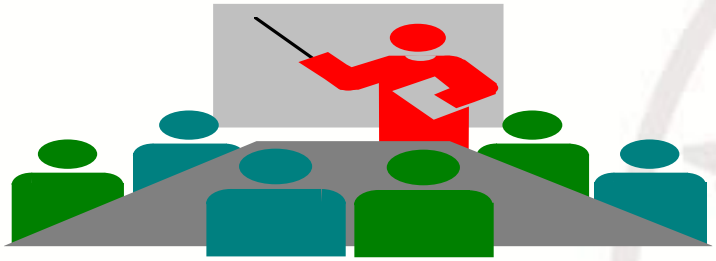


数字逻辑设计

LOGIC and Computer Design Fundamentals

CHAPTER 6

Register & Register Transfers part III

--Control of Register Transfers

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Overview

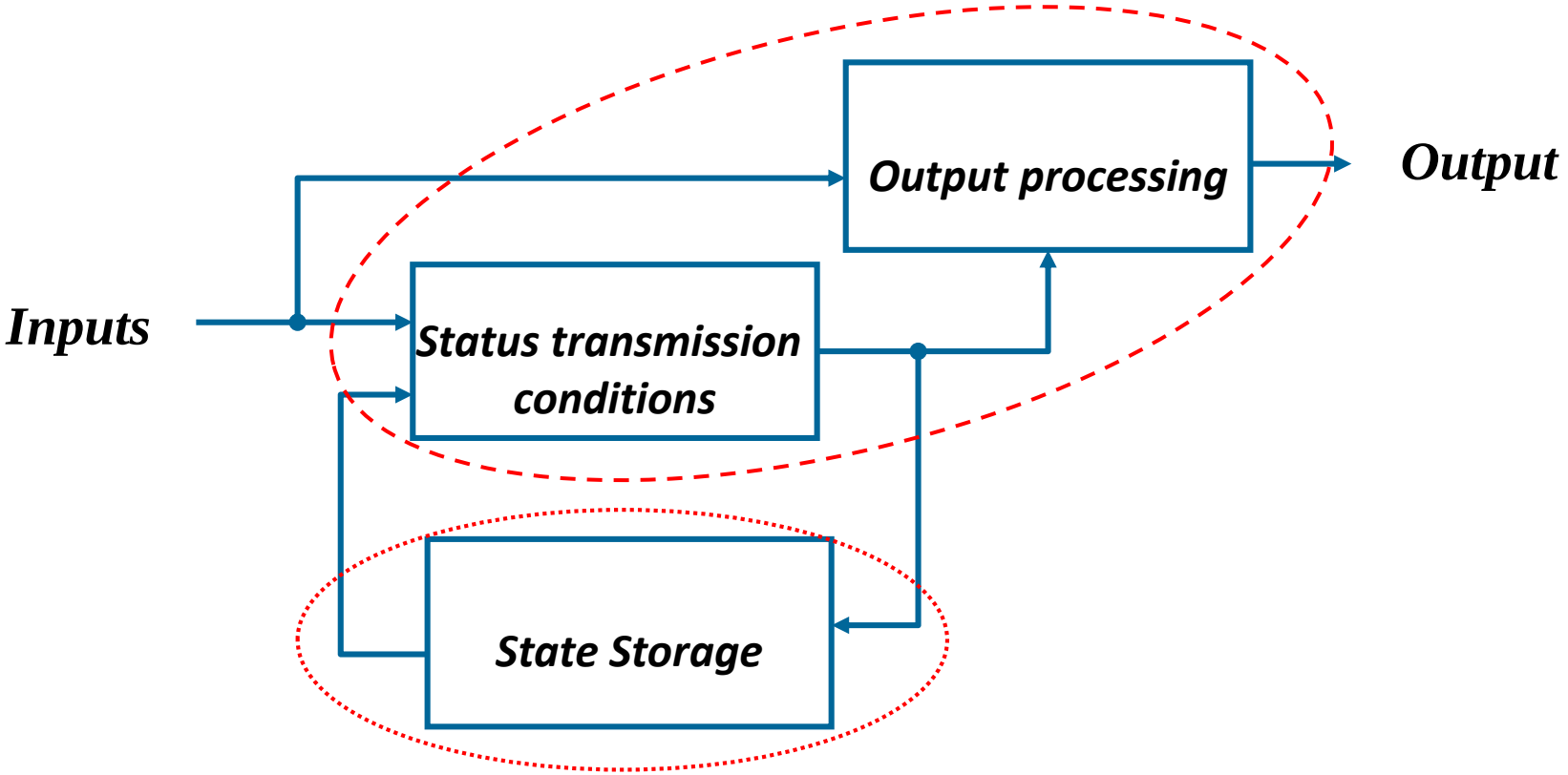
- Part 1 – Registers, Microoperations and Implementations
- Part 2 – Counters, Register Cells, Buses, & Serial Operations
- **Part 3 – Control of Register Transfers**
  - Introduction to register transfer systems
  - Register transfer system design procedure
  - A design example
  - Microprogrammed control

Course Outline

- Problems in state machine design
- Control of Register Transfer
- From Turing machine to CPU
- An example with RTL

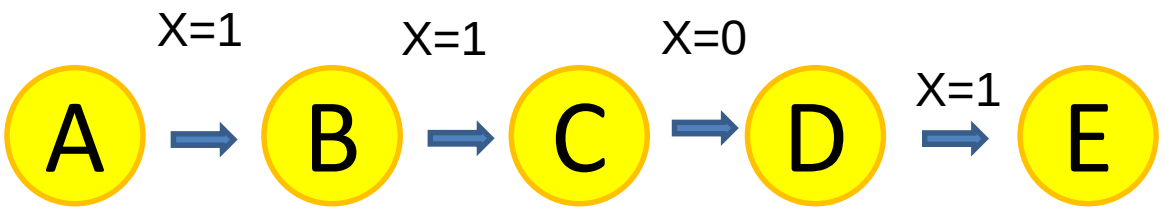
Introduction:
Digital System Design Method 1

- **Finite State Machine(FSM)**



Detection successive occurrence "1101" number of times

◎**How use to FSM ?**



■ ■ ■

?

