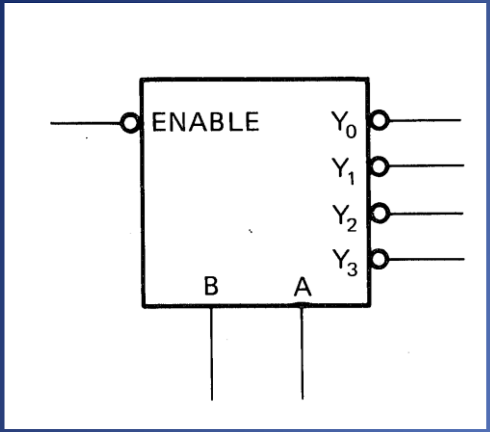
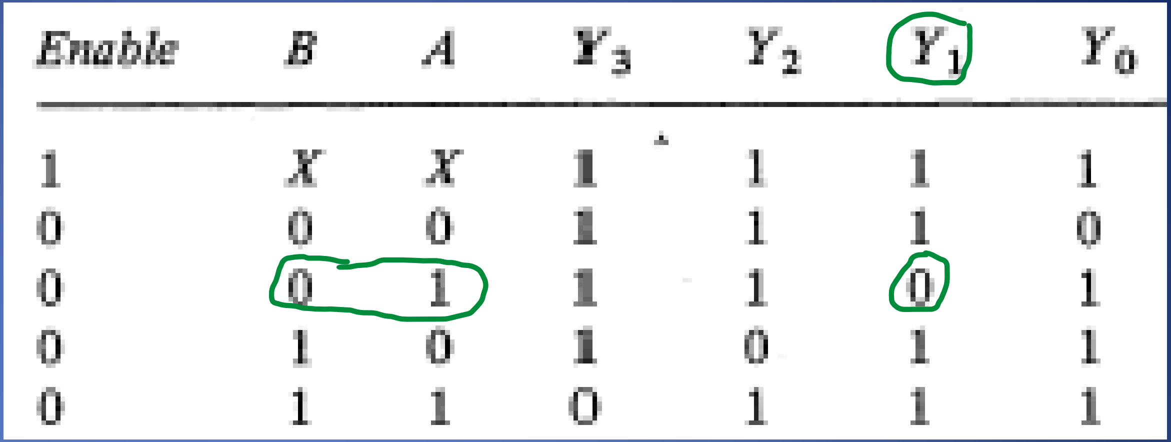
Digital Systems

Physics 5430

Wiatrowski Chapter 3 Decoders, Registers, Counters

Decoders





Enable	B	A	Y_3	Y_2	Y_1	Y_0
1	X	X	1	1	1	1
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1

4 to 1 decoder routes a single input to one of four outputs.
BA are the select lines.
Input and outputs are ACTIVE LOW.
Enable is the input, must be tied low.
If enable is high, all outputs are off (high).

Only one output on at a time (mutually exclusive).
Must have one on at all times if enabled.

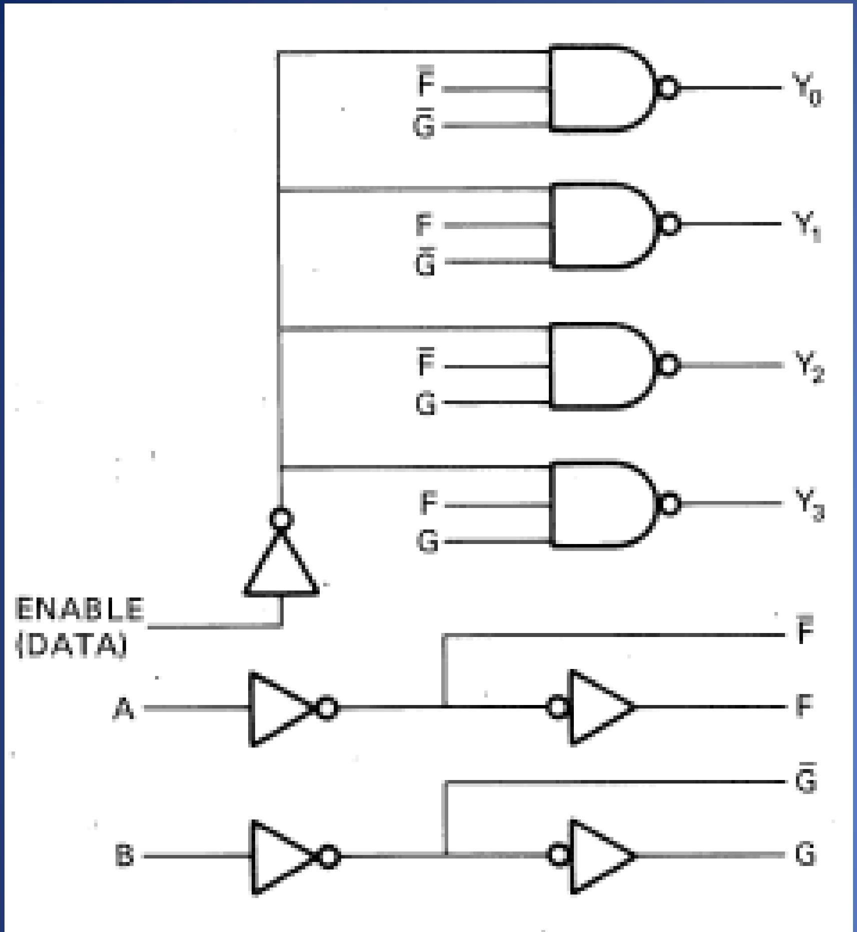
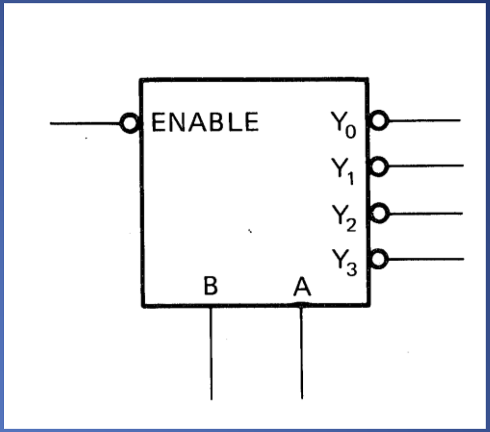
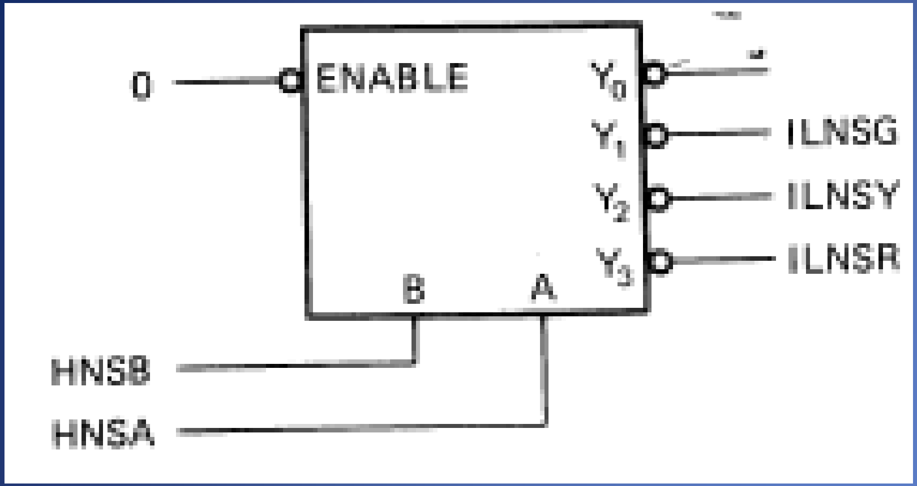


Figure 3-18 Decoder.



Mirror image of multiplexer.
Uses four enable gates
 (one for each output).
Can also get:
 1 to 2 (1 select line)
 1 to 8 (3 select lines)
 1 to 16 (4 select lines)

Decoders in ASM's



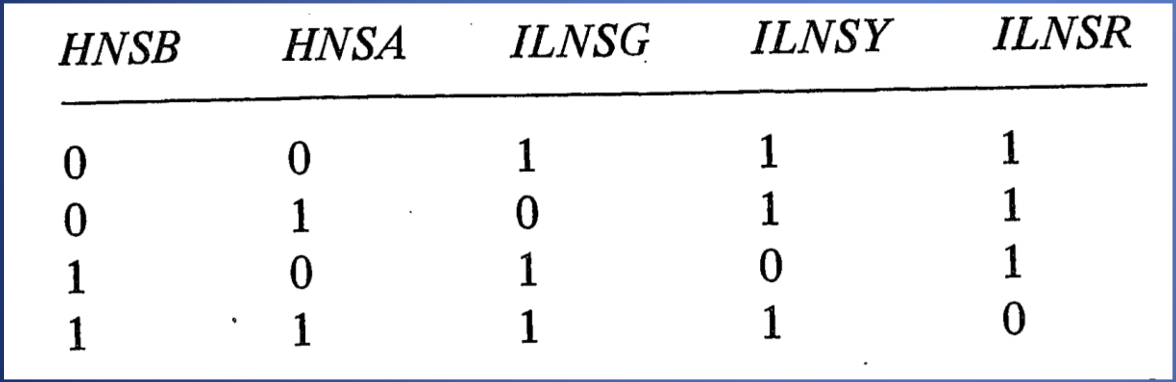
Can get 3 outputs from 2 ROM pins.

