5.5 Microprogramming

1 Introduction

- proposed by Maurice Wilkes in 1951
- a technique to implement the control unit of a (complex) digital system
- building a special computer for executing the algorithmic flow chart describing the controller of a system.

- Early microprocessors such as Intel 8086 and Motorola 68000 were microprogrammed.

  CISC : complex instruction set computing

- microprogrammed controllers : sequencers
  - the controller flow chart systematically specifies all the controller signals that should be generated at each time during the flow of control from the reset state through

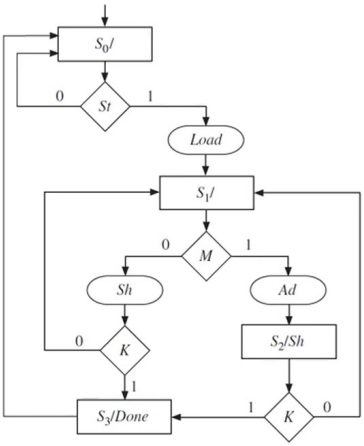
each of the other states.

- Control store or microprogram memory :
  - a component in the sequencer
  - the memory that stores the control words (microinstructions).
  - realize the controller by just sequencing through the memory

Ex.

Pseudocode for Controller Operation

- By inspection of the ***SM chart*** for a digital system, one can write pseudocode for its controller operation.
  - E.g.: the shift and add multiplier



S_0: if *St* is true, produce *Load* Signal and go to S_1,
else return to S_0
S_1: if *M* is true, produce *Ad* and go to S_2,
else produce *Sh*, check whether *K* is 1
if *K* is 1 go to S_3,
else go to S_1;
S_2: produce *Sh*;
if $K = 0$, go to S_1;
else go to S_3;
*S*3: produce *Done* and go to S_0

2 Advantages or Disadvantages

- advantages
  - debugging of systems is simple.
  - errors can be identified and corrected easily.
  - changes to system can be implemented relatively easily.
- disadvantages
  - slow

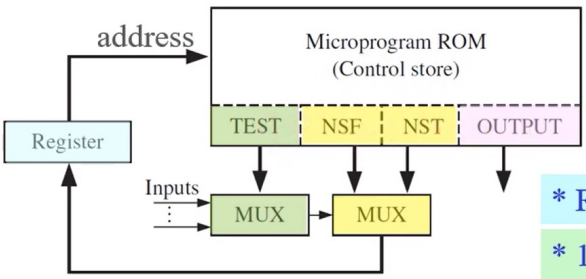
- computer architecture: CISC & RISC
  - the complexity of microprocessors led researchers and designers to the RISC era.
    - fewer memory-addressing modes
    - simpler control units
  - Today microprogramming may be used only for microprocessors w/ complex instruction set architectures (CISAs).

3 Implementation

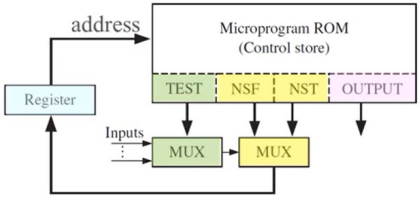
- General Idea:
  - store a control word (or microinstruction) corresponding to each state.
  - A microinstruction specifies the outputs to be generated and where the next microinstruction can be found corresponding to state transitions in the state diagram or SM chart.
  - 2 typical formats of control word:
    - ① single-qualifier two-address (SQTA)
      - single-qualifier: only 1 input can be tested in a state.
      - two-address: each microinstruction can specify 2 potential next states
        - next state if input qualifier is true (NST)
        - next state if input qualifier is false (NSF)

Typical Hardware Arrangement for SQTA Microprogramming

- Typical hardware for microprogramming w/ a single qualifier per state and two next-state addresses:
  - Each ROM location stores a ***control word*** or ***microinstruction***.