Uncertain state machine

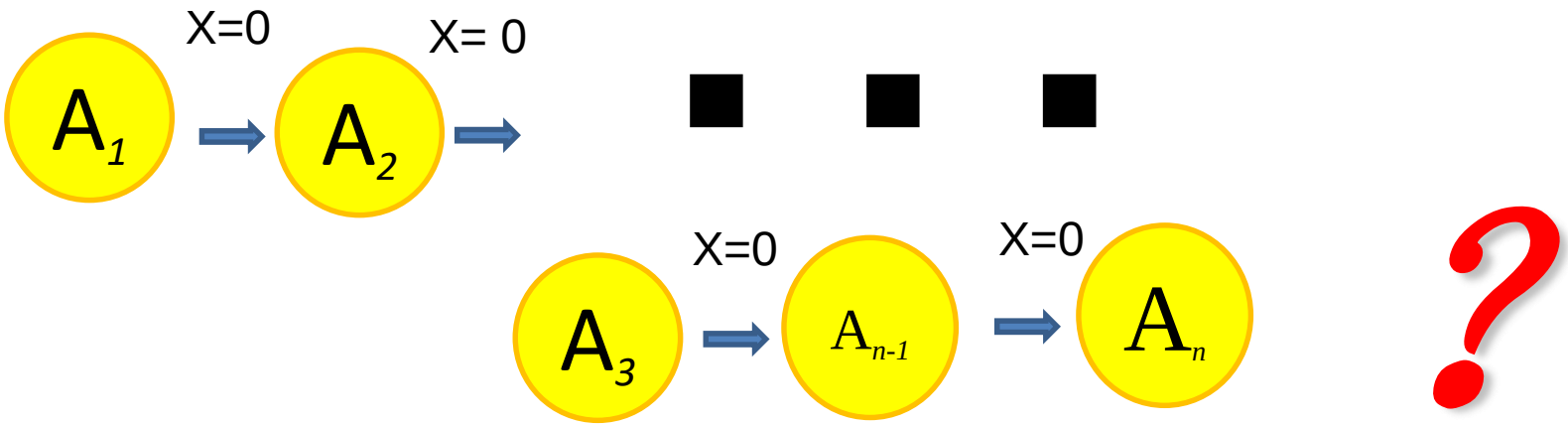
Uncertain state machine

Detection consecutive "0" number

Infinite state machine

Infinite state machine

Detection consecutive "0" number



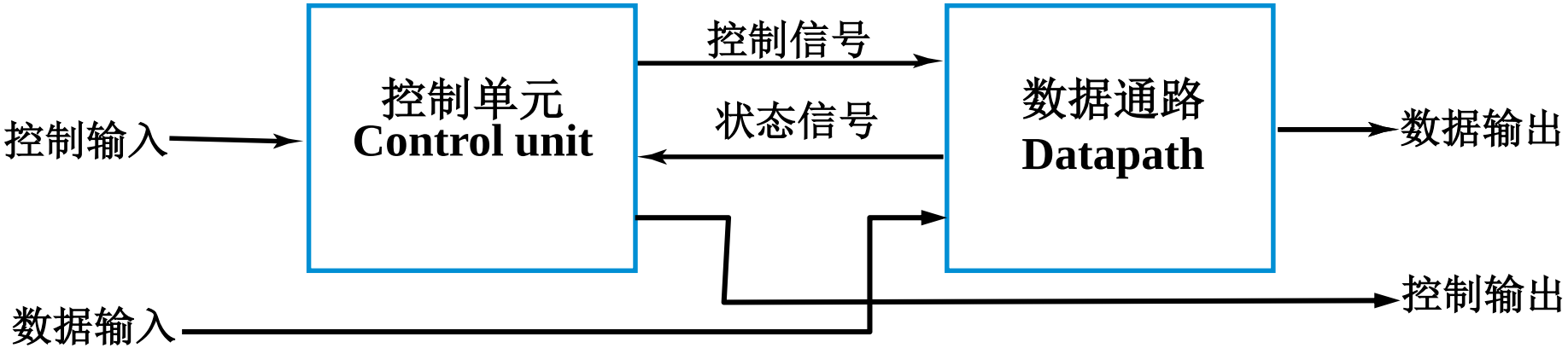
To Isolate uncertainties

Course Outline

- Problems in state machine design
- Control of Register Transfer
- From Turing machine to CPU
- An example with RTL

Digital System Design Method 2

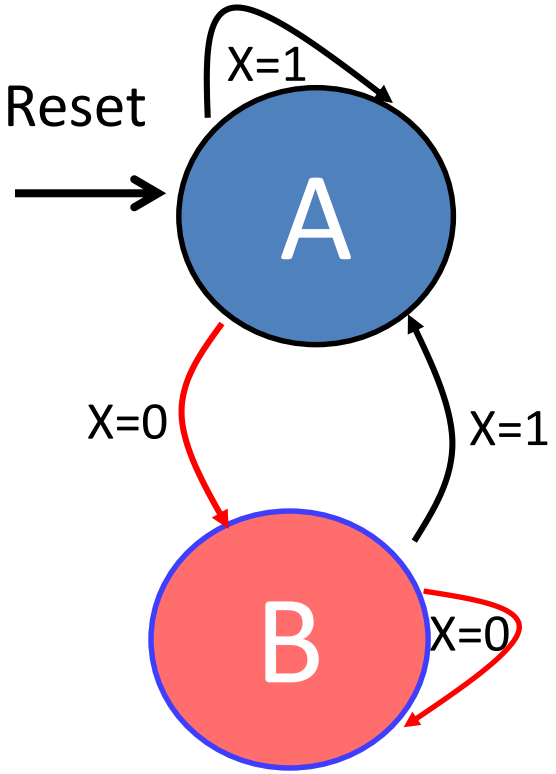
- **Control of Register Transfers （传输状态机：RSM）**
  - Register transfers performed on registers Control that supervises the sequencing of the register transfers
- **Three essential elements**
  - Set of registers: mostly in Datapath with some in Control Unit
  - Basic operation (micromanipulation): Register transfers performed
  - Control: that supervises the sequencing of the register transfers



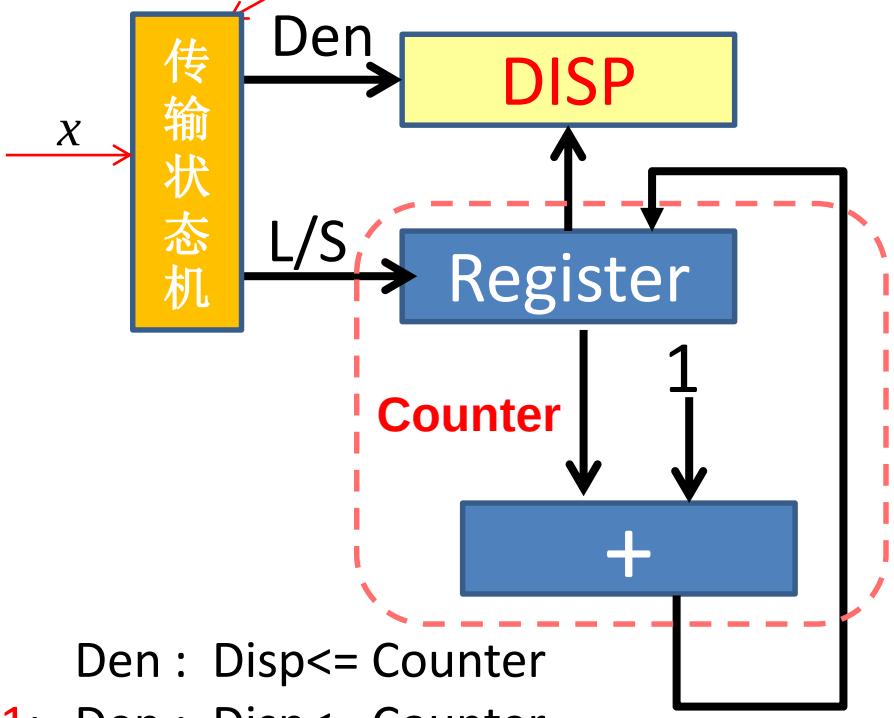
- **Register Transfer Language: *RTL***

Transmission state machine processing

Detection consecutive "0" number



根据问题设计传输状态机



Reset: L/S:Counter<=0; Den : Disp<= Counter

X=0: L/S:Counter<= Counter+1; Den : Disp<= Counter

X=1: $\overline{\text{L/S}}$:Counter<= Counter; $\overline{\text{Den}}$: Disp<= Disp

Specific and general system

◎ Non-programmable System: specific System

- the control unit does not deal with fetching and executing instructions

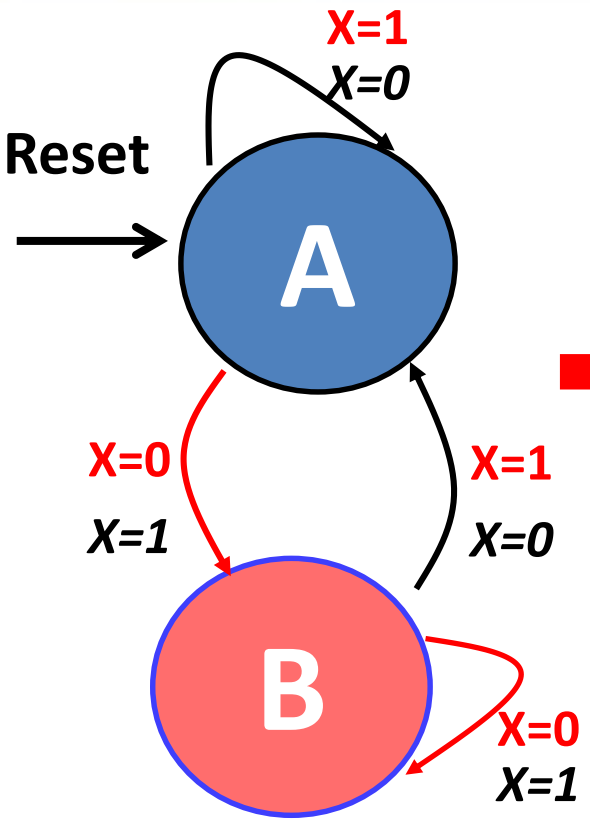
Hardware Programmable

- but contains all of the information for sequencing register transfers based on inputs and on status bits from the datapath