Tie Enable to GND.

Useful for a traffic light design.

Notice active-low lights!

Unused output could be used for blinking NS off in YNITE mode.



HNSB	*HNSA*	*ILNSG*	*ILNSY*	*ILNSR*
0	0	1	1	1
0	1	0	1	1
1	0	1	0	1
1	1	1	1	0

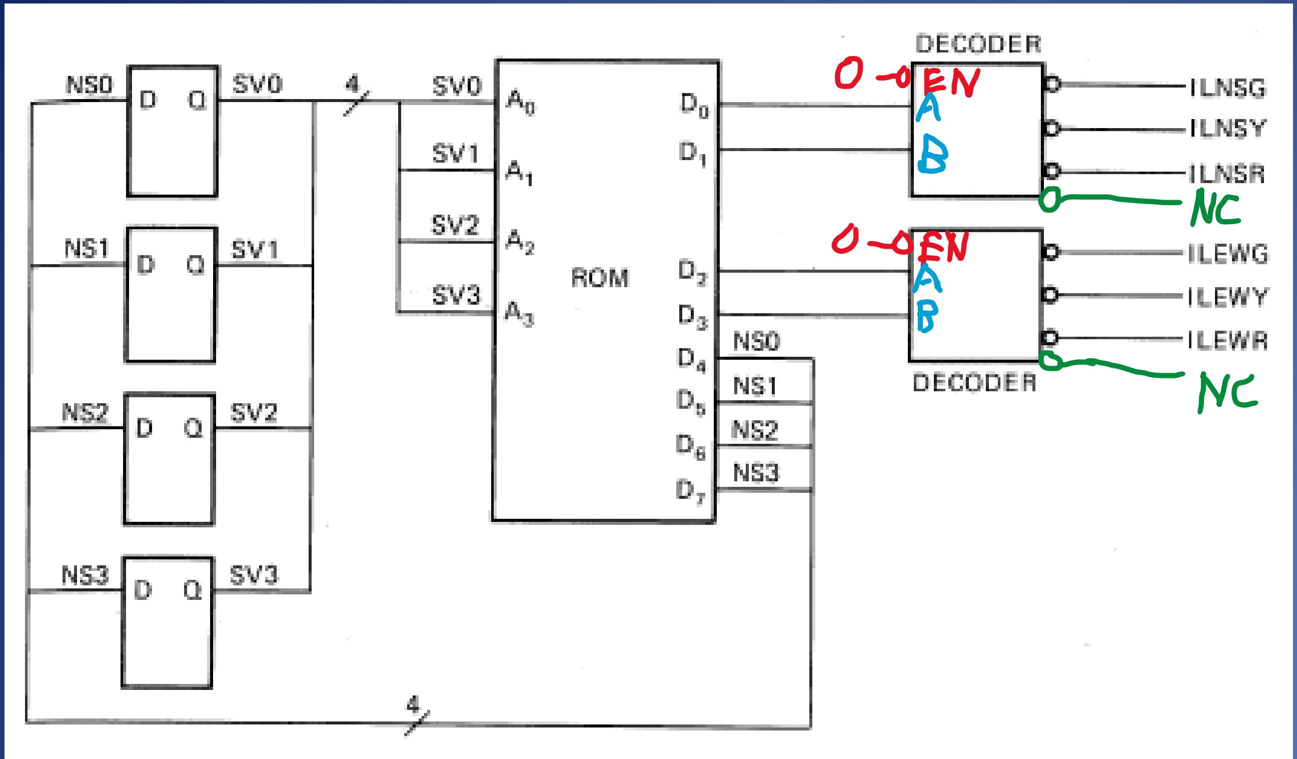


Figure 3-20 Traffic-light controller with decoders.

Use 2 decoders.
Saves 2 ROM outputs.

Two decoders is a perfect match for traffic light ASM because :
- Three lights per direction.
- Only one of the three is on at a time.
- Unused output can be selected to turn off all 3 lights.

However, in general, mutually exclusive restriction (one on at a time) could require more states.

Registers

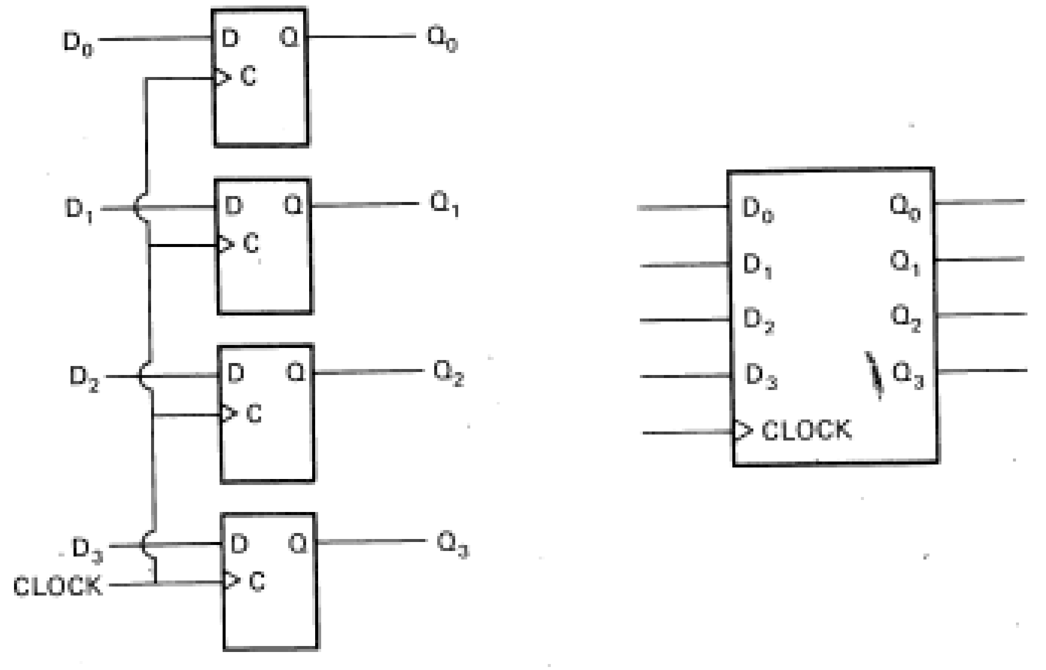


Figure 3-21 4-bit register and MSI circuit symbol.

A register is a group of flip flops with a common clock input.
Usually have multiples of 4 D FFs.
Usually used as the state register for ASMs.

Register with Enable

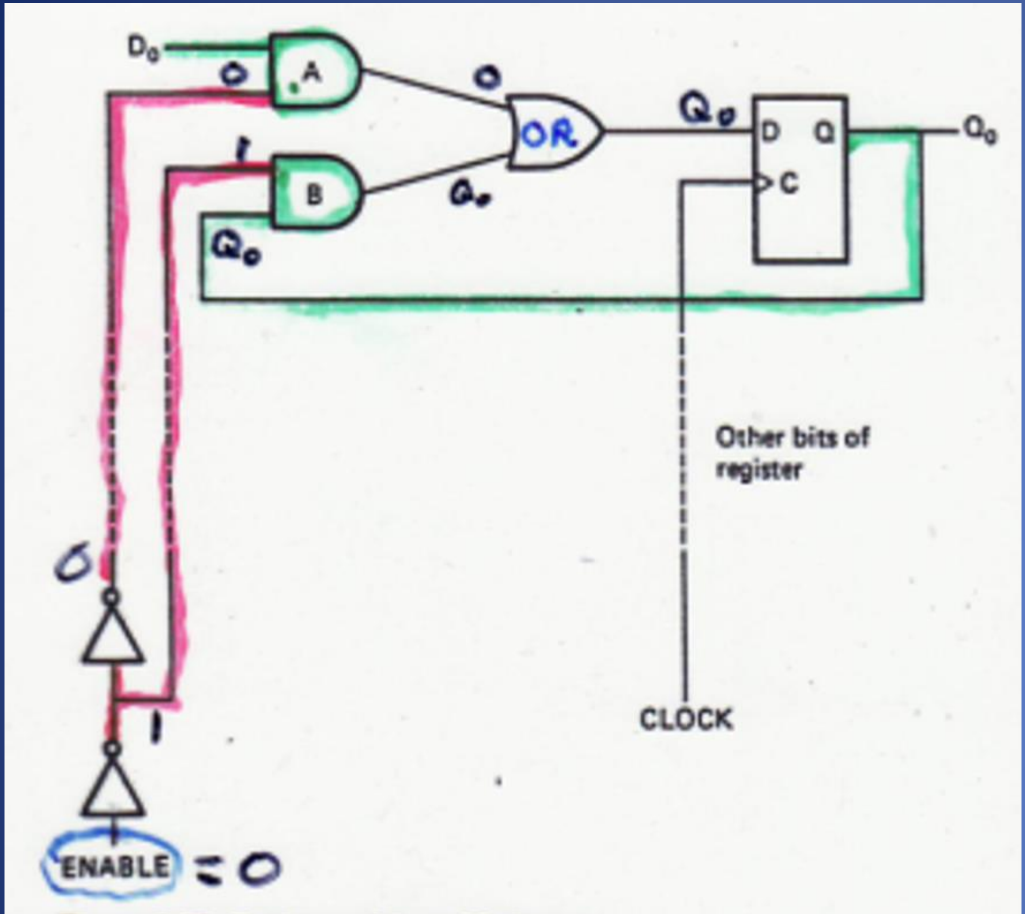


Figure 3-22 Register with enable inputs.