* NSF: next state if the input/qualifier is false
* NST: next state if the input/qualifier is true
* Register: state register
* 1st MUX: choose a single qualifier from the inputs



- Inputs and outputs of the microprogram ROM:
  - ROM input: ***address***
    - The only inputs, i.e., the address, to the ROM come from the ***state register***.
  - ROM output: ***microinstruction***, 4 fields
    - **TEST**: control the input MUX, which selects one of the inputs to be tested in each state, i.e., specifies the qualifier being tested.
    - **NSF** & **NST**: If the selected input is 0 (false), the 2nd MUX selects the NSF field as the next state; otherwise, it selects the NST field as the next state.
    - **OUTPUT**: The OUTPUT bits correspond to the ***control signals***.

② single-qualifier single-address (SQSA)

- single-address: each microinstruction needs to specify only one next-state address
  - only NST will be specified in the microcode
  - NSF = next row (succeeding microinstruction)

- The state assignment should be such that, if the qualifier is false, the next state should be the current state + 1.
  - If this is not possible, extra states (X-states) must be added.
  - To reduce X-states, assign long strings of states in sequence.

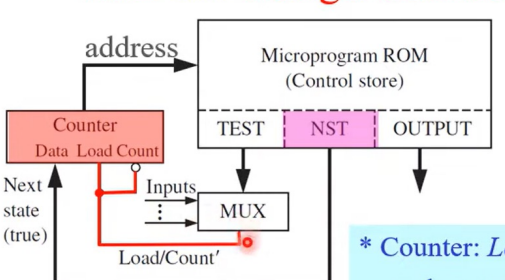
    ⇒ It may be necessary to complement some of the tested variables.

- pro: reduce width of μ-instructions.
- con: may require extra states to fulfill state assignment requirements.

* single qualifier: Although microprogramming can be done w/ multiple qualifiers per state, it is simpler to implement microprogramming when only one qualifier / variable is tested in each state.
  - 2^n next-state addresses for n qualifiers

Typical Hardware Arrangement for SQSA Microprogramming

- Typical hardware arrangement for microprogramming w/ a single qualifier per state and a single next-state address:



* No NSF field
* NST: next state if the input/qualifier is true

* Counter: *Load* & *Count* (+1)
  analogous to a program counter (PC), points to the current state of the controller
* MUX: choose a single qualifier from the inputs

4 SM Chart Transformations for Microprogramming

- Transform the SM chart such that only one entry is required per state.
  - some of the transformations may increase # (states)

(i) eliminate conditional outputs
- Construct the controller as a Moore machine, i.e. no conditional control signals
- If control signals are conditional on some inputs, store control signals corresponding to different combinations of inputs.

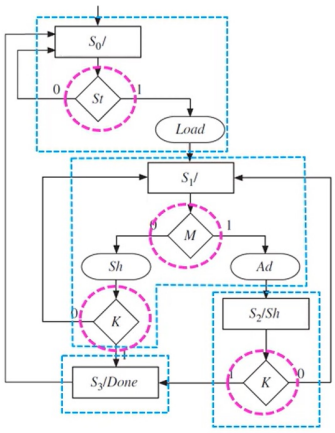
* Any Mealy machine can be converted into a Moore machine by adding an appropriate # (additional states).

↗ inputs that are tested in each state of the state machine

(ii) allow only one qualifier per state

Allowing Only One Qualifier Per State

- ***qualifiers***: inputs that are tested in each state of the state machine
  - E.g.:
    - *St*, *M*, and *K* are qualifiers.
    - S_0 and S_2 contain only one qualifier.
    - S_1 tests qualifiers *M* and *K*:
      - *The multiple qualifiers in a state led to **nested if statements** in the pseudocode.*

