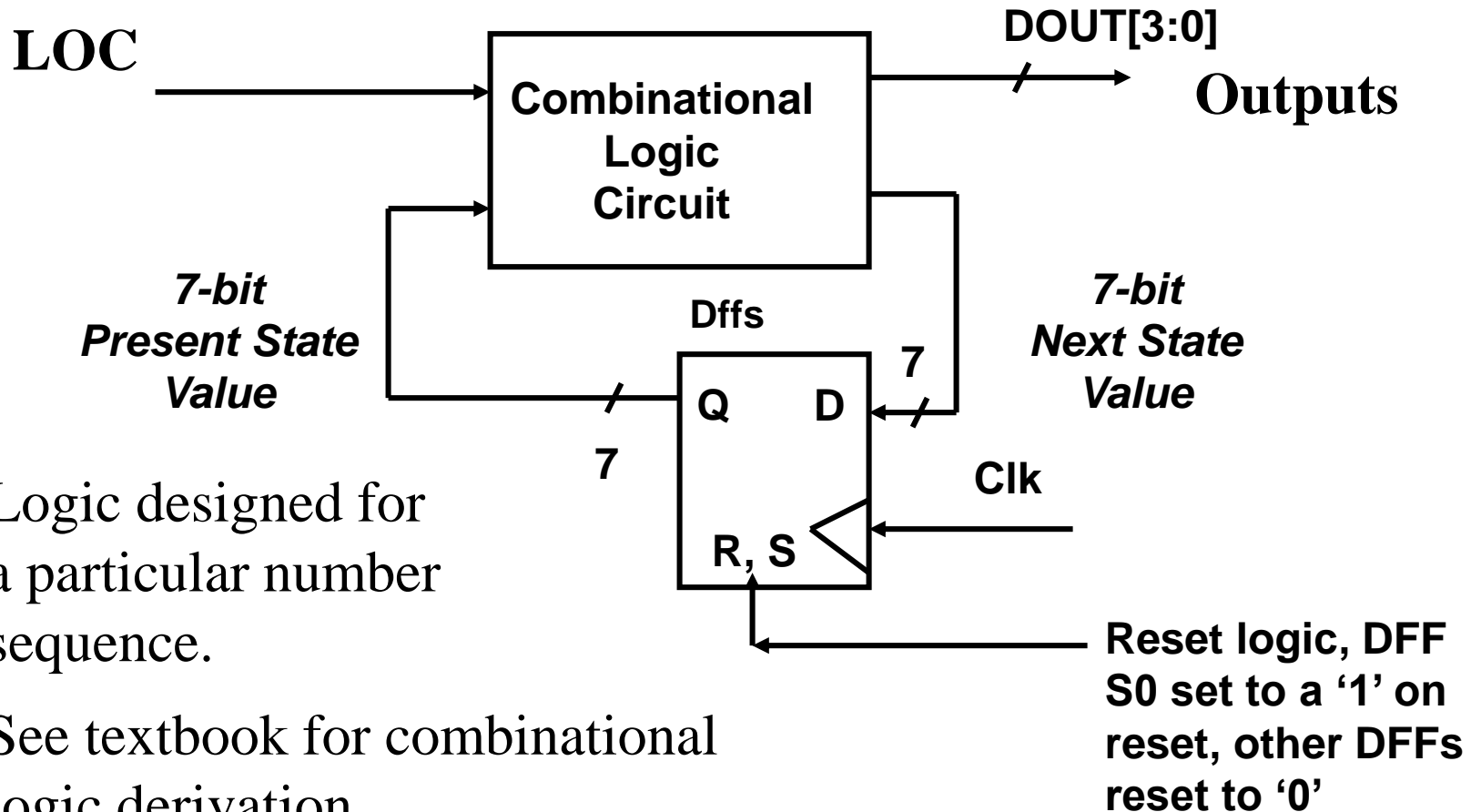
Table 2.1 Two Possibilities for State Assignment.

State	Binary Encoding	One-Hot Encoding
S^*	000	0000001
Y_2	001	0000010
Y_3	010	0000100
Z_1	011	0001000
Z_2	100	0010000
Z_3	101	0100000
Z_4	110	1000000