Enable = 0: register keeps old value.

Enable = 1: normal operation.

Acts like a digital camera with one pixel.

Each FF in the register has this circuit (multiple pixels!)

Can control Enable with ASM output.

What chip does the circuit in the dotted box act like?

A 2 to 1 MUX. The Enable is the select line, selects D_0 or Q_0.

Register with Enable

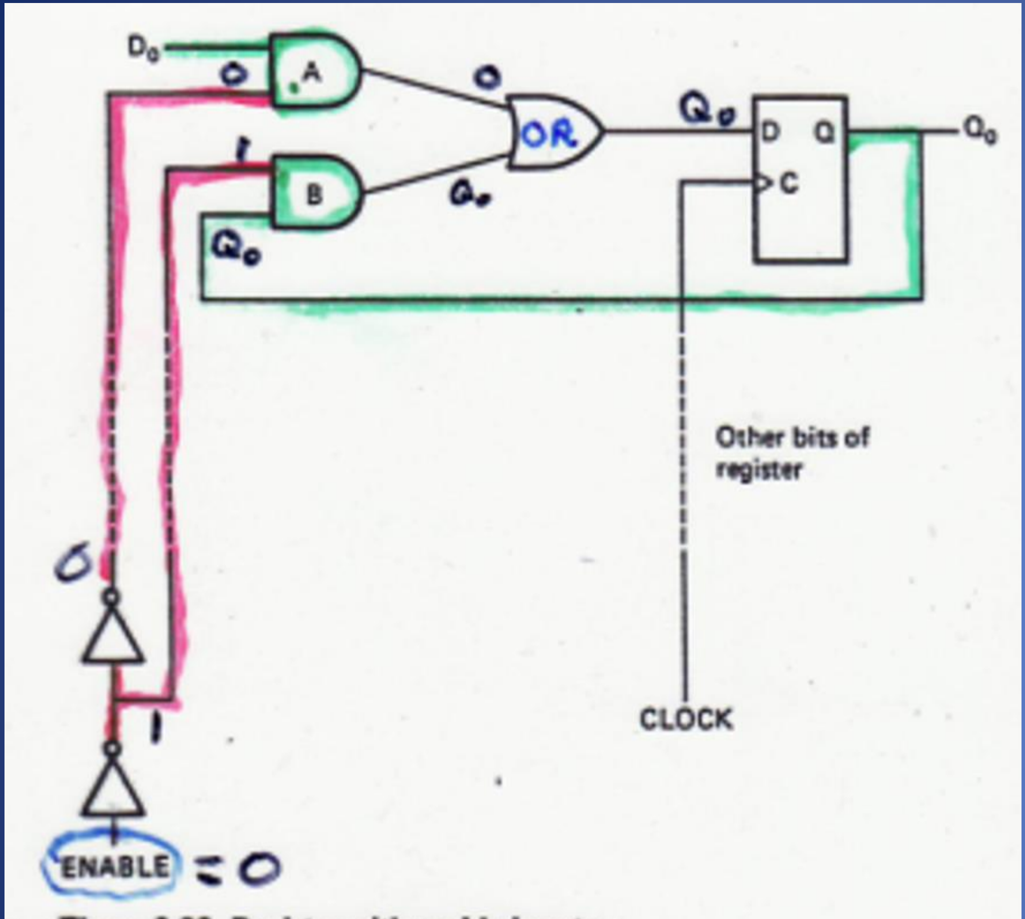


Figure 3-22 Register with enable inputs.

Enable = 0: register keeps old value.

Enable = 1: normal operation.

Can control Enable with ASM output.

Common uses in ASM:
1) Take "picture" of ASM inputs for MUX.
2) Expanding ROM outputs.
3) In FSMD (finite state machine with data path)

ASM Problem with One MUX and 2 inputs

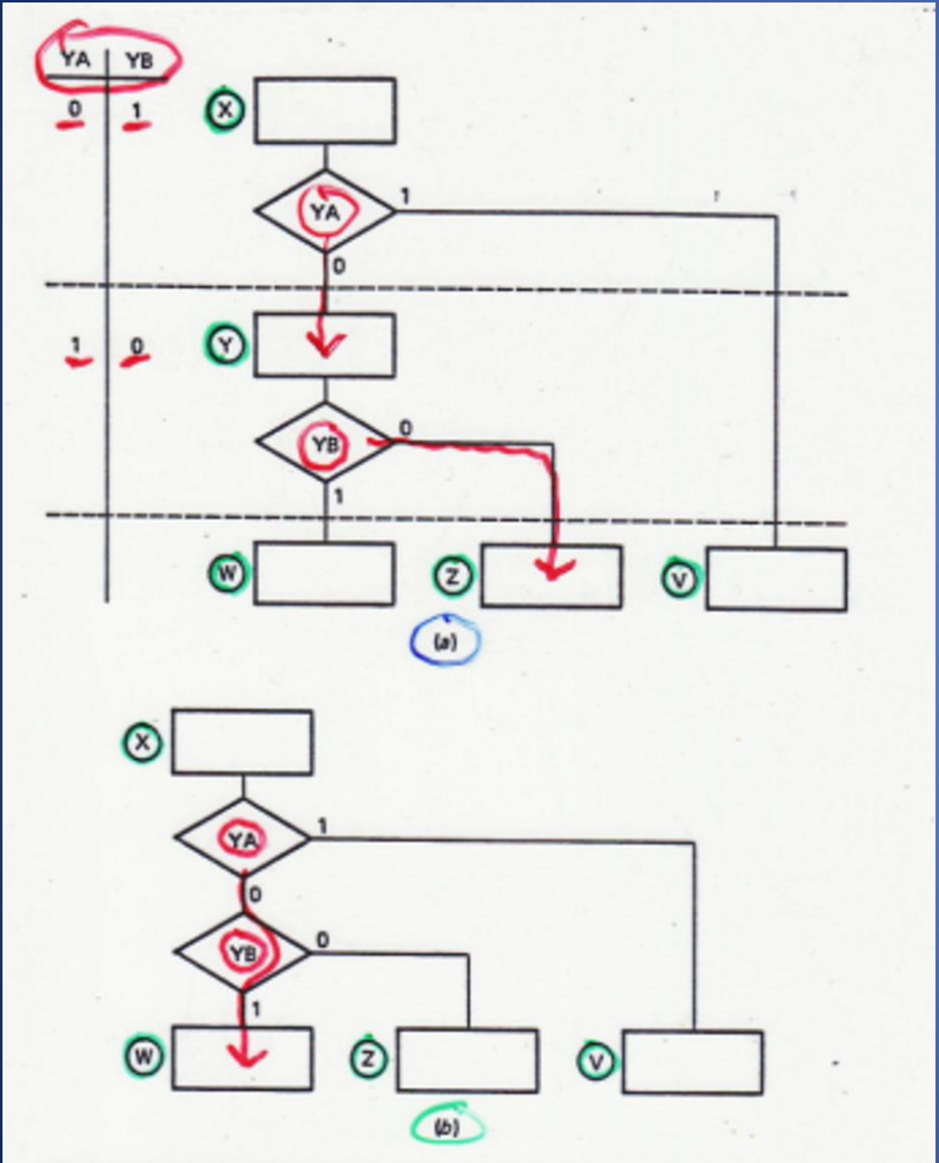


Figure 3-23 (a) Inputs changing during sequential testing. (b) Simultaneous testing.

Can only check one input at a time.

If the inputs change between states (which is likely), you won't be able to compare both inputs at the same time like you should.

a) Sequential testing goes to Z.
b) Inverted YA/YB goes to V.
c) Never goes to W.
d) Simultaneous testing goes to W.