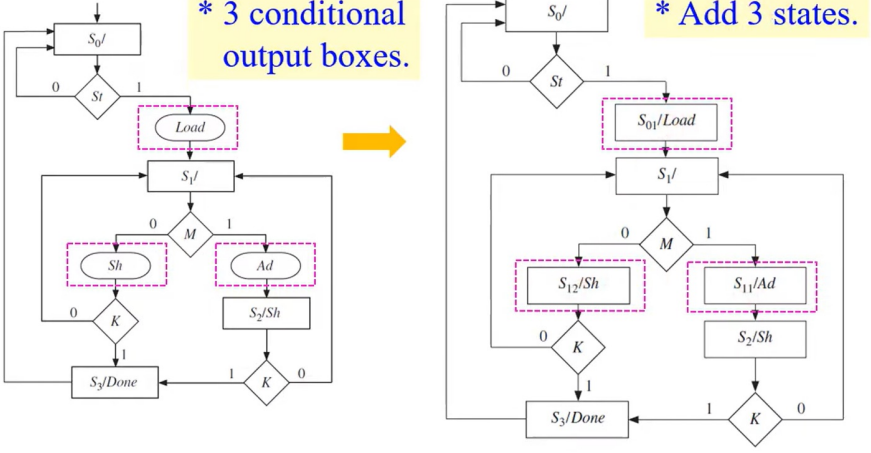


Ex1, SQTA

1. Eliminate conditional outputs

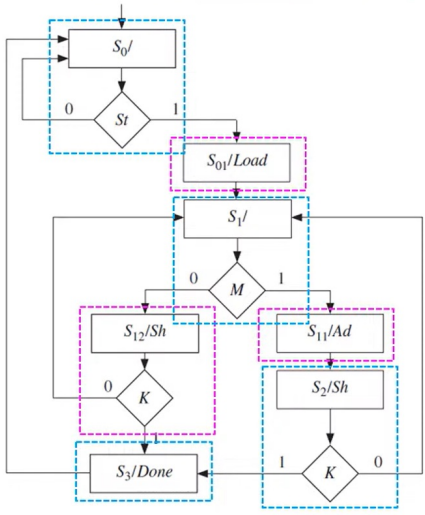
<Ans.>

- Transform the SM chart of the multiplier



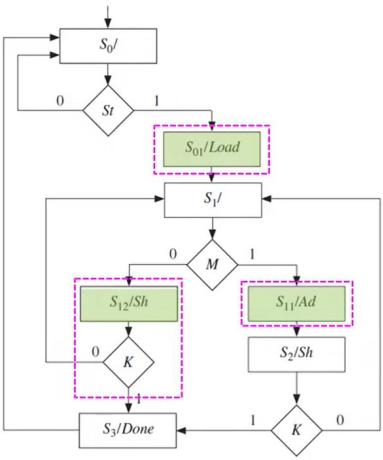
1. Eliminate conditional outputs
2. Allow only one qualifier per state

- Transform the SM chart of the multiplier



(✓)

- Corresponding pseudocode:

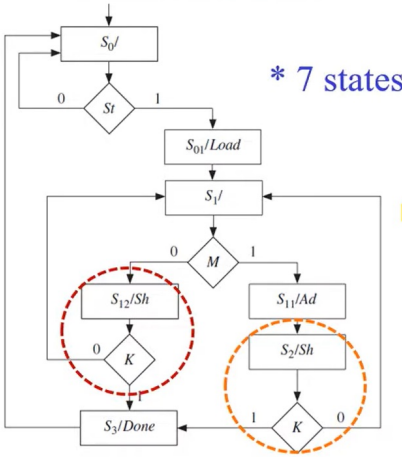


S_0: if *St* is true, go to S_{01}, else go to S_0;
S_{01}: produce *Load*; Go to S_1;
S_1: if *M* is true, go to S_{11}, else go to S_{12};
S_{11}: produce *Ad*; go to S_2;
S_{12}: produce *Sh*; if $K = 0$ go to S_1; else go to S_3;
S_2: produce *Sh*; if $K = 0$, go to S_1; else go to S_3;
$S3$: produce *Done*; go to S_0