One-Hot encoding: one DFF per state, requires 1 DFF per state but simplifies combinational logic.
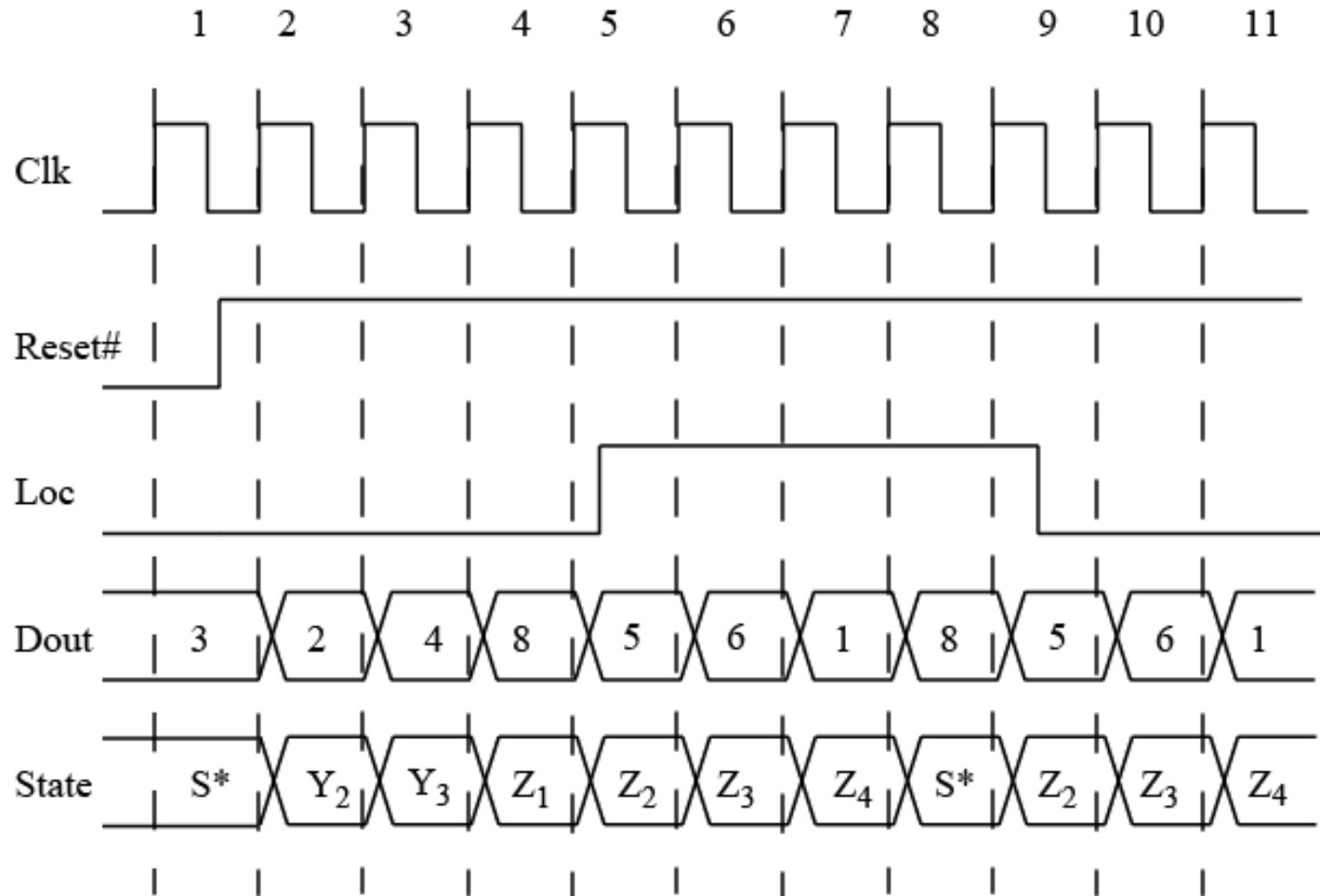
Binary Encoding: use minimal number of DFFs, but makes combinational logic more complex.

FSM Implementation (cont.)

Use one hot encoding, D-FFs for the 7 states.

LOC

Combinational Logic Circuit

DOUT[3:0]

Outputs

7-bit Present State Value

Dffs

7

7-bit Next State Value

7

Q　**D**

Clk

R, S

Logic designed for a particular number sequence.

Reset logic, DFF S0 set to a '1' on reset, other DFFs reset to '0'

See textbook for combinational logic derivation.

FSM Operation

Each state requires one clock cycle

V 0.7

9

Computer System Implementation
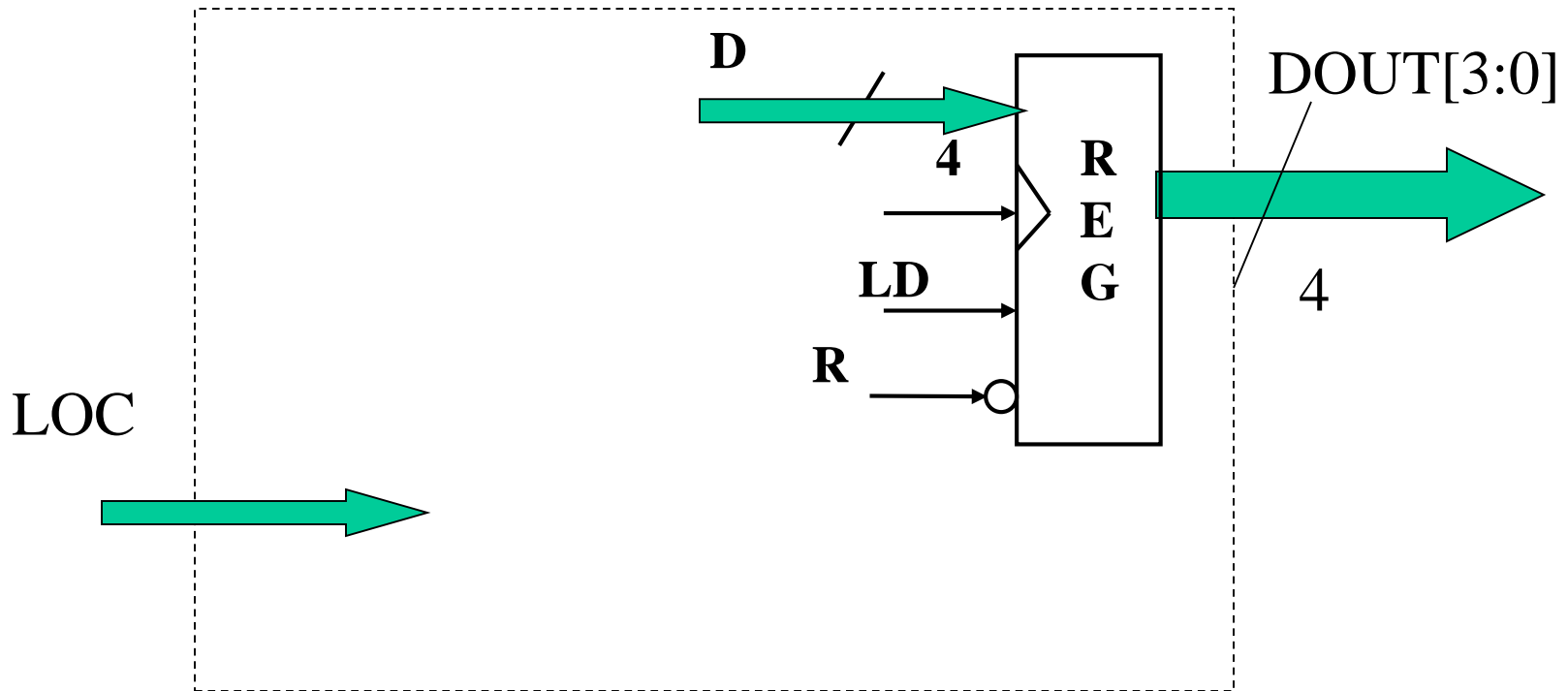# What do We Need?
Input/Output First, same as FSM