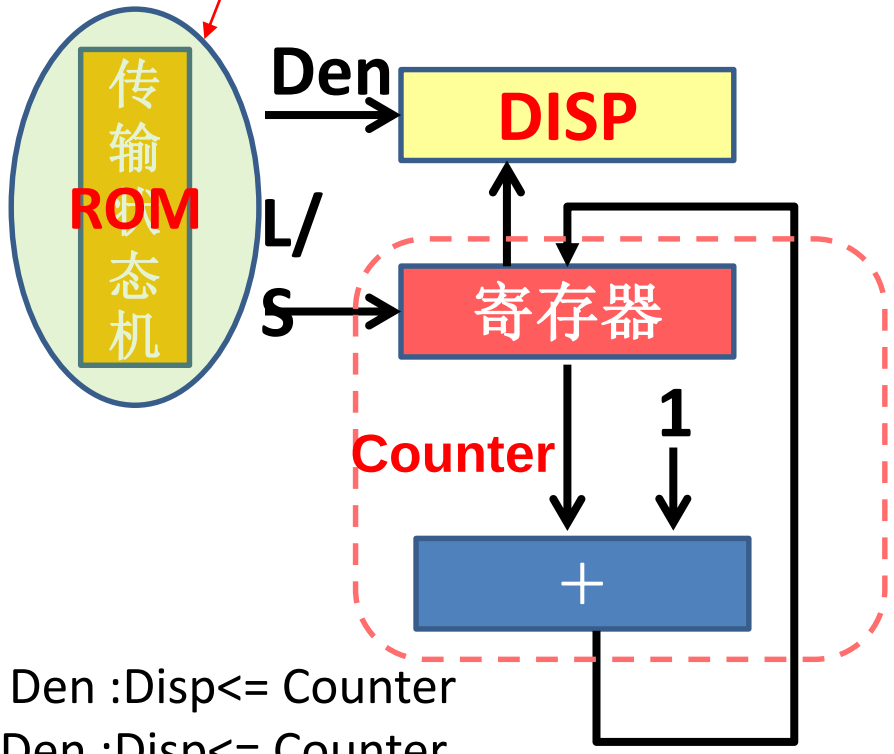
◎ Programmable System : General system

- a portion of the input consists of a sequence of *instructions* called a *program*,
- typically stored in a memory and addressed by a *program counter*.
- The Control Unit is responsible for fetching and executing these instructions.

Hardware Programmable：
Detection consecutive “0/1" number



问题改变，控制信号序列改变

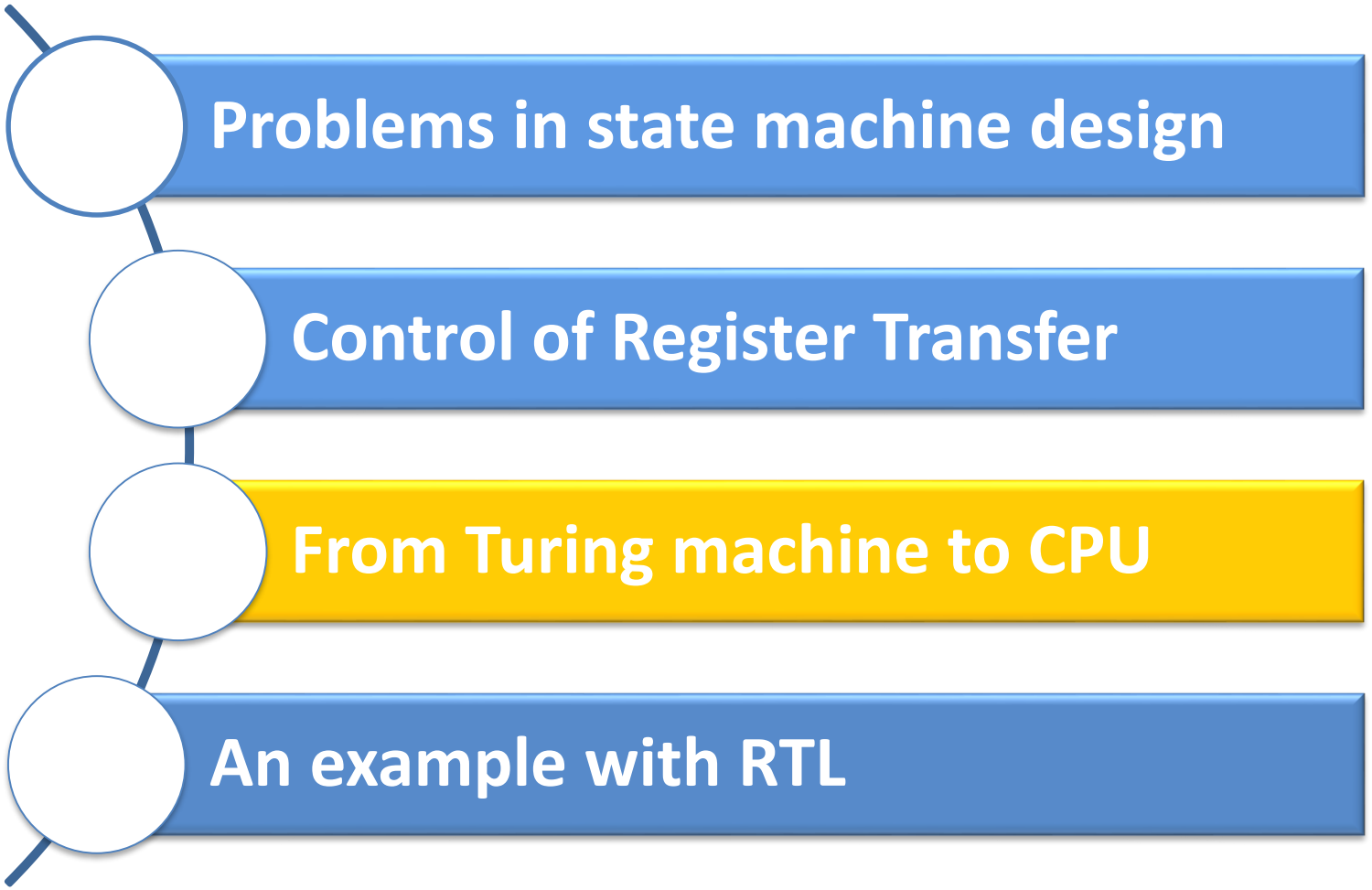


Reset： L/S:Counter<=0； Den :Disp<= Counter

X=0/1： L/S:Counter<= Counter+1; Den :Disp<= Counter

X=1/0： $\overline{L/S}$:Counter<= Counter； $\overline{Den}$ Disp<= Disp

Course Outline



Is there a better way?