* 7 states

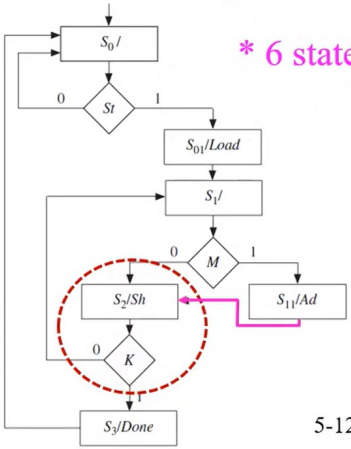
* At most one qualifier per state

- ***State minimization***: eliminate/combine redundant states



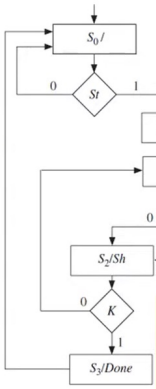
* 7 states

Improved SM chart:

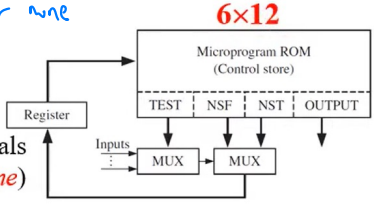


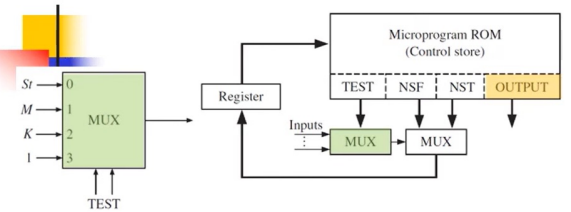
* 6 states

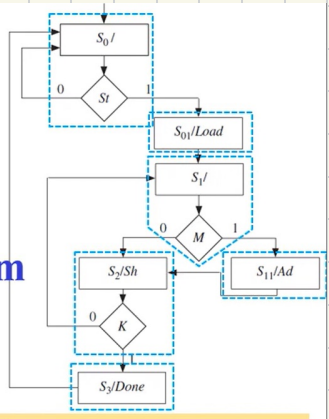
- Hardware arrangement for multiplier:
  - a 3-bit register: six states
  - Microprogram ROM: 6×12
    - six entries: one for each state
    - 12 bits/entry:
      TEST: 2 bits (3 inputs *St*, *M*, *K*)
      NSF: 3 bits
      NST: 3 bits
      OUTPUT: 4 bits (for control signals *Load*, *Ad*, *Sh*, and *Done*)
  - a 4-1 MUX & a 2-1 MUX



* 6 states
3 input conditions
4 outputs







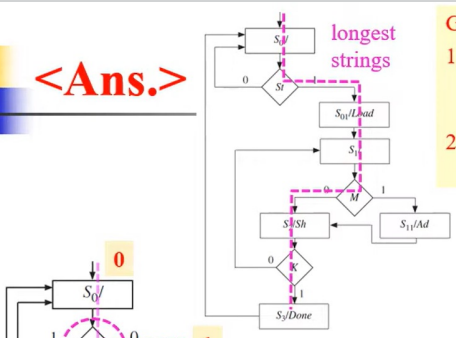
- **Two-address microprogram** for multiplier:

the addr at which the microinstr is stored

State	ABC	TEST	NSF	NST	Load	Ad	Sh	Done
S_0	000	00	000	001	0	0	0	0
S_{01}	001	11	010	010	1	0	0	0
S_1	010	01	100	011	0	0	0	0
S_{11}	011	11	100	100	0	1	0	0
S_2	100	10	010	101	0	0	1	0
S_3	101	11	000	000	0	0	0	1