DOUT[3:0]

4

LOC

DOUT[3:0] - 4 bit output bus that has the value of the digit

LOC – 1 bit input that controls whether or not the full number sequence is displayed
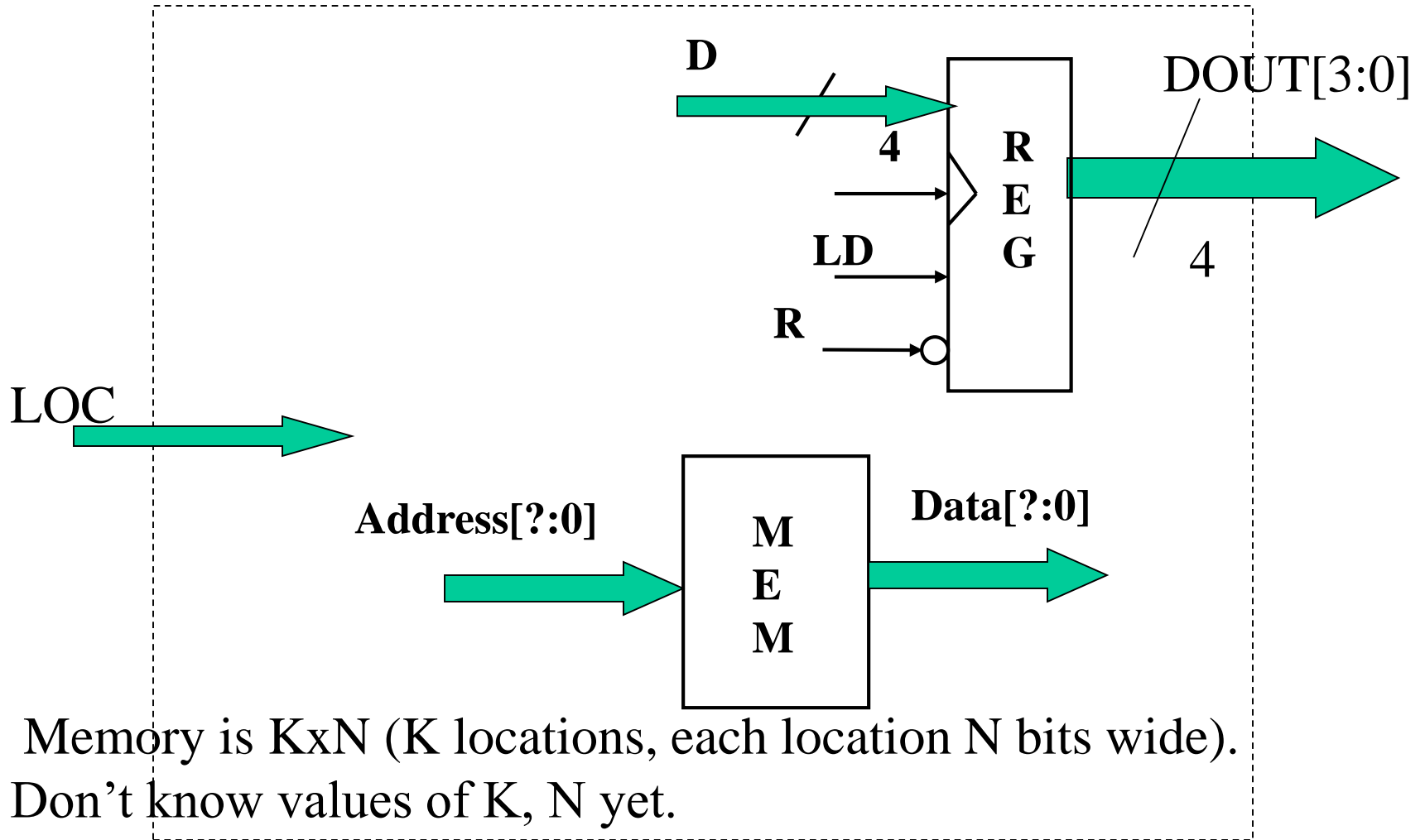
Register for holding DIGIT output value

D

DOUT[3:0]

4

R
E
G

4

LD

R

LOC

Register loads D on rising clock edge when LD = 1.