--Universal Turing machine thinking

考察图灵机

◎ 基本思想

- 用机器来模拟人们用纸笔进行数学运算的过程
- 两种简单的动作
  - 纸上写上或擦除某个符号
  - 把注意力从纸的一个位置移动到另一个位置
- 下个动作依赖于
  - （a）此人当前所关注的纸上某个位置的符号
  - （b）此人当前思维的状态
- 图灵构造出一台假想的机器

图灵机的组成部分

1. 一条无限长的纸带TAPE

纸带被划分为连续的小格子，每个格子上包含一个来自有限集合（字母表）的符号，字母表中有一个特殊的符号□表示空白。纸带上的格子从左到右依次被编号为0, 1，2，...，纸带的右端可以无限伸展

2. 一个读写头HEAD

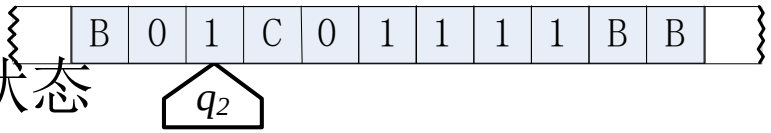
该读写头可以在纸带上左右移动，它能读出当前所指的格子上的符号，并能改变当前格子上的符号

3. 一套控制规则TABLE

它根据当前机器所处的状态以及当前读写头所指的格子上的符号来确定读写头下一步的动作，并改变状态寄存器的值，令机器进入一个新的状态

4. 一个状态寄存器

它用来保存图灵机当前所处的状态



图灵机模型：马文•闵斯基 （1967）

图灵机指令结构

◎ 典型七元组（有多种形式的变体）

- 典型的是一个七元组[3]，M={Q，Σ，Γ，δ，q0，qaccept，qreject}，其中 Q，Σ，Γ 都是有限集合

1. Q 是状态集合；
2. Σ是输入字母表，其中不包含特殊的空白符“□”；
3. Γ是字母表，其中 b∈Γ为空白符，且Σ∈Γ；
4. δ是转移函数：Q × Σ→Q ×Γ{L,R}，其中L、R表示读写头是向左移还是向右移；
5. q0∈Q是起始状态；
6. qaccept是接受状态；
7. qreject是拒绝状态，且qreject≠qaccept。

图灵机程序

◎ 状态转移函数

- 程序

  δ={ qi Sj Sk R(L或N) ql }

  qi表示机器目前所处的状态；

  sj表示机器从方格中读入的符号；

  sk：表示机器用来代替Sj写入方格中的符号；

  R、L、N表示向右移一格、向左移一格或不移动；

  ql：表示下一步转移的机器状态。

- 例如： δ={q0 0 0 R q0； q0 1 0 R q0； }
  - 右移并抹去所有经过的纸带上的内容
  - 永不停止永不后退

图灵机程序δ：连续“1”检测

◎ 除起始状态q0外有7个内部状态

```
q0 B B R q1          //起始状态,读写头指向左边缘,次态为q1,如图 3-14 (a)。
q1 1 C L q2          //状态1,检测到“1”,标记C,开始计数,  如图 3-14 (b)。
q1 0 C L q7          //状态1,检测到“0”,标记C,开始清零,  如图 3-14 (c)。
q1 B B N q1(STOP)    //状态1,检测到右边缘停机,左边是检测结果，如图 3-14 (d)。
q2 0 1 L q4          //状态2,计数。从右（低位）向左（高位）计数。
q2 1 0 L q2          //状态2,计数。从右（低位）向左（高位）计数。
q2 B 1 L q4          //状态2,到左边缘计数结束，次态是q4。
q4 1 1 L q4          //状态4,读写头移到左边缘。
q4 0 0 L q4          //状态4,读写头移到左边缘。
q4 B B R q3          //读写头到左边缘,开始计数值右移1位。
q3 1 0 R q5          //计数值将右移1位，左边填“0”。
q3 0 0 R q6          //计数值将右移1位，左边填“0”。
q3 C 0 R q1          //右移结束，清计数标志。
q5 1 1 R q5          //状态5，右移“1”。
q5 0 1 R q6          //状态5，右移“1”。
q5 C 1 R q1          //右移“1”并结束右移，清计数标志。
q6 1 0 R q5          //状态6，右移“0”。
q6 0 0 R q6          //状态6，右移“0”。
q6 C 0 R q1          //右移“0”并结束右移，清计数标志。
q7 1 0 L q7          //状态7，清零。
q7 0 0 L q7          //状态7，清零。
q7 B B R q3          //检测到左边缘，清零结束。转到状态3，计数值右移一位。
```

普适图灵机实现

- **理想图灵机是无法实现的**
  - 无法找到无限长的纸带
    - 也不是通用的计算系统
    - 只是固定内容机械的计算系统
- **普适(通用)图灵机**
  - 足够长的磁带并将其首尾相连来代替无限长纸带
  - 任意一台图灵机指令表编码
    - 写在**纸带/磁带**的开始部分
  - 作用于输入的余下部分
  - 效率依然很低
- **存在2个问题**
  - 载体
  - 效率

可实用的图灵机--真正意义上的计算机

- **一个足够大的存储器来代替磁带**
  - 磁芯存储器代替磁带
  - 半导体存储器代替磁带
- **保存内态**
  - 内态不保存计算的中间值，使得效率非常低下
  - 寄存器来存放中间值（相当于磁带的子集）
  - 可以极大地简化计算结构并提高计算效率
- **增强读写“头”处理能力--CPU**
  - 需要一个更强的读写头
  - 寄存器传输控制技术可以实现：CPU
  - 令人相当满意的通用图灵机

Specific and general system

◎ Non-programmable System: specific System

- the control unit does not deal with fetching and executing instructions
- but contains all of the information for sequencing register transfers based on inputs and on status bits from the datapath

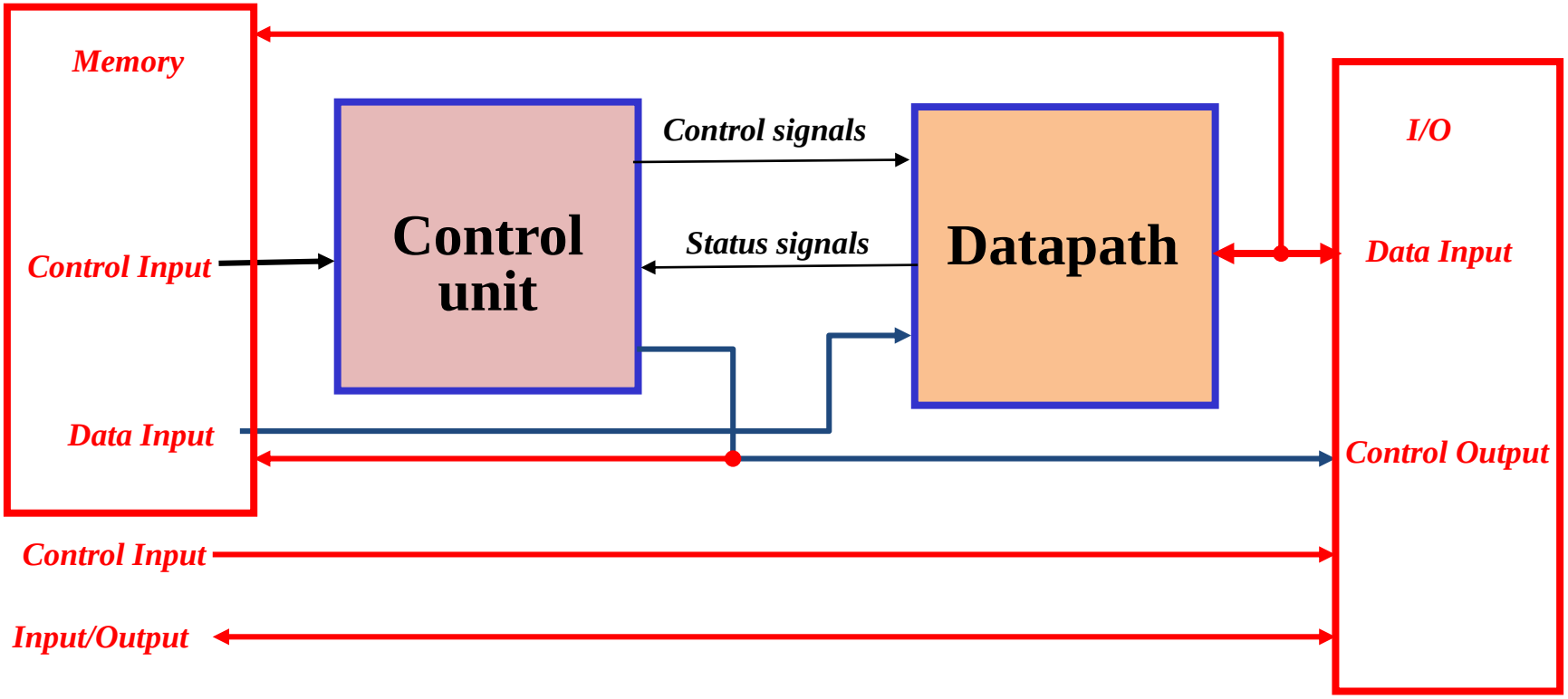
◎ Programmable System : General system

- a portion of the input consists of a sequence of *instructions* called a *program*,
- typically stored in a memory and addressed by a *program counter*.
- The Control Unit is responsible for fetching and executing these instructions.