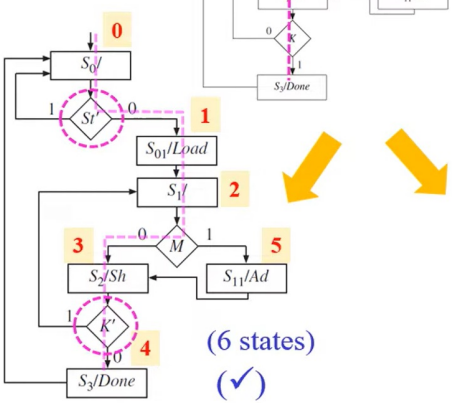
Ex2. 5Q5A

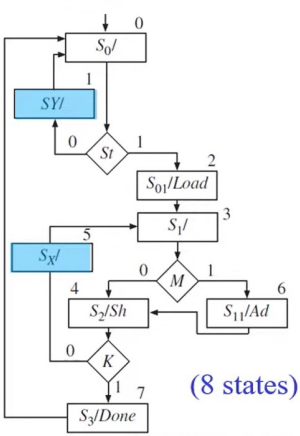
<Ans.>

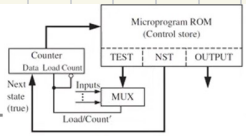


Guideline of state assignment:

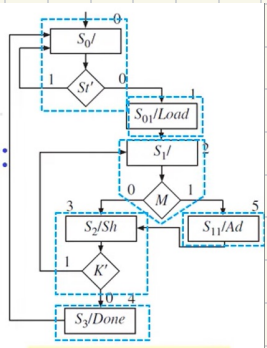
1. For each condition box, for the **false branch**, the next state must be assigned in sequence, if possible.
2. Assign **long strings** of states in sequence, if possible.







6×9



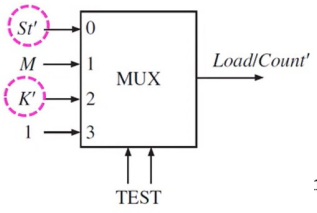
- Microprogram for the multiplier:

State	ABC	TEST	NST	Load	Ad	Sh	Done
S_0	000	00	000	0	0	0	0
S_{01}	001	11	010	1	0	0	0
S_1	010	01	101	0	0	0	0
S_2	011	10	010	0	0	1	0
S_3	100	11	000	0	0	0	1
S_{11}	101	11	011	0	1	0	0

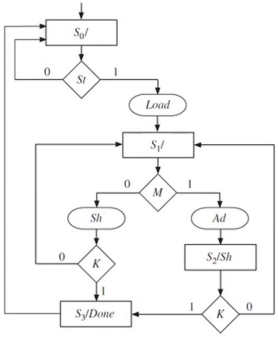
* 6 states.
3 input conditions.
4 outputs.

  - Microprogram ROM:
    - 6 entries, 9 bits/entry

- Multiplexer for the multiplier:



- Standard ROM (LUT) implementation of the multiplier: original SM chart
  - 4 states ⇒ 2 flip-flops, 2 next state equations
  - 3 inputs: St, M, K
  - 4 outputs: *Load*, *Sh*, *Ad*, *Done*
  - ROM size: 32 × 6



A	B	St	M	K	A^+	B^+	Ld	Sh	Ad	Done
...										

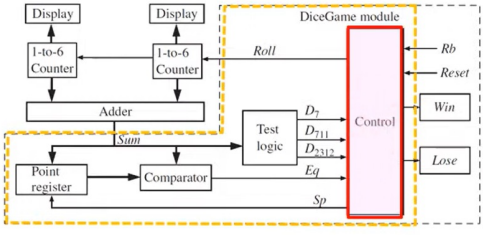
- Comparison of different implementations of the multiplier control:

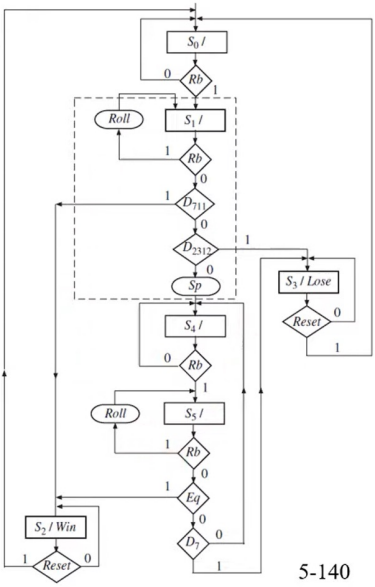
Method	Size of ROM	
	#entries × width	# bits
ROM method with original SM chart	32 × 6	192 bits
Two-address microcode	6 × 12	72 bits
Single-address microcode	6 × 9	54 bits