R is an asynchronous clear.

Memory for holding instructions

D

DOUT[3:0]

4

R E G

LD

4

R

LOC

Address[?:0]

Data[?:0]

M E M

Memory is KxN (K locations, each location N bits wide).
Don't know values of K, N yet.

What Instructions do we need?

START:
 If (loc) goto LOCAL;
  dout = 3;
  dout = 2;
  dout = 4;
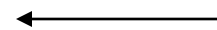 LOCAL:
  dout = 8;
  dout = 5;
  dout = 6;
  dout = 1;
  goto START;

⟵ Psuedo Code for operations

Needed Instructions

1. *Jc location*   Jump conditionally
    If LOC = 1, then jump to location (counter set equal to specified location).
    If LOC = 0, then fetch next instruction (counter increments by 1).

2.   *Jmp location*   Jump unconditionally
    Fetch next instruction from location (counter loaded with specified location).

3.   *out data*
        load output register with data. Used for setting the DOUT[3:0] value.

Instruction Encoding

The binary encoding for instructions is usually divided into different fields; with each field representing part of the information needed by the instruction.

Our instructions require two fields: *Operation Code* and *Data*

| Opcode | Data |

How many bits for the Opcode? Have 3 instructions, need at least 2 bits! (2 bits can encode 2^2 items)

How many bits for Data? The data field must specify the 4 bits for the DOUT number, and also specify a memory location. For now, lets use 4 bits for data. Instruction is 6 bits total.

I5 I4   I3 I2 I1 I0

| Opcode | Data |

Instruction Table

	I5 I4		I3 I2 I1 I0
JMP location	0 0	|	4-bit location
JC location	0 1	|	4-bit location
OUT data	1 0	|	4-bit data

means memory will have maximum of $2^4 = 16$ locations.

Each location will contain 6 bits.

Note that Opcode = 11 is unused.

The opcode assignment was chosen so that the OUT instruction could be distinguished from the two jump instructions by only the most significant bit of the opcode.

Could have chosen another opcode assignment, but this could make the decode logic more complex.

A Program for 324 8561
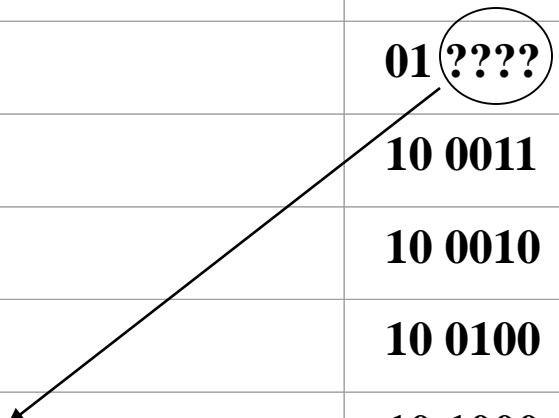
```
start:          JC      local       ; jump only if LOC input=1
                OUT  3
                OUT 2
                OUT 4
local:          OUT  8
                OUT  5
                OUT  6
                OUT  1
                JMP   start
```

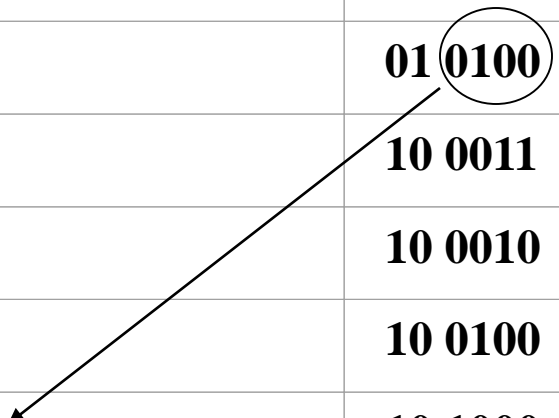A program written using the native instructions of the computer is called an ***Assembly Language Program.***

Convert Program to Binary, Put in Memory

Memory Location	Machine Code	Instruction
0x0	01 ????	`START: JC LOCAL`
0x1	10 0011	`OUT 3`
0x2	10 0010	`OUT 2`
0x3	10 0100	`OUT 4`
0x4	10 1000	`LOCAL: OUT 8`
0x5	10 0101	`OUT 5`
0x6	10 0110	`OUT 6`
0x7	10 0001	`OUT 1`
0x8	00 0000	`JMP START`

Convert Program to Binary, Put in Memory (final)

| Memory Location	Machine Code	Instruction
0x0	01 0100	`START: JC LOCAL`
0x1	10 0011	`OUT 3`
0x2	10 0010	`OUT 2`
0x3	10 0100	`OUT 4`
0x4	10 1000	`LOCAL: OUT 8`
0x5	10 0101	`OUT 5`
0x6	10 0110	`OUT 6`
0x7	10 0001	`OUT 1`
0x8	00 0000	`JMP START`