Solution to ASM – MUX Problem

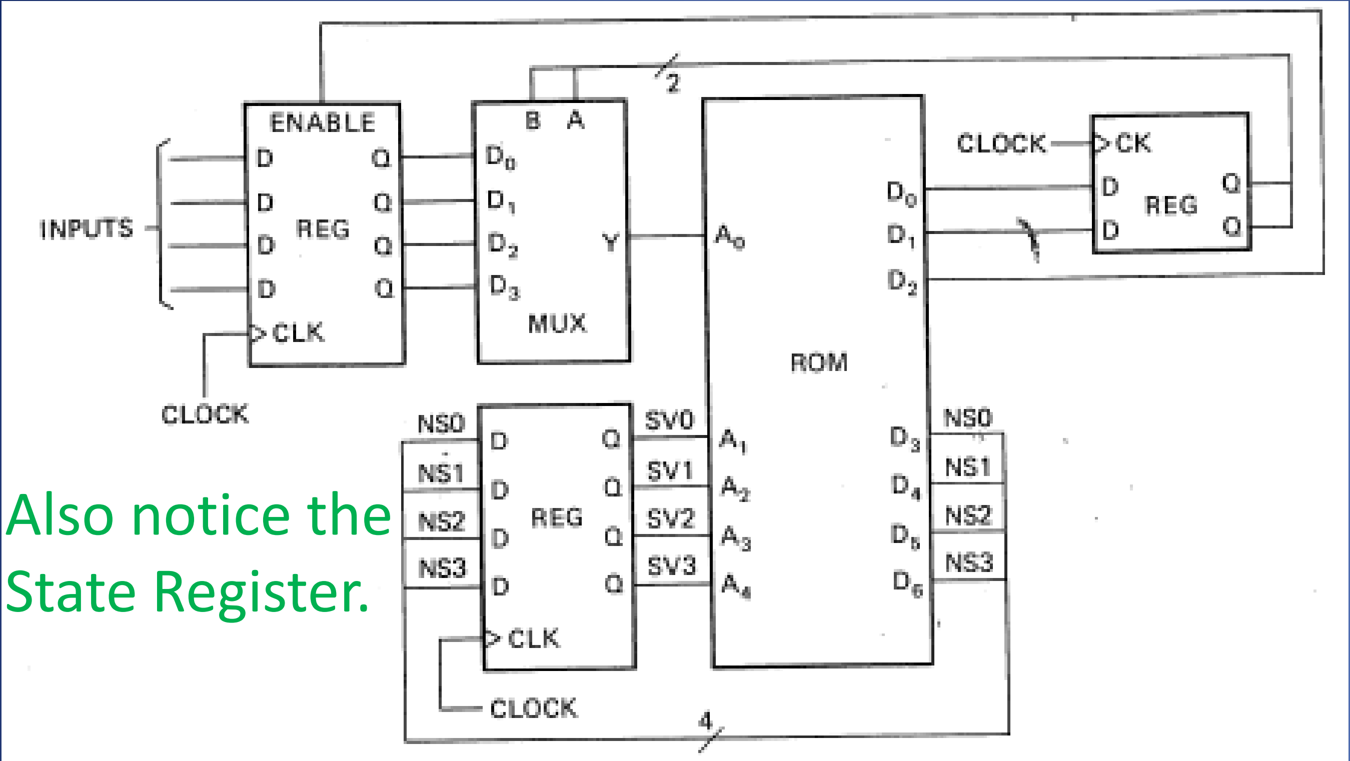


Figure 3-24 ASM using one MSI register to simultaneously save inputs and another MSI register for the state variables.

Also notice the State Register.

Take “snapshot” of inputs with ROM output D2.
Then cycle through them using the MUX in multiple ASM states.
ROM sees values relative to each other at same time.
However, takes more states, and misses input changes during cycling.

Expanding Outputs with Registers

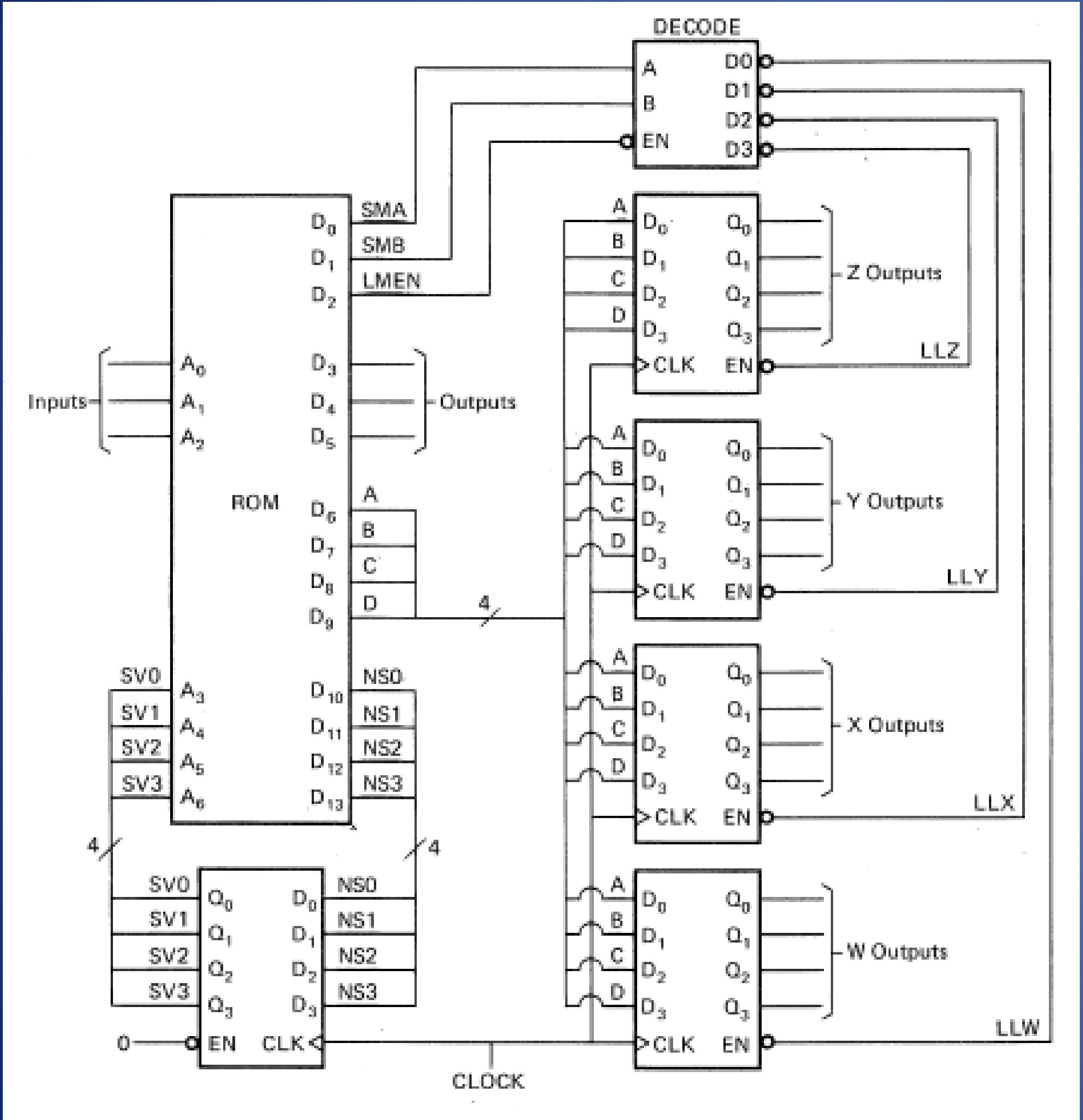


Figure 3-25 Using registers to expand ROM outputs.

ASM outputs can be stored in registers W, X, Y, Z.

[1] Can have many outputs using small ROM.

[2] Outputs are NOT mutually exclusive.

[3] Outputs remain on or off AFTER the ASM exits the state, and do not have to be asserted in every state.