5 Microprogramming Dice Game Controller

C. Microprogramming Dice Game Controller

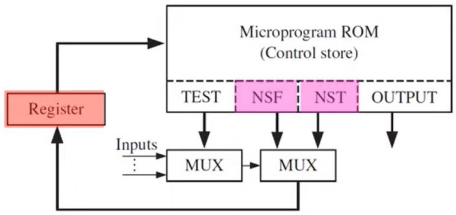
- Dice game: §5-2
  - Block diagram
  - SM chart of the controller
    - 6 states
    - 6 inputs: *Rb*, *Reset*, D_7, D_{711}, D_{2312}, *Eq*
    - 4 outputs: *Roll*, *Sp*, *Win*, *Lose*

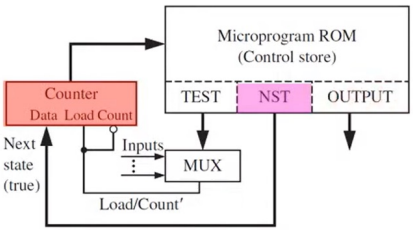




Microprogramming Dice Game Controller

- Microprogrammed the dice controller by using
  i. two-address microcoding (*SQTA*) or
  ii. single-address microcoding (*SQSA*).



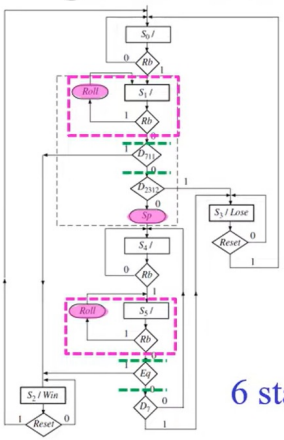


(a) SQTA

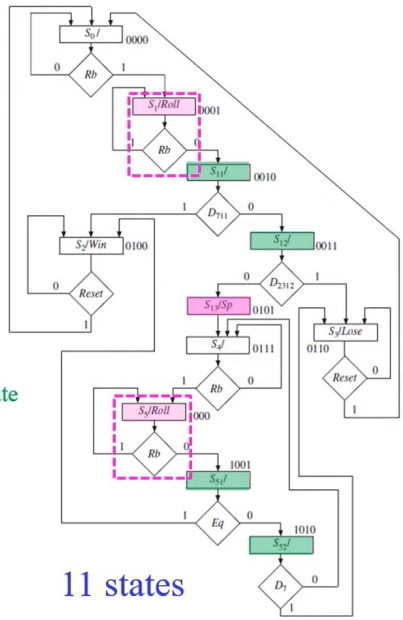
SM Chart for 2-Address Microcode

- SM chart w/ Moore outputs and one qualifier per state :

Original SM chart



1. Moore outputs
2. One qualifier/state



6 states

11 states

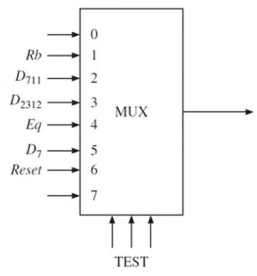
Two-Address Microprogram

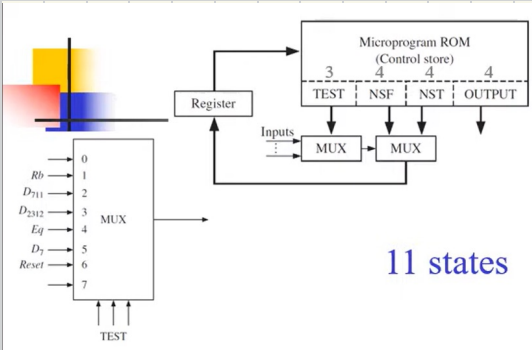
- Two-address microcode implementation:
  - Derive the microprogram using a ***straight binary state assignment***.
  - The input variables Rb, D_{711}, D_{2312}, Eq, D_7, and $Reset$ must be tested ⇒ Use an 8-to-1 MUX.