Register for specifying address – Use counter
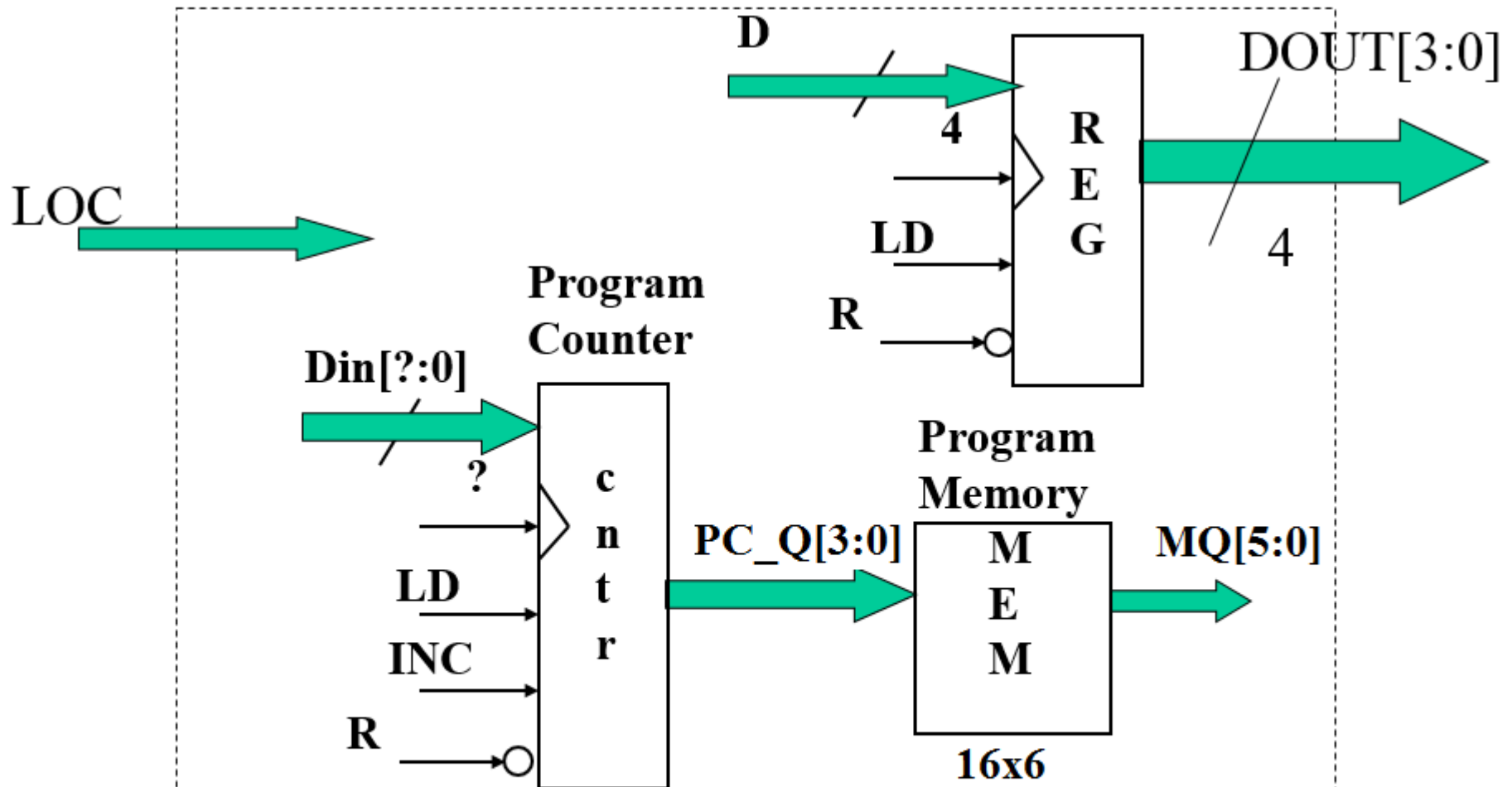
Counter increments on rising clock edge when INC = 1. Loads on rising clock edge when LD = 1. R is an asynchronous clear.

Add *control* Logic to Execute Instructions

16 locations by 6 bits wide.