Expanding Outputs with Registers

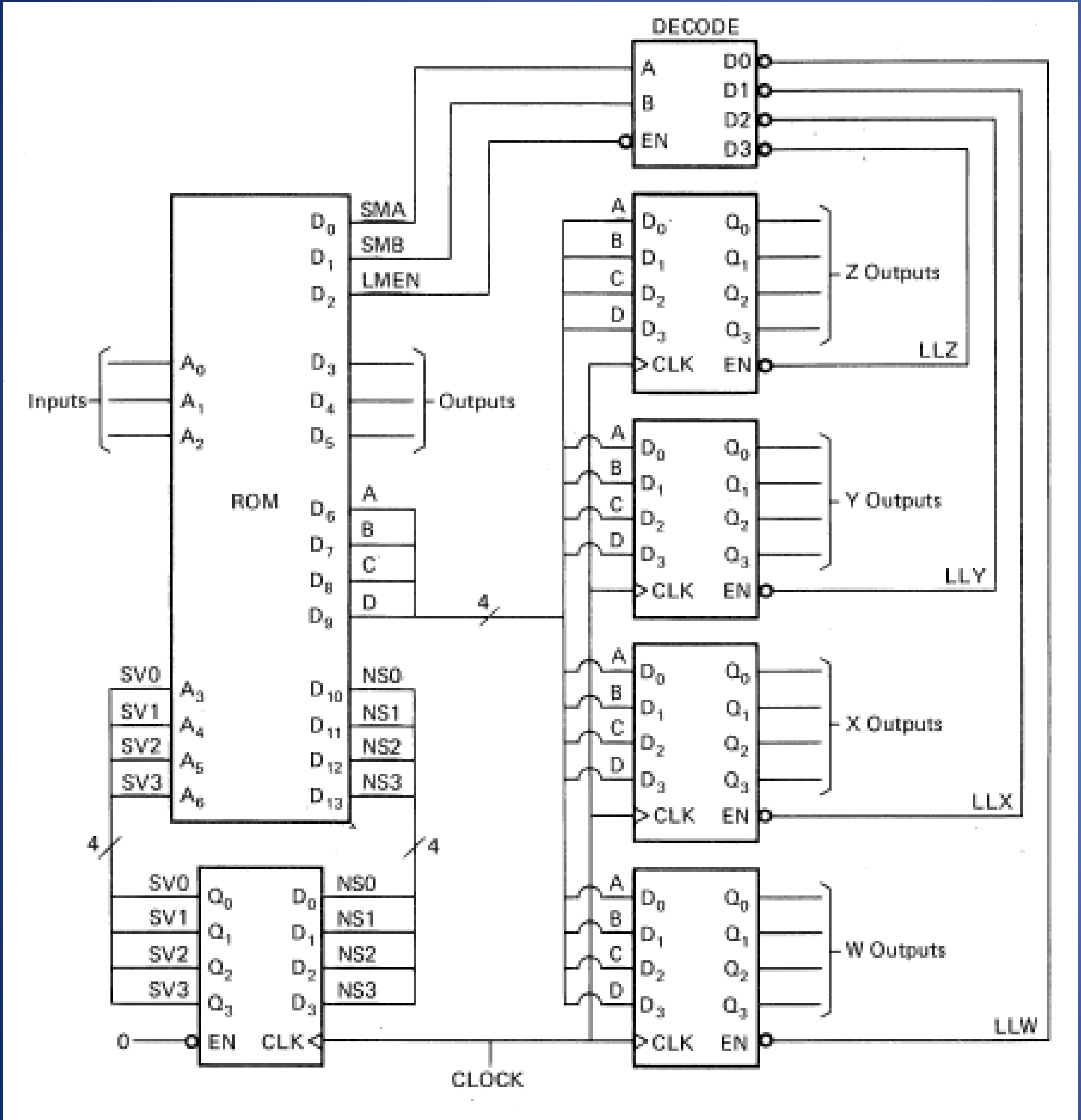


Figure 3-25 Using registers to expand ROM outputs.

[4] Disadvantage:
ROM output (DCBA) is "loaded" into ONE register at a time as selected by decoder (SMA,SMB). Takes 4 states. Can only change 4 outputs simultaneously at a time.

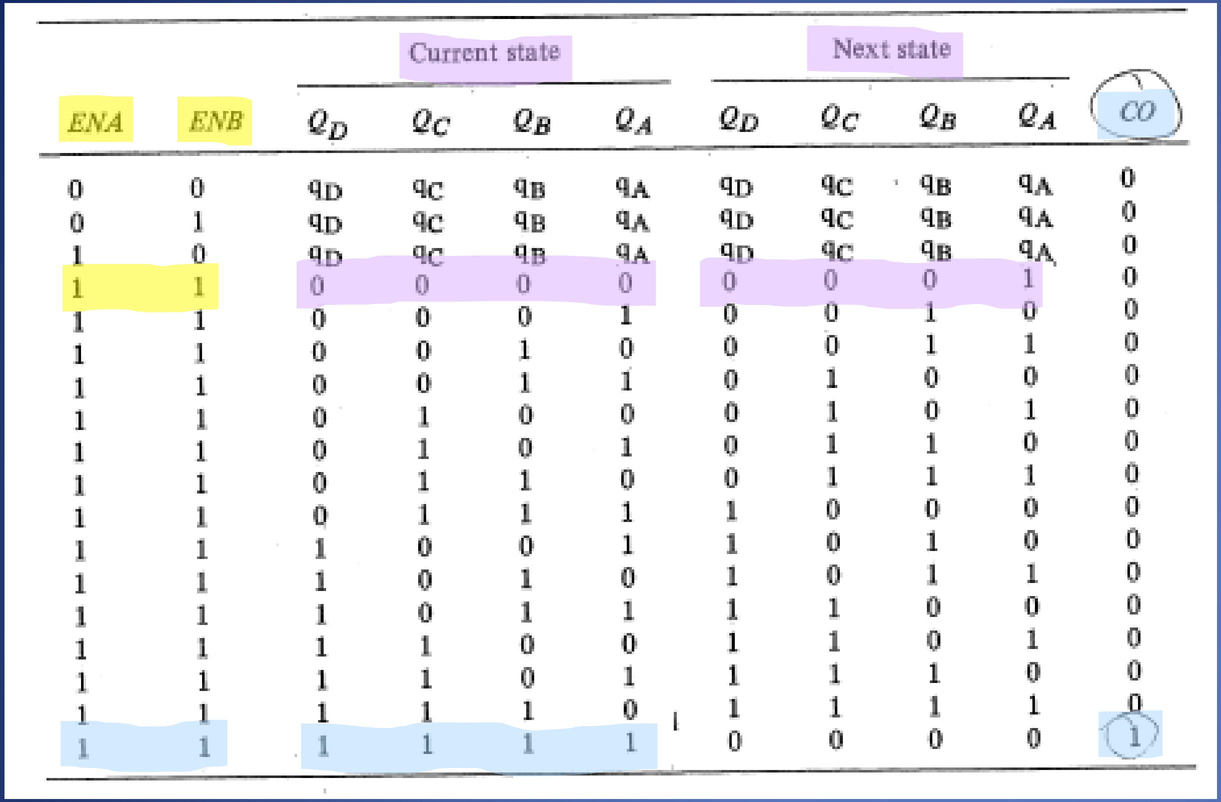
Why is LMEN used?

LMEN is used to disable all 4 register so they keep their values after they are loaded.

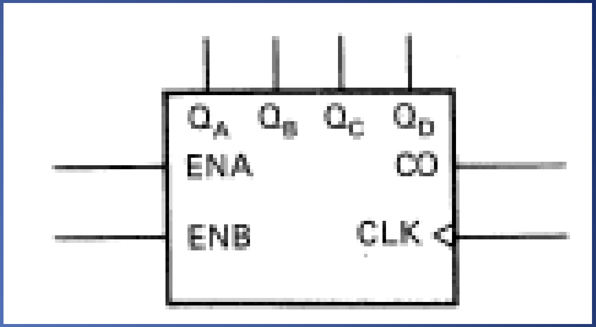
Is it required?

It is NOT required, but you would have to tie EN to GND, and keep reloading one of the registers every clock cycle by putting its values on the ABCD bus, and selecting it with SMA/SMB.

Counters



		Current state				Next state				
ENA	*ENB*	Q_D	Q_C	Q_B	Q_A	Q_D	Q_C	Q_B	Q_A	*CO*
0	0	q_D	q_C	q_B	q_A	q_D	q_C	q_B	q_A	0
0	1	q_D	q_C	q_B	q_A	q_D	q_C	q_B	q_A	0
1	0	q_D	q_C	q_B	q_A	q_D	q_C	q_B	q_A	0
1	1	0	0	0	0	0	0	0	1	0
1	1	0	0	0	1	0	0	1	0	0
1	1	0	0	1	0	0	0	1	1	0
1	1	0	0	1	1	0	1	0	0	0
1	1	0	1	0	0	0	1	0	1	0
1	1	0	1	0	1	0	1	1	0	0
1	1	0	1	1	0	0	1	1	1	0
1	1	0	1	1	1	1	0	0	0	0
1	1	1	0	0	1	1	0	1	0	0
1	1	1	0	1	0	1	0	1	1	0
1	1	1	0	1	1	1	1	0	0	0
1	1	1	1	0	0	1	1	0	1	0
1	1	1	1	0	1	1	1	1	0	0
1	1	1	1	1	0	1	1	1	1	0
1	1	1	1	1	1	0	0	0	0	1



Two Enable pins so can chain counters.

Important output: CO = carry out.

Counters are actually ASMs.

When BOTH enables are high, it counts.
Else count stays the same (stays in same state).

Connecting Counters

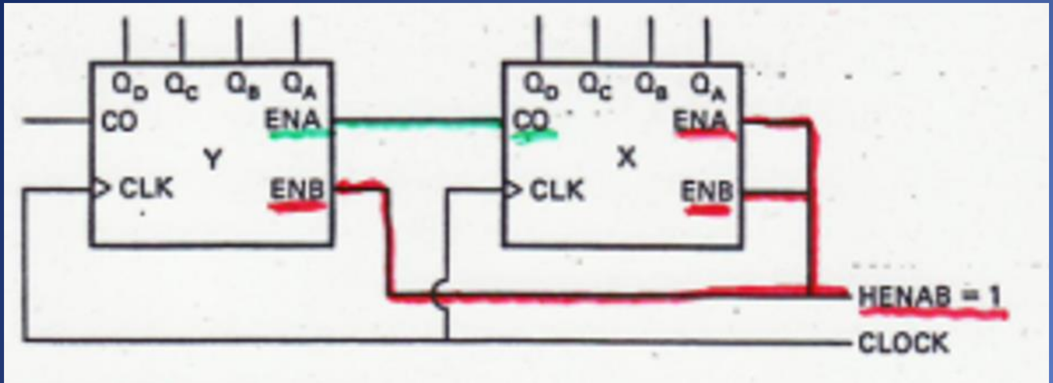


Figure 3-27 8-bit counter.