(11 states)

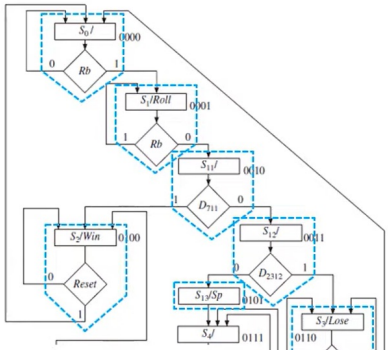
State	ABCD	TEST (3 bits)	NSF (4 bits)	NST (4 bits)	ROLL	Sp	Win	Lose (4 bits)
S_0	0000							
S_1	0001							
S_{11}	0010							
S_{12}	0011							
S_2	0100							
S_{13}	0101							
S_3	0110							
S_4	0111							
S_5	1000							
S_{51}	1001							
S_{52}	1010							

Microprogrammed ROM: 11 × 15





11 states



State	ABCD	TEST	NSF	NST	ROLL	Sp	Win	Lose
S_0	0000	001	0000	0001	0	0	0	0
S_1	0001	001	0010	0001	1	0	0	0
S_{11}	0010	010	0011	0100	0	0	0	0
S_{12}	0011	011	0101	0110	0	0	0	0
S_2	0100	110	0100	0000	0	0	1	0
S_{13}	0101	xxx	0111	0111	0	1	0	0
S_3	0110	110	0110	0000	0	0	0	1
S_4	0111	001	0111	1000	0	0	0	0
S_5	1000	001	1001	1000	1	0	0	0
S_{51}	1001	100	1010	0100	0	0	0	0
S_{52}	1010	101	0111	0110	0	0	0	0

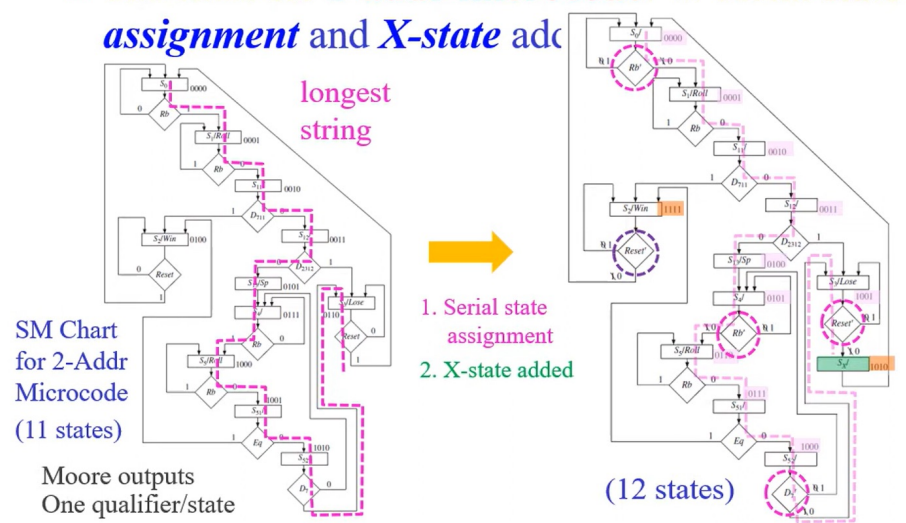
(b) SQSA

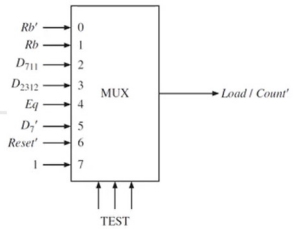
SM Chart for 1-Addr Microcode

- Modification of SM chart for single-addr implementation:
  - ***Serial state assignment***:
    - State assignment be made in a serial fashion.
  - ***X-state*** added:
    - If serial state assignment is not possible, extra states are added.
    - The required # of X-states can be reduced by assigning ***long strings of states*** in sequence. It may be necessary to ***complement*** some of the variables that are tested.