Add *control* Logic to Execute Instructions

16 locations by 6 bits wide.
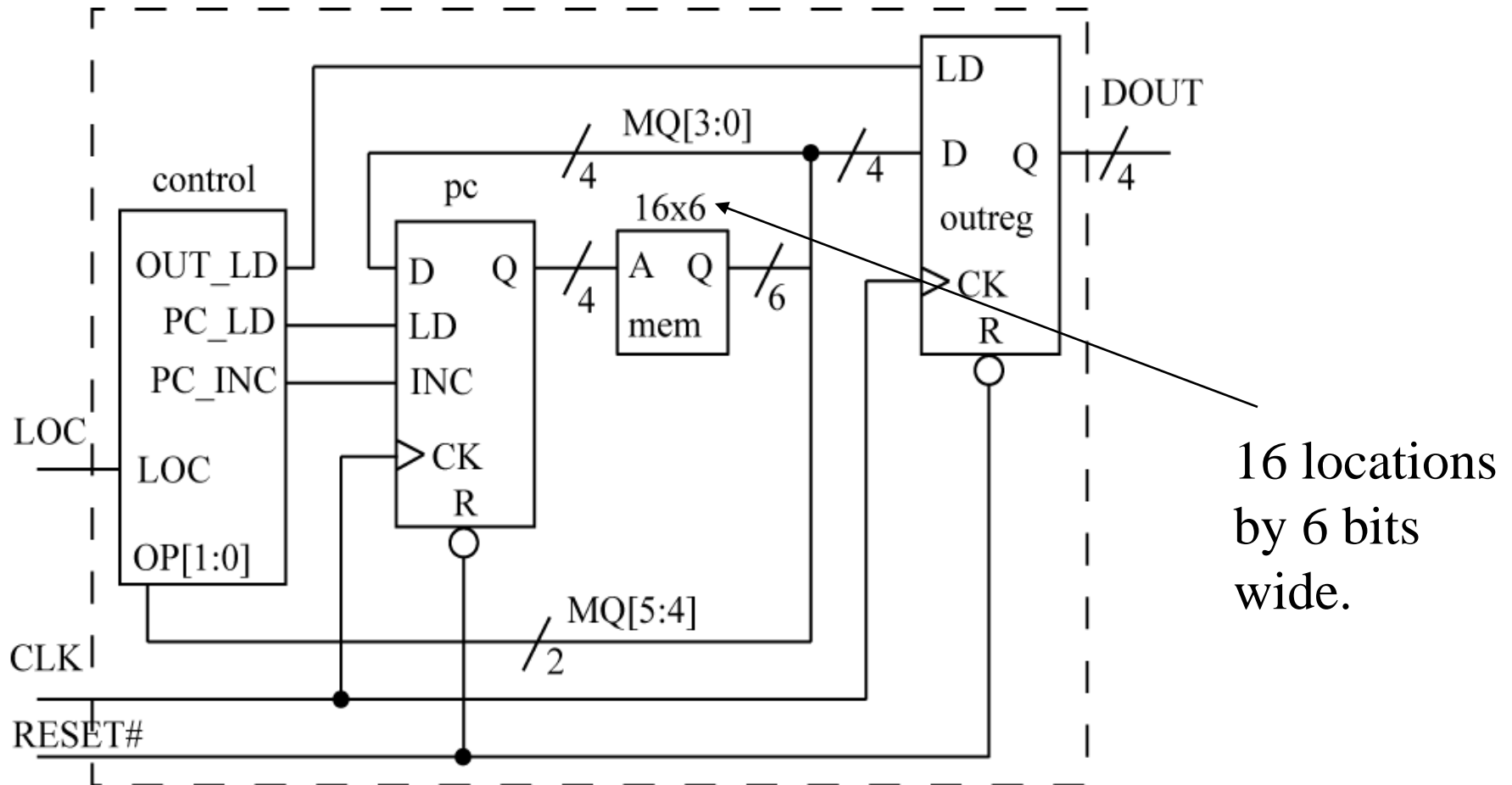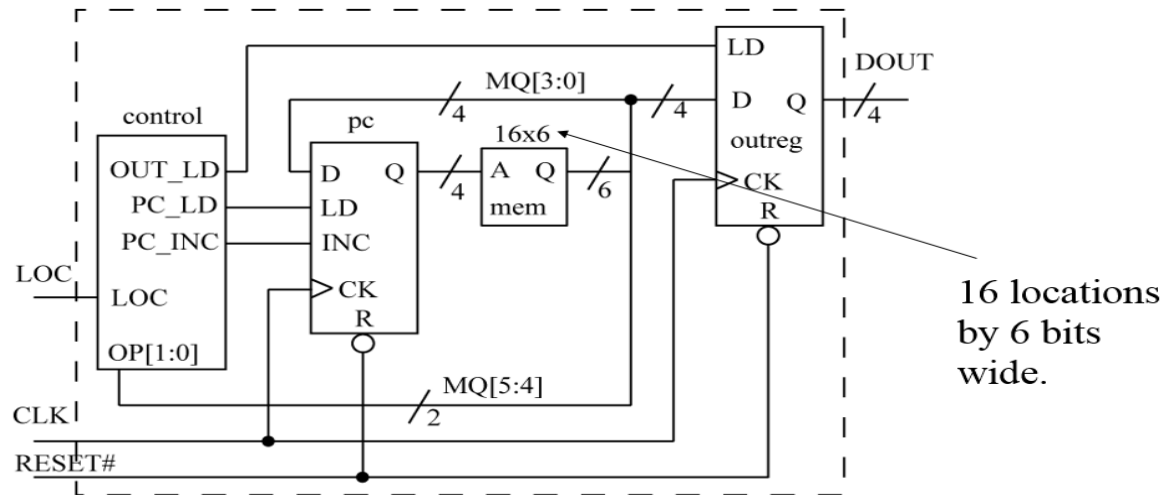
Write equations for control logic

Instruction	Opcode	PC Register	Output Register
JC	01	LD=LOC, INC=~LOC	LD=0
JMP	00	LD=1, INC=0	LD=0
OUT	1x	LD=0, INC=1	LD=1

What is *control* Logic?

Control logic controls *count* register, *out* register based on Op code value (op[1:0] = Data[5:4]).

When does *out* register get loaded?   When OP = 10!! (OUT instruction):
VHDL:

> ***out_ld   <=   '1' when   (op = "10") else '0';***

When does Counter Load? When JMP instruction (OP=00) or when JC instruction and LOC = '1'!!!!

> ***pc_ld <= '1' when (op="00" or (op = "01" and LOC='1'))***
> ***else '0';***

When does counter increment?   When NOT Loading!!

> ***pc_inc <= not (pc_ld);***

pc_ld, *pc_inc* are LD, INC inputs to counter.

out_ld is LD input to output register.