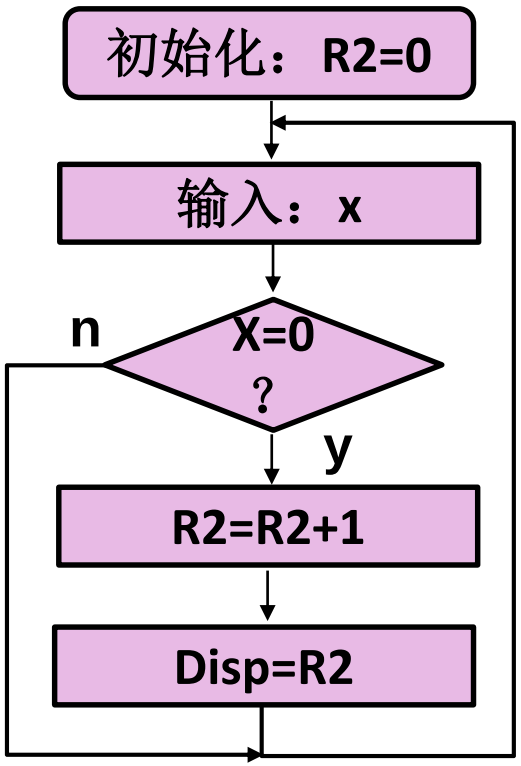
Digital System Design Method 3: *Programmable* Control of Register Transfers

◎ General system

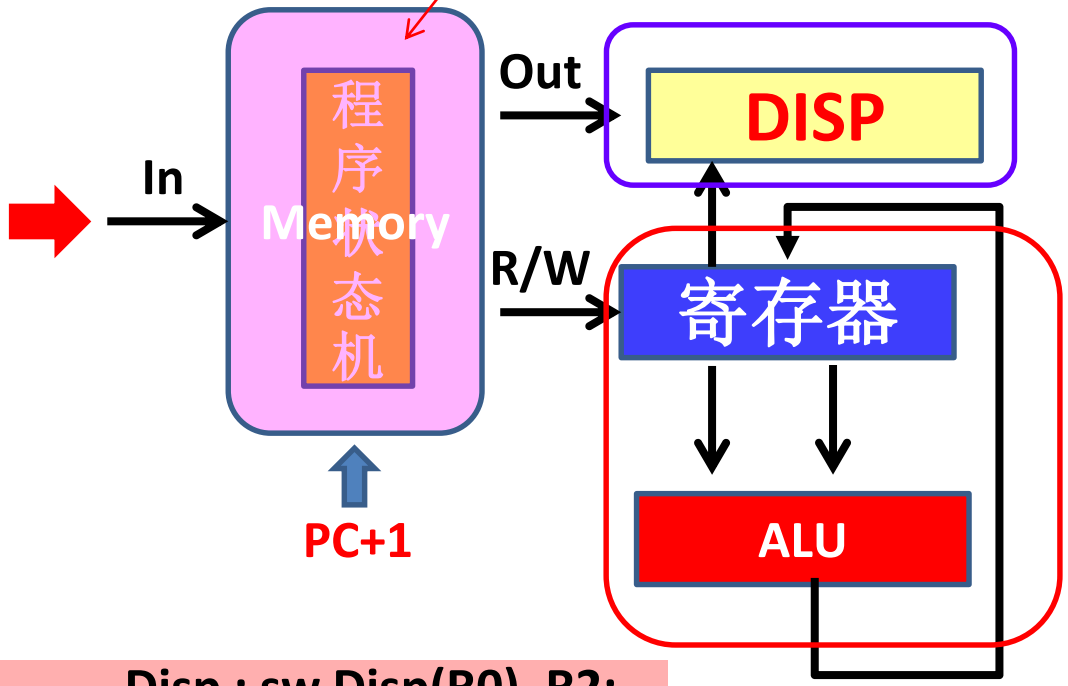
- Program state machine （PSM）



可编程寄存器传输技术实现通用普适图灵机



问题改变，程序状态机改变




```
Reset:   add R2, R0,R0;          Disp : sw Disp(R0), R2;
Input:   lw R3, InPort(X);
X=0:     addi R2, R0,1;          Disp :Sw Disp(R0), R2;
X=1:     Nop;
```

From Turing machine to CPU

```
main( )
    {……
    if i=0
        counter=+1;
    else
        counter=0;
    printf(counter)
    ……
    }
```

Only non-programmable designs are considered here

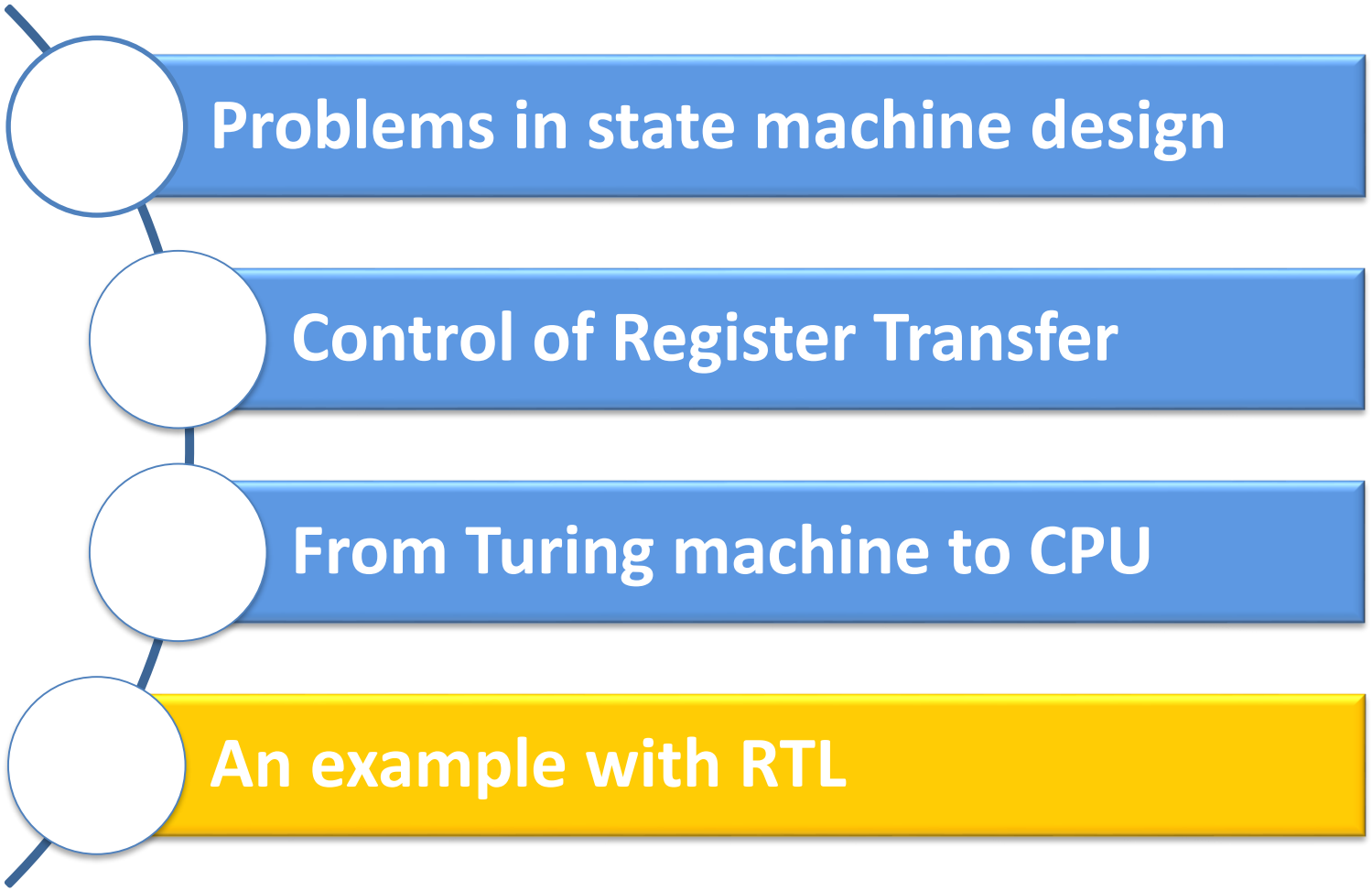
Thinking from with hardware to computer

- **CPU实现图灵机的核心思想**
  - 通用数字系统　图灵机的一种很好的实现方式
    - 存储器控制序列：解决无限长纸带
    - 三指令能做什么：读写头HEAD基本操作
    - 控制流指令：更丰富的控制规则TABLE
    - 寄存器：更丰富的状态存储和中间存储
  - 计算思维：程序解题
    - 设计算法：计算思维
    - 设计程序：软件思维
    - 编程规则：指令→规则工作流程→指令的控制流程
      - 定义指令格式：设计DataPath和控制器
      - 定义数据结构：分配存储空间（Memory、I/O）