Use two 4-bit counters to make an 8-bit counter.

Counter X has both enables activated, so it counts from 0000 to 1111.

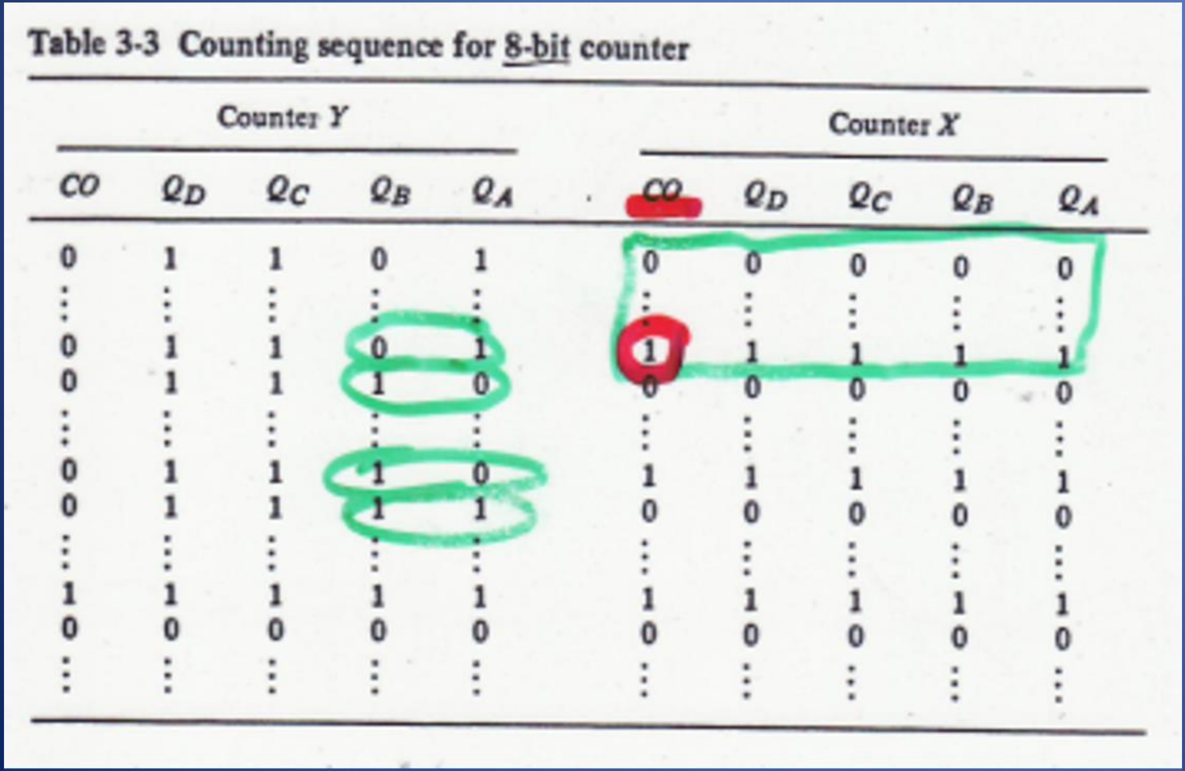


Table 3-3 Counting sequence for 8-bit counter

Counter Y					Counter X				
CO	Q_D	Q_C	Q_B	Q_A	CO	Q_D	Q_C	Q_B	Q_A
0	1	1	0	1	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	0	1	1	1	1	1	1
0	1	1	1	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	1	1	1	0	1	1	1	1	1
0	1	1	1	1	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

When X = 1111, CO = 1 which enables counter Y.

Counter Y counts the rollovers of counter X.

Lower enable line controls entire 8-bit counter.

Connecting Counters (contd)

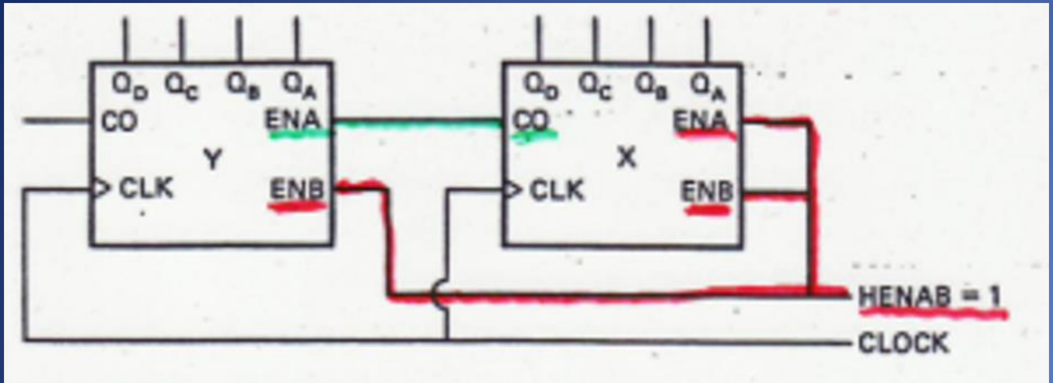


Figure 3-27 8-bit counter.

Counter Y counts the rollovers of counter X.

When X reaches 1111, CO = 1, enables counter Y for ONE count (because CO = 0 when X rolls over to 0000).

Grouping all 8 counter bits together yields this transition from 15 to 16: 00001111 goes to 00010000.

In table 3, counter transitions from 1101111 to 11100000

Counter Y’s CO goes high when count is 11111111.

Counter CLEAR and LOAD

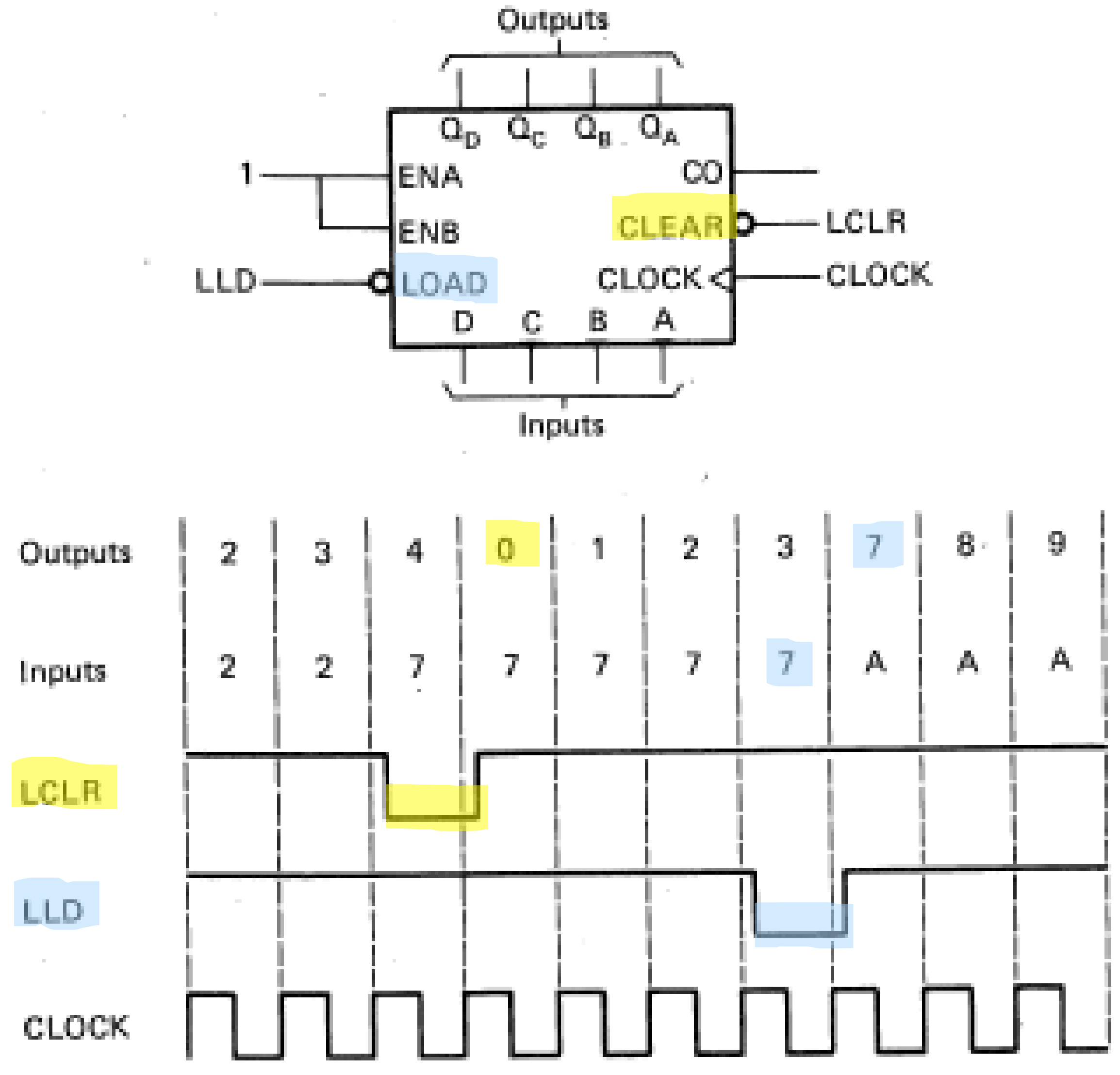


Figure 3-28 MSI counter with load and clear inputs.