Decode Boolean Equations

out_ld *<=* ***op(1)*** -- don't really need op(0)
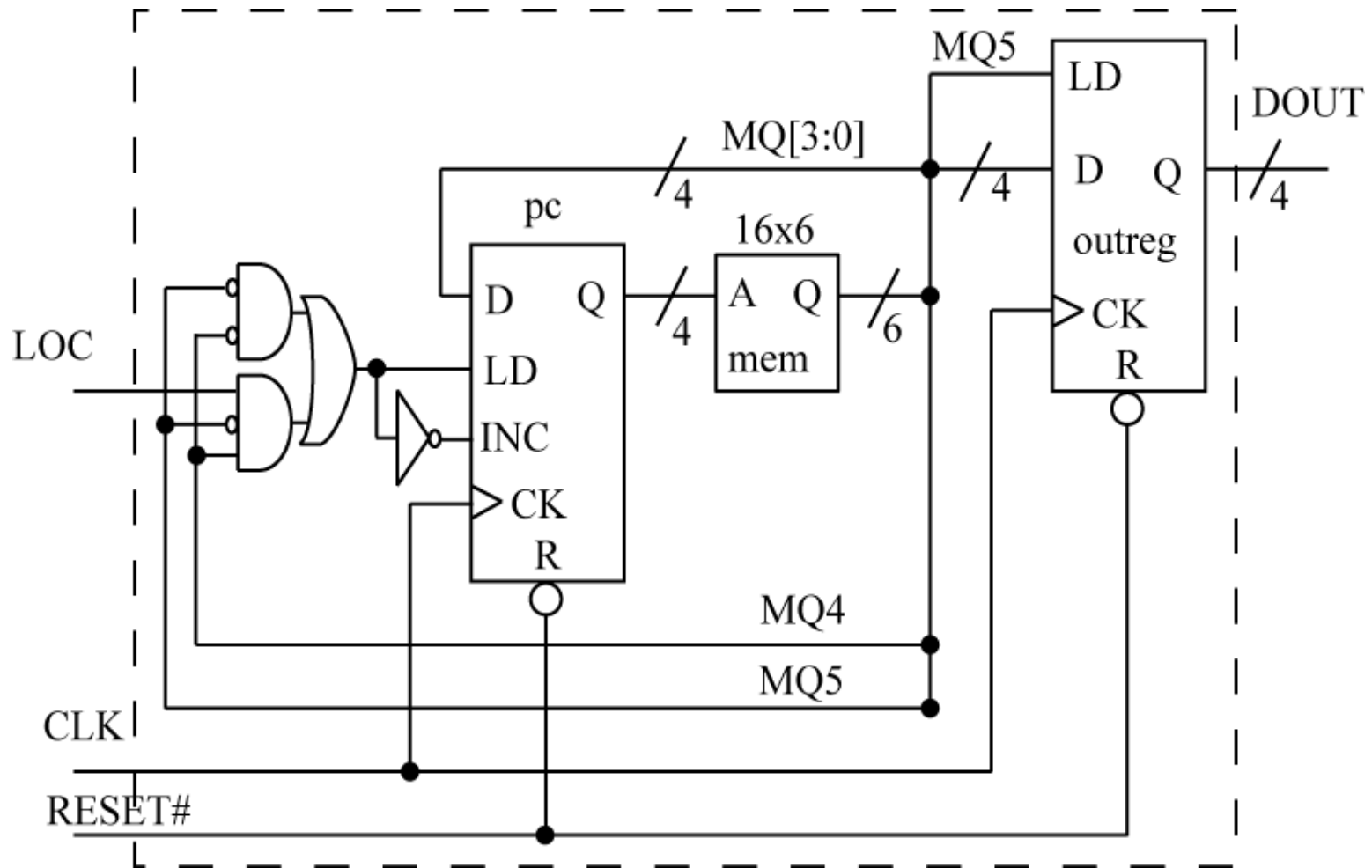
pc_ld *<= ((not op(1)) and (not op(0)) or*
               ((not op(1)) and op(0) and LOC));

pc_inc *<= not (pc_ld);*

pc_ld, *pc_inc* are LD, INC inputs to counter.

out_ld is LD input to output register.