转向CPU实现数字信息处理

  - 软硬协同处理思想

Course Outline

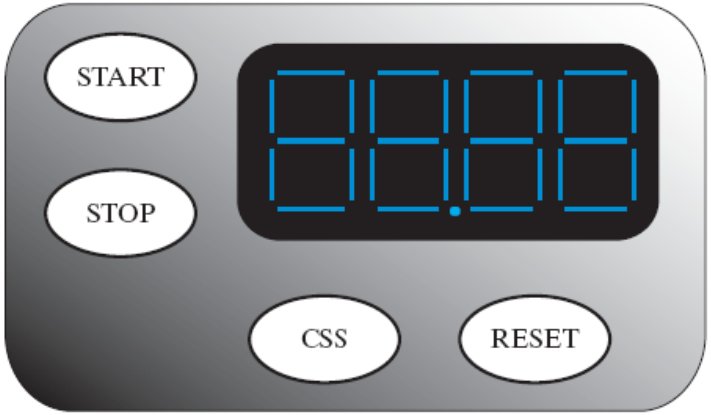


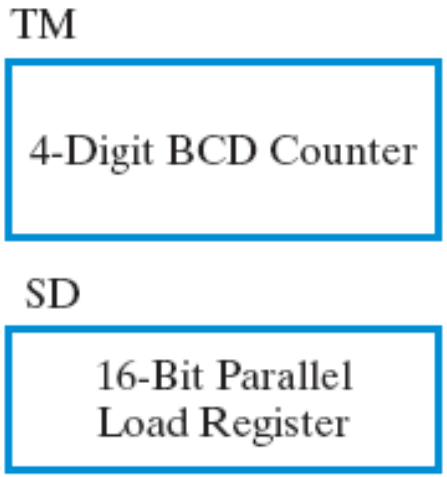
Register Transfer System Design Procedure

- **Write a detailed system specification**
- **Determine all data, control and status input signals, all data, control and status output signals, and registers of the datapath and control unit.**
- **Find a state machine diagram for the system**
  - including register transfers for the datapath and control unit as outputs.
- **Determine all internal control and status signals.**
  - Use these signals to separate output conditions and actions, including register transfers
- **Draw a block diagram of the datapath**
  - including all control and status inputs and outputs. Draw a block diagram of the control if it includes register transfer hardware.
- **Design any specialized register transfer logic**
  - as needed for the datapath and the control.
- **Design the control unit logic.**
- **Verify the correct operation of the combined datapath and control unit. If verification fails, debug the system and verify the changed system.**

Design Example – DASHWATCH - Specs

- Very Inexpensive Stop Watch for “dash” runners
- Times intervals to at most 99.99 seconds
- Stopwatch action plus storage of best performance time per session (session ended by turning off power or pushing RESET)
- Inputs START, STOP, CSS (compare and store shortest), RESET
- Registers: 4-digit BCD Counter and 16-bit Parallel Load Register
- Output: 4 digit BCD LCD with decimal point



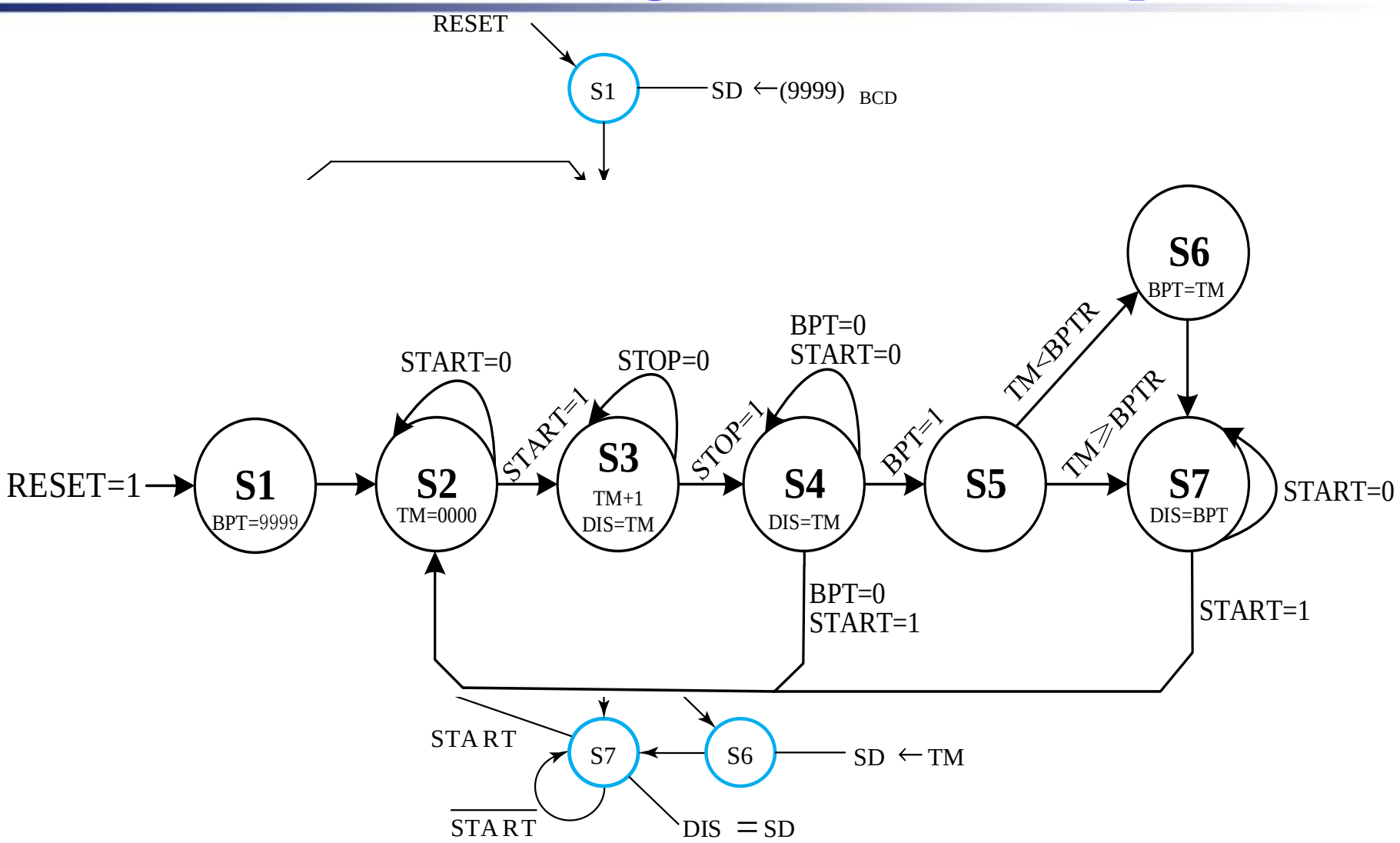


DASHWATCH Inputs, Outputs, and Registers

TABLE 7-15
Inputs, Outputs, and Registers of the DashWatch

Symbol	Function	Type
START	Initialize timer to 0 and start timer	Control input
STOP	Stop timer and display timer	Control input
CSS	Compare, store and display shortest dash time	Control input
RESET	Set shortest value to 10011001	Control input
B_1	Digit 1 data vector a, b, c, d, e, f, g to display	Data output vector
B_0	Digit 0 data vector a, b, c, d, e, f, g to display	Data output vector
DP	Decimal point to display (= 1)	Data output
B_{-1}	Digit –1 data vector a, b, c, d, e, f, g to display	Data output vector
B_{-2}	Digit –2 data vector a, b, c, d, e, f, g to display	Data output vector
B	The 29-bit display input vector (B_1, B_0, DP, B_{-1}, B_{-2})	Data output vector
TM	4-Digit BCD counter	16-Bit register
SD	Parallel load register	16-Bit register