Counter is counting, but gets overridden by CLEAR and LOAD.

Sync active-low CLEAR zeros the counter at next clock edge.

Sync active-low LOAD transfers DCBA inputs to counter outputs Q_D Q_C Q_B Q_A at next clock edge.
If LOAD is held low, counter acts like a register.

Counter used as Timer

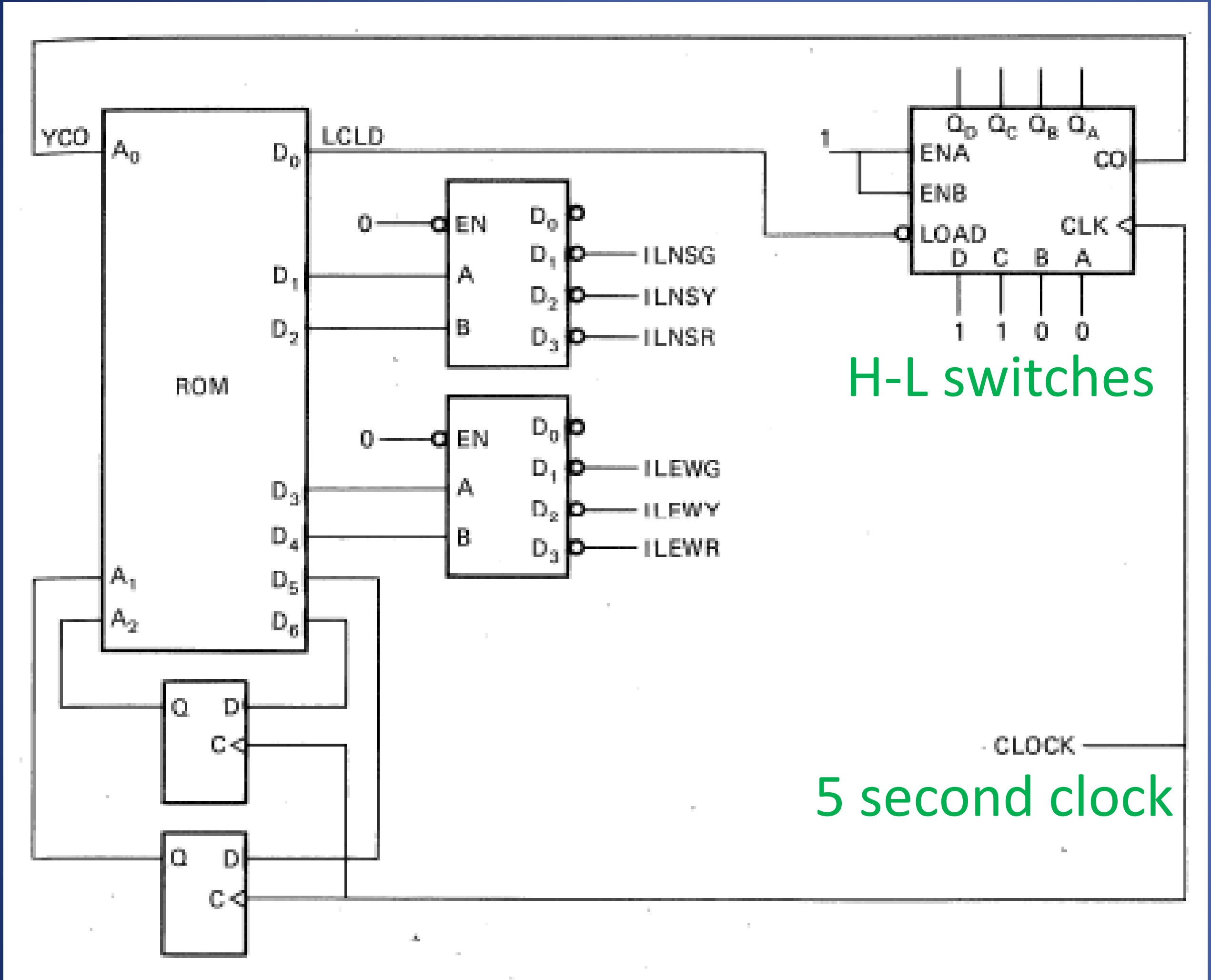


Figure 3-29 Traffic-light controller using counter for delay.

Why are LCLD and CO delayed outputs?

Use counter to time how long to stay in state.

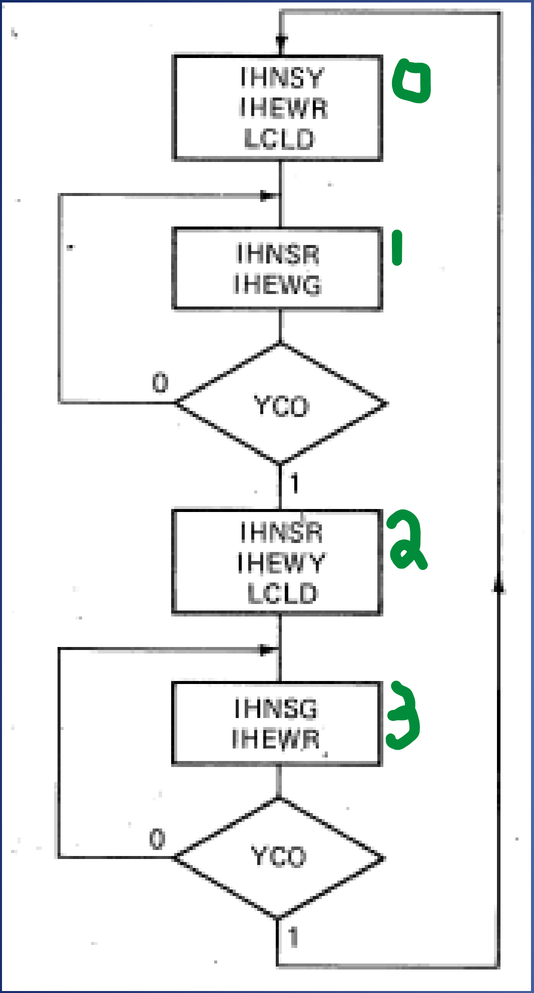
Connect load lines to switches to set delay.

Connect LOAD pin to ROM output. LCLD is a delayed output.

Connect CO pin to ROM input. CO is a delayed output.

Tie both counter enables high so it runs continuously.

Counter used as Timer cont’d



5 second clock

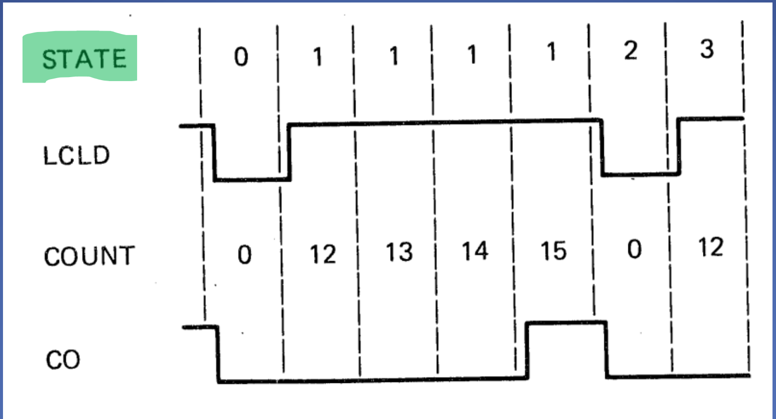
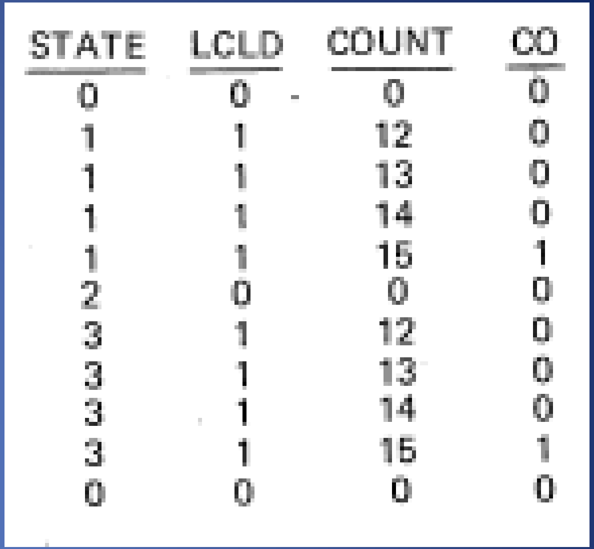


Figure 3-31 Traffic-light controller sequencing.