Final Hardware

Timing

Location	Instruction
0	JC 4
1	OUT 3
2	OUT 2
3	OUT 4
4	OUT 8
5	OUT 5
6	OUT 6
7	OUT 1
8	JMP 0
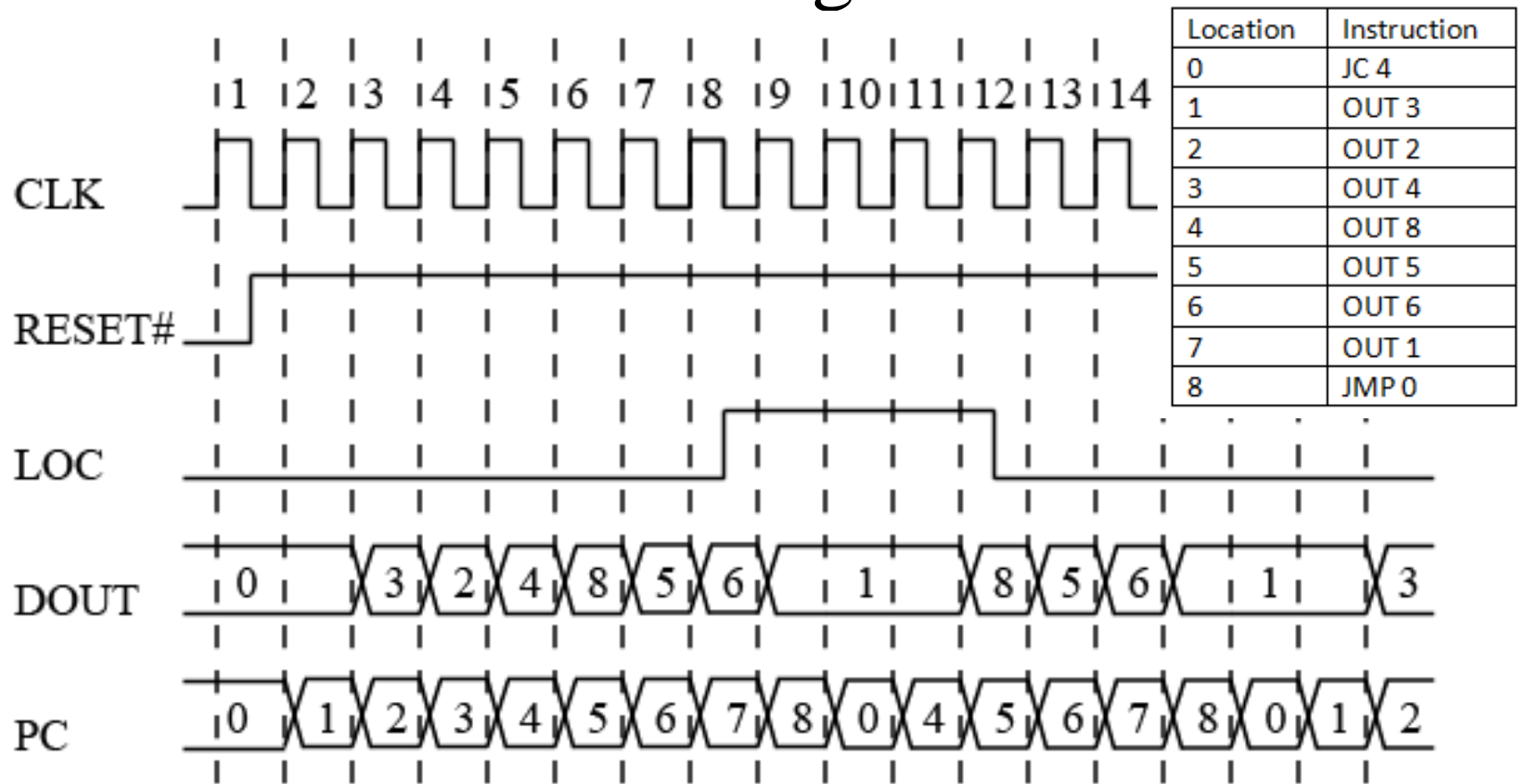
Observe that DOUT value does not change each clock cycle as with FSM implementation. This is because of the extra clock cycles needed by the JC, JMP instructions.

32

V 0.7

Comments

- Notice that the *RESET#* line forces the processor to fetch its **first** instruction from location 0.

  – All processors have a RESET# line like this to force the first instruction fetch from a particular location.

- Notice that execution never stops!!! Processor is always fetching, executing instructions!

- Called the *Fetch,Execute* loop.

- Must make sure that memory is loaded with valid instructions BEFORE execution starts!!!

Program Counter

- The counter in this processor is a special purpose register that exists in one form or another in *every* processor

- Usually is called the *Instruction Pointer* (IP) register or *Program Counter* (PC) register.

- This register contains the address of the next instruction to be fetched.
  - Normal operation is to fetch very next instruction in memory
  - Jump instructions change the PC value so that fetch occurs from some non-sequential memory location

Implementation Comparisons

- FSM Implementation
  - Only 7 DFFs + combinational logic
  - Will only do one number sequence
  - Will operate a faster clock rate than Processor implementation because of simpler logic

- Processor Implementation
  - Many more gates needed than FSM implementation
  - Will execute at a slower clock rate than FSM
  - General purpose: can implement any number sequence by simply changing program.

- MANY applications are better suited for implementation by general purpose digital systems (Processors) than by dedicated logic

Vocabulary

- *Address bus* – input bus to memory device specifying location of data to read/write

- *Data bus* – input/output bus to memory device containing data value being read or written.

- *Program Counter* – special register in a processor specifying address of next instruction to be executed.

- *Instruction Mnemonic* – the ASCII representation of an instruction (i.e., OUT 4).

- *Machine Code* – the binary representation of an instruction (i.e. OUT 4 = 10 0100)

Vocabulary (cont.)

- *Operation code (Op code)* – the part of the machine code for an instruction that tells what the instruction is (JMP = 00).

- *Assembly* – the process of converting instructions to their machine code representation

    OUT 4 $\rightarrow$ 10 0100

- *Disassembly* – the process of converting machine code to its instruction mnemonic

    10 0100 $\rightarrow$ OUT 4

- *Fetch/Execute* - what processors do all day long (fetch instruction from memory, execute it).

How are modern Computers different from Number Sequencing Computer?

- NSC processor had 4-bit registers. Com. processors have registers with widths from 8 bits to 128 bits wide.

- NSC processor has 2 registers. Com. proc have many registers, some general purpose, some special purpose.

- NSC processor has 3 instructions. Com. Proc have 10's to a few hundred instructions (arithmetic, logical, control, Input/output, data movement,etc).

- NSC processor could address 16 memory locations. Com. Proc can address billions of memory locations.

- NSC processor can be implemented in a few 10's of gates. Com. Processors can take millions of gates to implement.

What do you need to know?

- Differences between specific logic networks and general purpose logic networks for digital systems.
- Basics of a computer system
- Logic Structure, timing of our NSC sequence processor
- Instruction assembly,disassembly, execution of NSC sequence processor
- Vocabulary