DASHWATCH State Machine Diagram with Register Transfer Outputs



State Machine Diagram Design

- Specify only Moore outputs (no particular reason)
- S1: Reset state - in this state, initialize SD to 1001100110011001 (99.99), the maximum possible dash time.
- S2: Because of Moore output spec, S1 cannot be used for this state since SD is not to be initialized again to 99.99 after having passed through states S4 or S7. TM is initialized to $(0000)_{BCD}$ for next dash.
- S3: State during dash. Entered with START and exited with STOP. While in state, 1 (0.01 seconds) is added to TM for each clock pulse. (Clock frequency is 100 Hz), and DIS shows TM value.
- S4: Decision state whether to Compare, Store, and display Shortest dash time, or to continue to display TM. Also START begins new dash.
- S5: State for comparison of TM to SD.
- S6: State for loading TM into SD if TM is smaller.
- S7: State for START to begin new dash and display of SD as shortest dash time.

DASHWATCH 输出/状态表

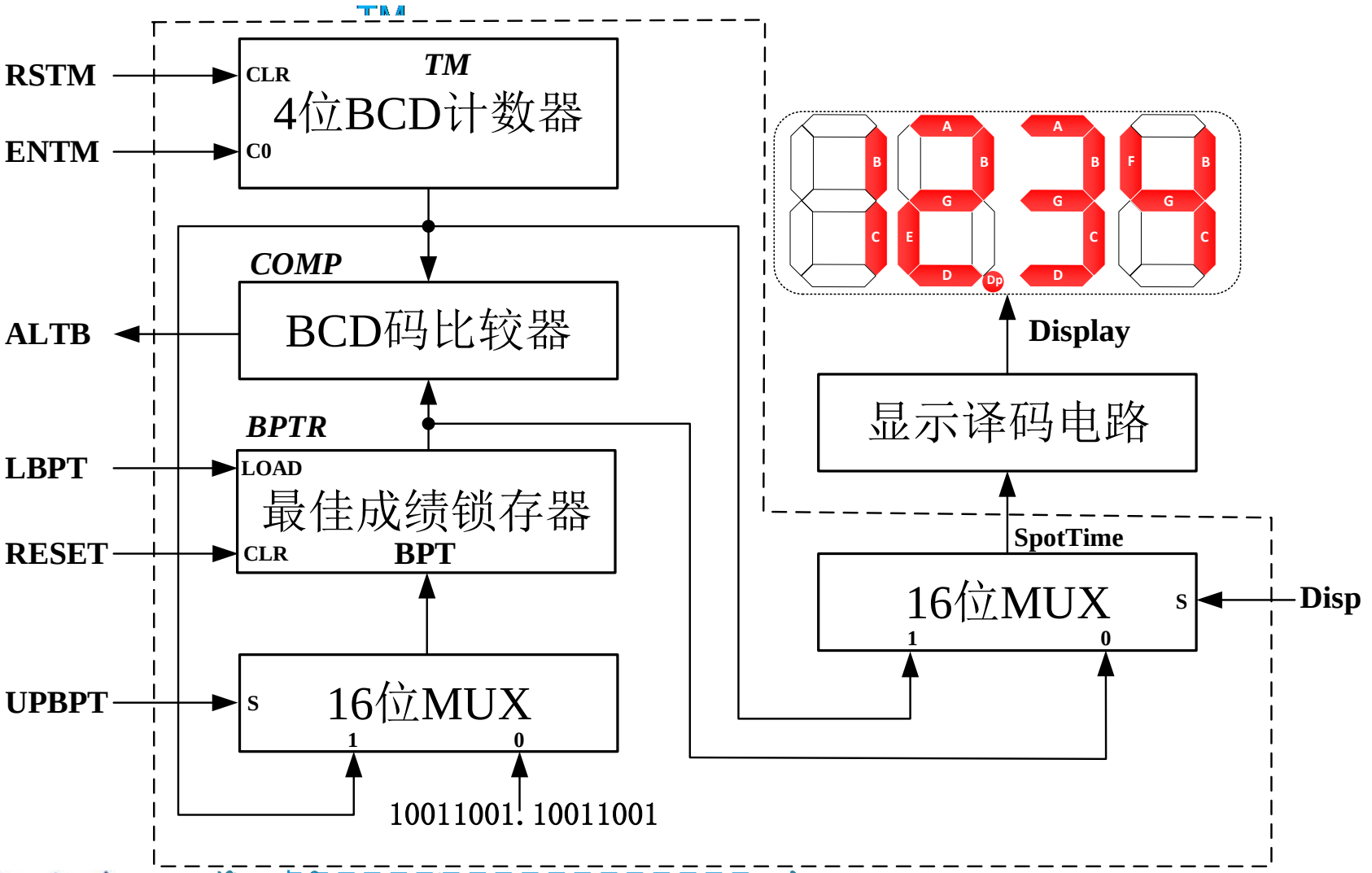
TABLE 7-16
Datapath Output Actions and Status Generation with Control and Status Signals

Action or Status	Control or Status Signals	Meaning for Values 1 and 0
$TM \leftarrow (0000)_{BCD}$	RSTM	1: Reset TM to 0 (synchronous reset) 0: No reset of TM
$TM \leftarrow (TM + 1)_{BCD}$	ENTM	1: BCD count up TM by 1, 0: hold TM value
$SD \leftarrow (9999)_{BCD}$	UPDATE LSR	0: Select 1001100110011001 for loading SD 1: Enable load SD, 0: disable load SD
$SD \leftarrow TM$	UPDATE LSR	1: Select TM for loading SD Same as above
DIS = TM DIS = SD	DS	0: Select TM for DIS 1: Select SD for DIS
TM < SD $TM \geqq SD$	ALTB	1: TM less than SD 0: TM greater than or equal to SD

Determination of Internal Control/Status Signals

- **TM – Timer**
  - Reset to 0000: RSTM
  - Enable to Count Up: ENTM
- **SD – Shortest Dash**
  - Load SD: LSR = 1;
  - Select input 9999: UPDATE = 0
  - Select input TM: UPDATE = 1
- **DIS – Display (B_1, B_0, DP, B_{-1}, B_{-2})**
  - Select input TM: DS = 0
  - Select input SD: DS = 1
- **Compare TM and SD (Status)**
  - TM < SD: ALTB = 1
  - TM ≥ SD: ALTB = 0

DASHWATCH Datapath



(a)