STATE	LCLD	COUNT	CO
0	0	0	0
1	1	12	0
1	1	13	0
1	1	14	0
1	1	15	1
2	0	0	0
3	1	12	0
3	1	13	0
3	1	14	0
3	1	15	1
0	0	0	0

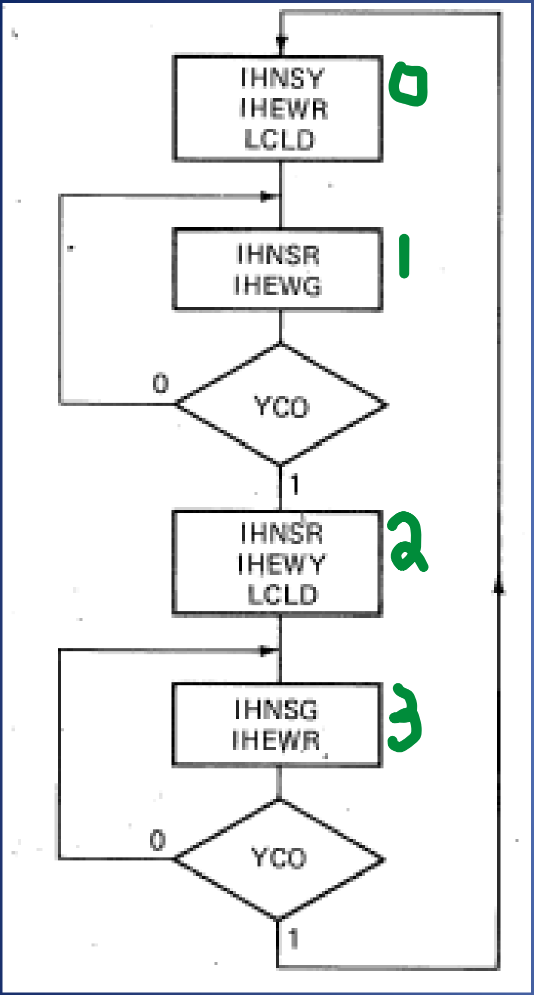
USE SWITCHES TO SET TIMER TO 20 S DELAY

The timer counts UP, so set switches in respect to how many counts needed to reach count 15, i.e., the CO times-up signal). Using a 5 s clock, so 4 counts x 5 s = 20 s for RED/GREEN state.

So switches = 12 gives 20 s delay (12 13 14 15).

Only 4 states needed with timer instead of 10 without.

Counter used as Timer cont’d



5 second clock

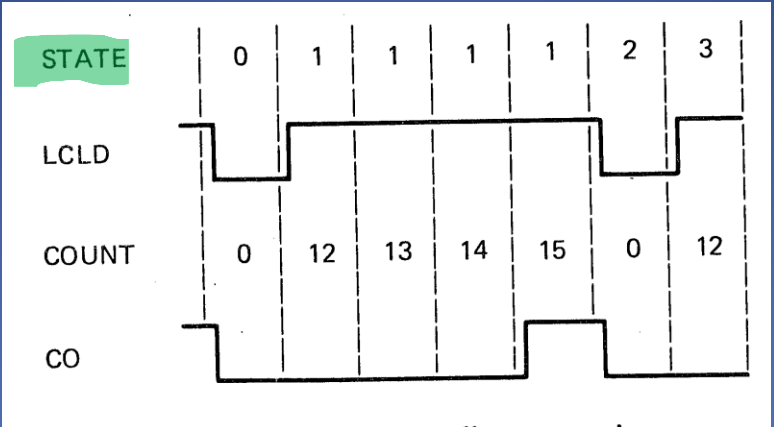
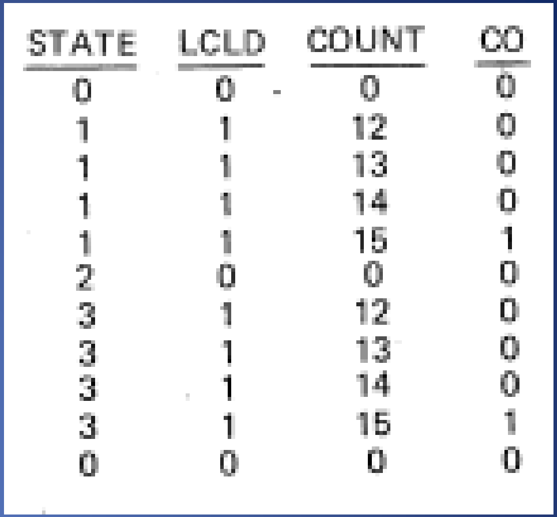


Figure 3-31 Traffic-light controller sequencing.



STATE	LCLD	COUNT	CO
0	0	0	0
1	1	12	0
1	1	13	0
1	1	14	0
1	1	15	1
2	0	0	0
3	1	12	0
3	1	13	0
3	1	14	0
3	1	15	1
0	0	0	0

Load timer in state 0 (NSY). LCLD is delayed output, so load happens as enter state 1 (NSR).

Stays in state 1 until YCO goes high at count 15. So counts 12, 13, 14, 15 (20 seconds). CO is delayed output; ASM goes to state 2 after count 15 is over.

Timer rolls over to count 0 as ASM enters state 2. (ASM and timer are both “running”).

Load timer in state 2 (EWY). Delayed output, so load happens as enter state 3 (NSG).

Using ROM outputs to set delay

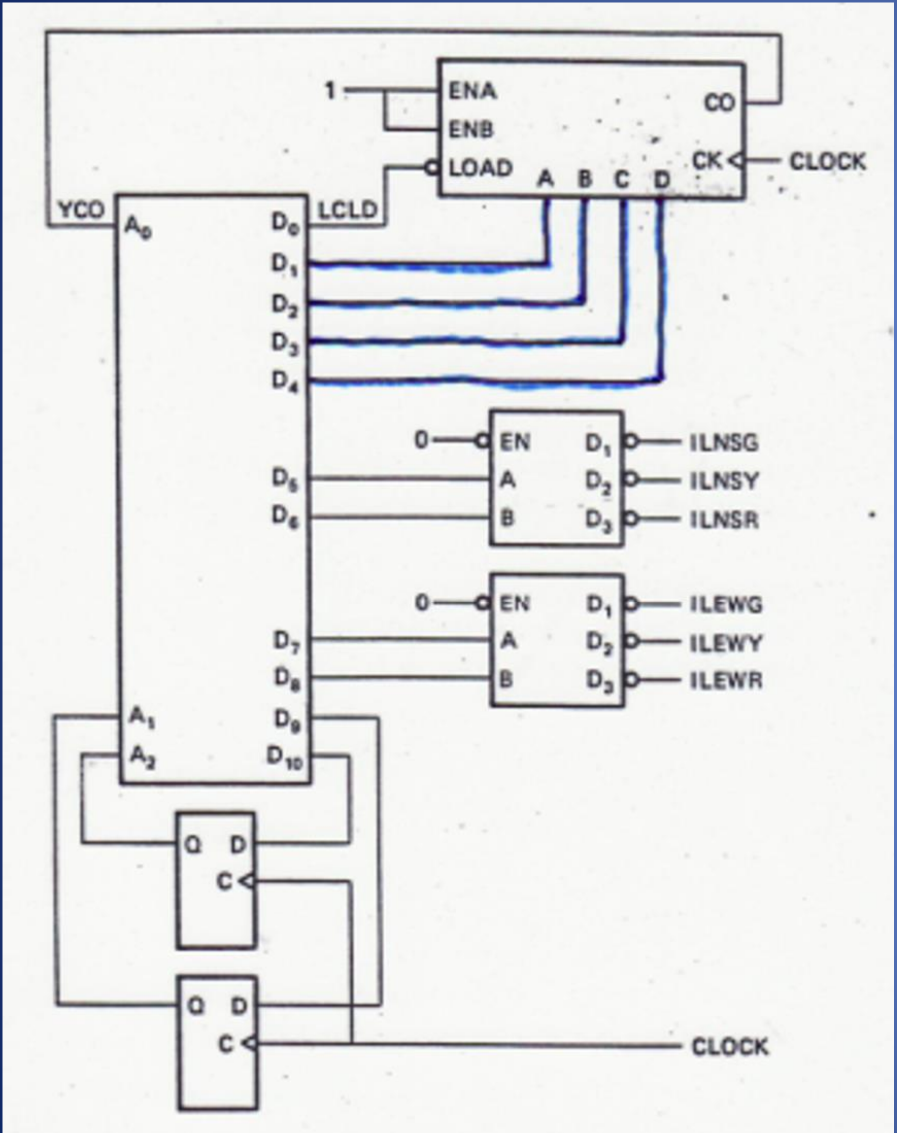


Figure 3-32 Using ROM outputs to set delay.

If need different delays for other states, use ROM outputs to set various delay times.

If only need two different delay times, and you should not be adding four outputs to the ROM, could use two timers with switches. Connect each timer's CO to a ROM input, and look at that input in the delayed state.

END