SM Chart for Dice Game

- SM chart for **1-addr microcode**: w/ ***serial state assignment*** and ***X-state*** add

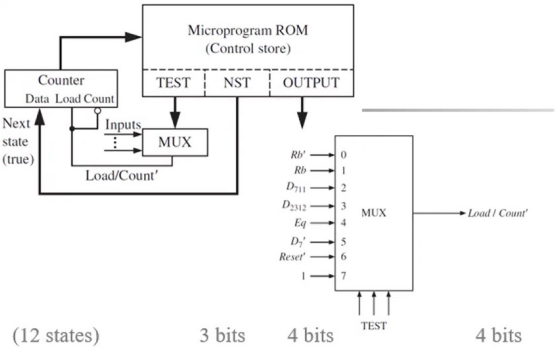


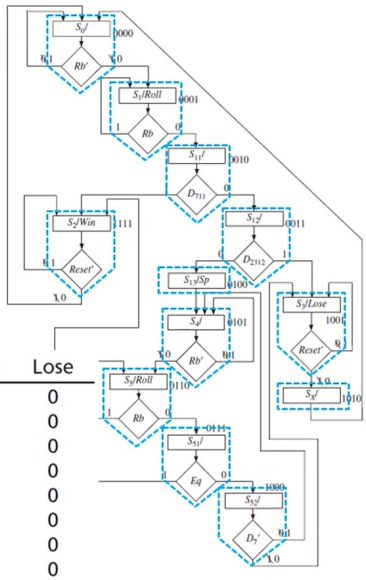


Singel-addr Microprogram for Dice Game:

(12 states) State	ABCD	3 bits TEST	4 bits NST	4 bits ROLL	Sp	Win	Lose
S_0	0000						
S_1	0001						
S_{11}	0010						
S_{12}	0011						
S_{13}	0100						
S_4	0101						
S_5	0110						
S_{51}	0111						
S_{52}	1000						
S_3	1001						
S_x	1010						
S_2	1111						

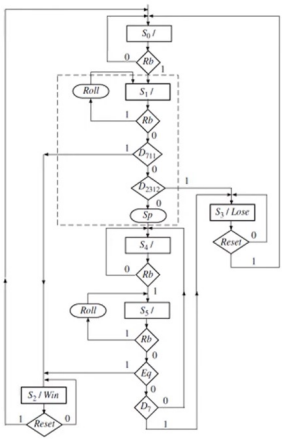
Microprogrammed ROM: 12×11





(12 states) State	ABCD	3 bits TEST	4 bits NST	4 bits ROLL	Sp	Win	Lose
S_0	0000	000	0000	0	0	0	0
S_1	0001	001	0001	1	0	0	0
S_{11}	0010	010	1111	0	0	0	0
S_{12}	0011	011	1001	0	0	0	0
S_{13}	0100	111	0101	0	1	0	0
S_4	0101	000	0101	0	0	0	0
S_5	0110	001	0110	1	0	0	0
S_{51}	0111	100	1111	0	0	0	0
S_{52}	1000	101	0101	0	0	0	0
S_3	1001	110	1001	0	0	0	1
S_x	1010	111	0000	0	0	0	0
S_2	1111	110	1111	0	0	1	0

- Standard ROM (LUT) implementation of the Dice Game: original SM chart
  - 6 states
    - ⇒ 3 flip-flops
      3 next state equations
  - 6 inputs
  - 4 outputs 6 + 3
  - ROM size: $2^9 \times 7 = 512 \times 7$
    4 + 3



(c) Comparison

- Comparison of different implementations of the Dice Game:

Method	Size of ROM	
	#entries × width	#bits
ROM method with original SM chart	512 × 7	3584 bits
Two-address microcode	11 × 15	165 bits
Single-address microcode	12 × 11	132 bits