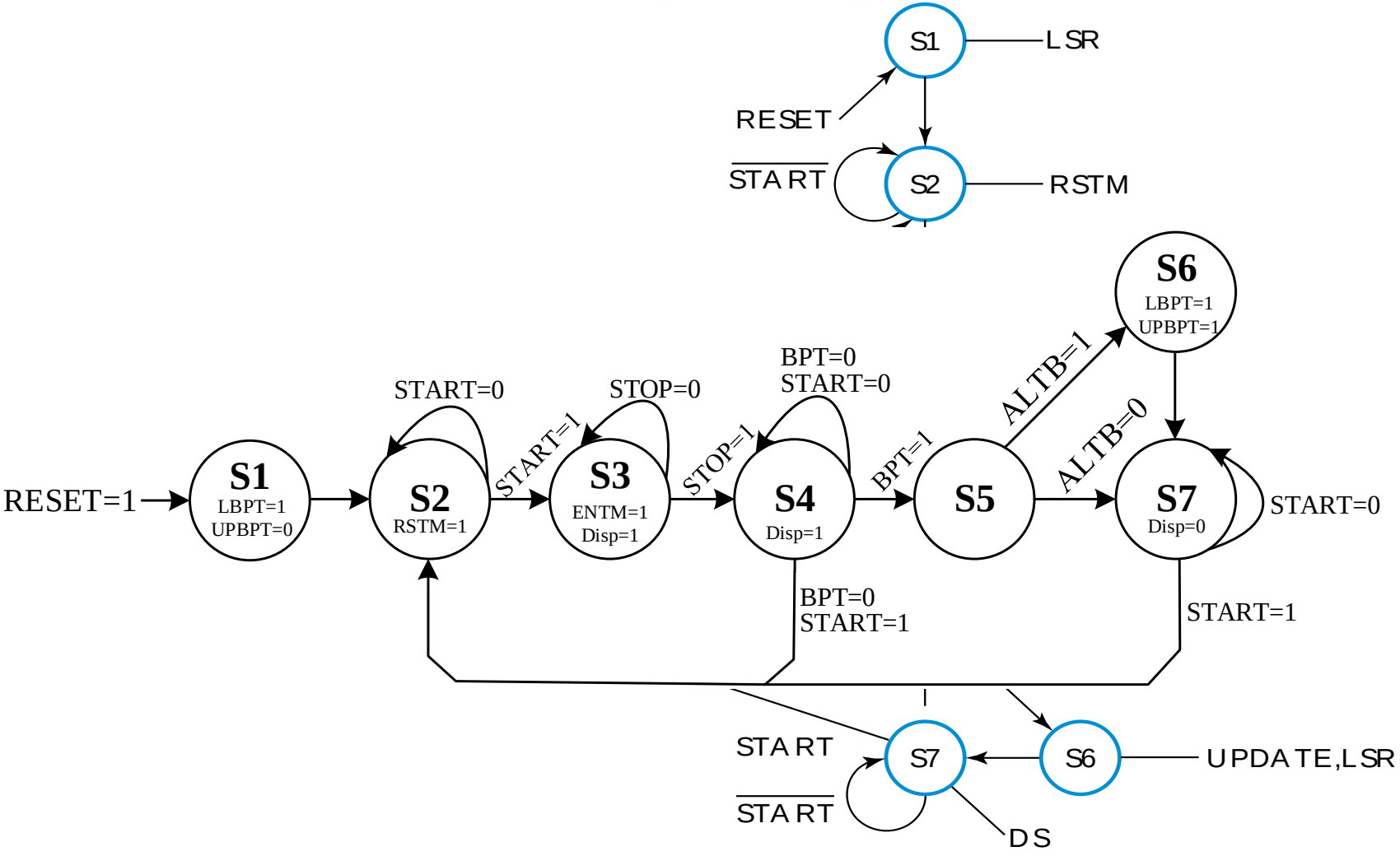
DASHWATCH – Datapath Development

- **TM: 4-digit BCD Counter with Synchronous Reset**
  - Based on previous BCD adder digit design
  - synchronous reset SRST added
  - SRST = RSTM
  - C0 (Incoming carry) = ENTM
- **A < B Comparator**
  - Compares TM to SD
  - Designed as left-to-right iterative cell array with output C0
- **SD: Standard 16-bit parallel load register**
  - LOAD = LSR
  - Contracted standard 2-way, 16-bit multiplexer used to select between 9999_{BCD} and TM as parallel load input D
  - S = UPDATE

DASHWATCH – Datapath Development – Display Logic

- **2-way 16-bit multiplexer**
  - Selects between TM and SD
  - S = DS
- **4-digit BCD-to-7 Segment Converter**
  - Uses previous design
- **4-digit 7-Segment Display with Decimal Point**
  - 2-digit fractional part
  - Decimal Point control = DP
  - DP = 1

DASHWATCH – SMD with Control Signal Outputs Replacing Register Transfers



(b)

DASHWATCH – FF Input Equations

- **One-Hot State Assignment – 7 bits**
- **State S1 entered only by using asynchronous RESET**

$$D_{S1} = S1(t+1) = 0$$

$$D_{S2} = S2(t+1) = S1 + S2 \cdot \overline{STAR\ \ T} + S4 \cdot \overline{CS\,S} \cdot STAR\ \ T + S7 \cdot STAR\ \ T$$

$$D_{S3} = S3(t+1) = S2 \cdot STAR\ \ T + S3 \cdot \overline{STO\ \ P}$$

$$D_{S4} = S4(t+1) = S3 \cdot STO\ \ P + S4 \cdot \overline{CS\,S} \cdot \overline{STAR\ \ T}$$

$$D_{S5} = S5(t+1) = S4 \cdot CS\,S$$

$$D_{S6} = S5 \cdot ALT\ \ B$$

$$D_{S7} = S7(t+1) = S5 \cdot \overline{ALT\ \ B} + S6 + S7 \cdot \overline{STAR\ \ T}$$

DASHWATCH – Output Equations

$$LSR = S1 + S6$$

$$RSTM = S2$$

$$ENTM = S3$$

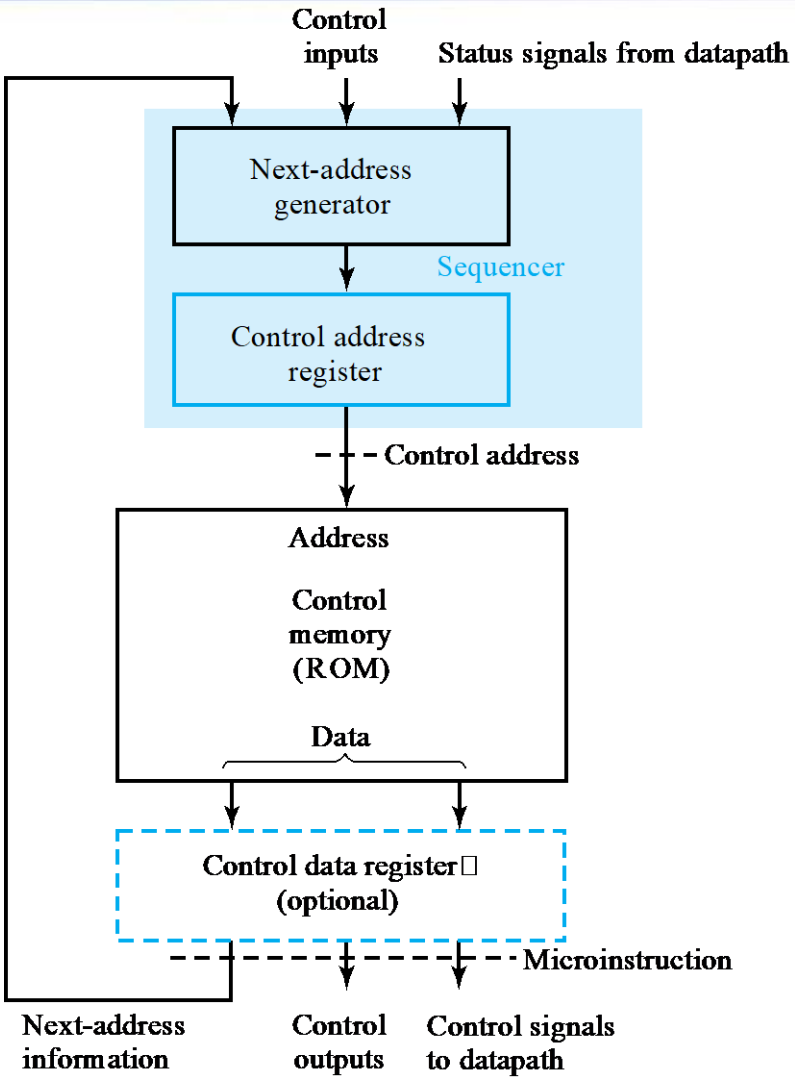
$$UPDATE = S6$$

$$DS = S7$$

Microprogrammed Control

- *Microprogrammed Control* — a control unit with binary control values stored as *words* in *memory*.
- *Microinstructions* — words in the control memory.
- *Microprogram* — a sequence of microinstructions.
- *Control Memory* — RAM or ROM memory holding the microinstructions.
- *Writeable Control Memory* — RAM Memory into which microinstructions may be written

Microprogrammed Control (continued)




Thank you!