PHY 5430 Digital Systems

Wiatrowski Chapter 1

Digital Systems

Introduction:

Graduate class – Different than undergrad!

Transition to a company environment.

Independent. Teach yourself as much as possible.

Most of your time will be spent working on labs. Learn by doing approach.

Hands on.

Materials

"Logic Circuits and Microcomputer Systems" by C. Wiatrowski and C. House. (pdf).

VHDL tutorial: A programmed-Learning Approach for VHDL and Programmable Logic, in Digital Hardware Lab Manual by Michael D. Furman and Tyson S. Hall. (pdf)

"Rapid Prototyping of Digital Systems - SOPC Edition" book by James O. Hamblen, T.S. Hall and Michael D. Furman. Used by Georgia Tech, one of the top engineering schools in the USA.\

FPGA Prototyping by VHDL Examples and RTL Hardware Design Using VHDL by Chu (sections needed are posted on ASUlearn). **This is also your digital textbook. It is on ASUlearn under MATERIALS.**

Uses an FPGA board programmed with VHDL. Altera boards cost ~$500 and use expensive Quartus software. However, they have large educational discounts. This course uses real-world equipment

Materials (continued)

For about 1/3 of the course, we will be using the excellent Wiatrowski book from 1980 (now out of print). The first half of the book is very good and still relevant. Includes timing as well as covering state machines. The last half of the book is obsolete.

It is posted on ASUlearn.

Algorithmic State Machines

- Also called Finite State Machines
- Used as a model in LabView code
- The control circuit for CPUs
- Used in custom controllers (FSM)
- The control circuit for data manipulation (FSMD)

ASM

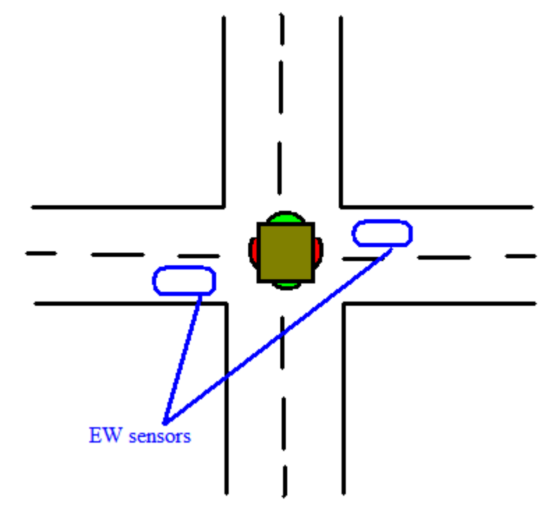
Algorithmic State Machine

Example of ASM:

Traffic Light Controller.

Inputs WT and ET.

Example: NS street is very busy.
EW street has little traffic.



Traffic intersection with East and West sensors.

Flow Chart

Flow Chart for the traffic light controller.

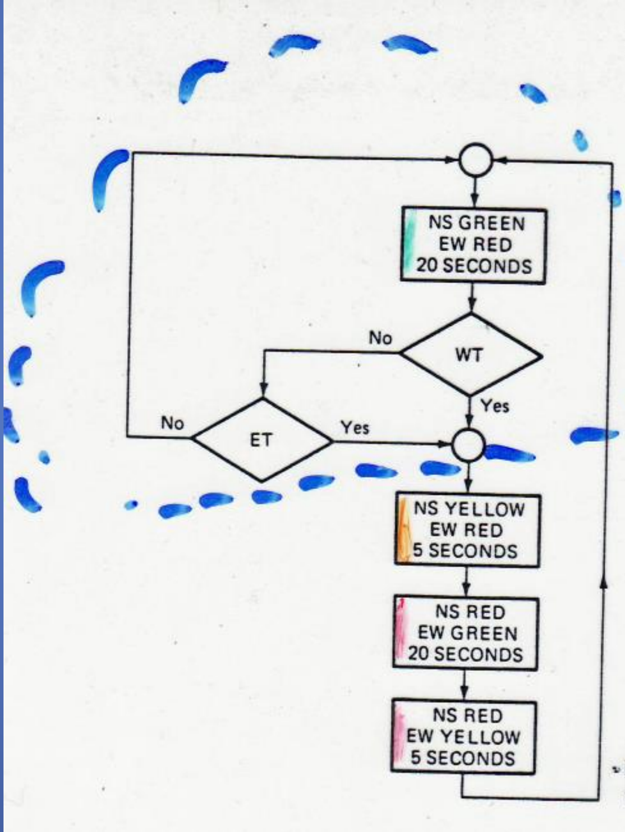


Figure 1-5 Flowchart for traffic intersection with traffic sensors.

Flow Chart (continued)

The states of the ASM are shown in boxes. (Each state has a time period.)

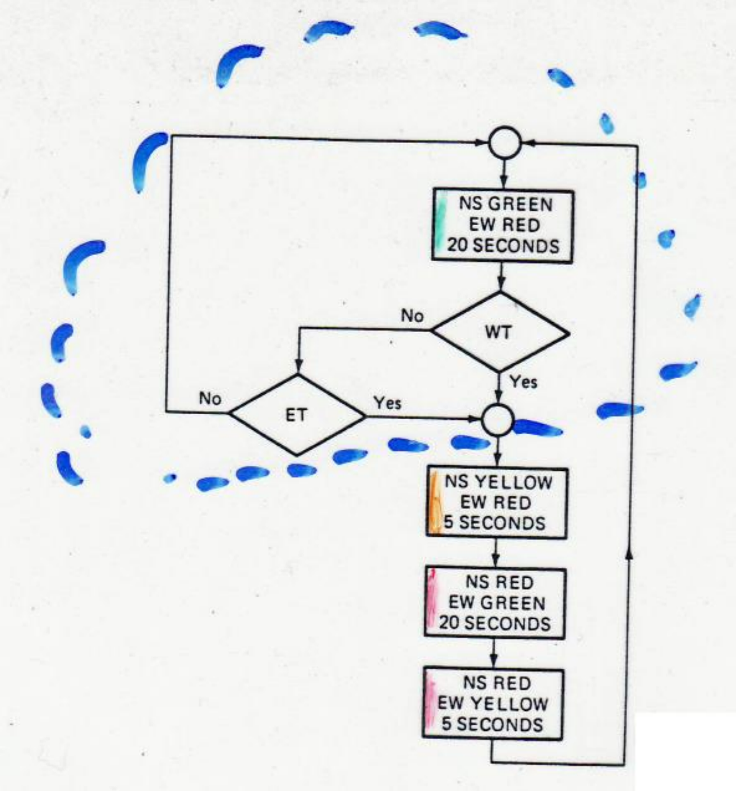
The diamonds indicate decisions based on the inputs. There is no time period associated with the diamonds.

In general, algorithms must have a finite number of steps (although they can repeat forever), precisely defined steps, definite order steps will be performed. Inputs and outputs are optional.

Flow Chart (continued)

SPECIAL NOTE: Everything in a state happens simultaneously!

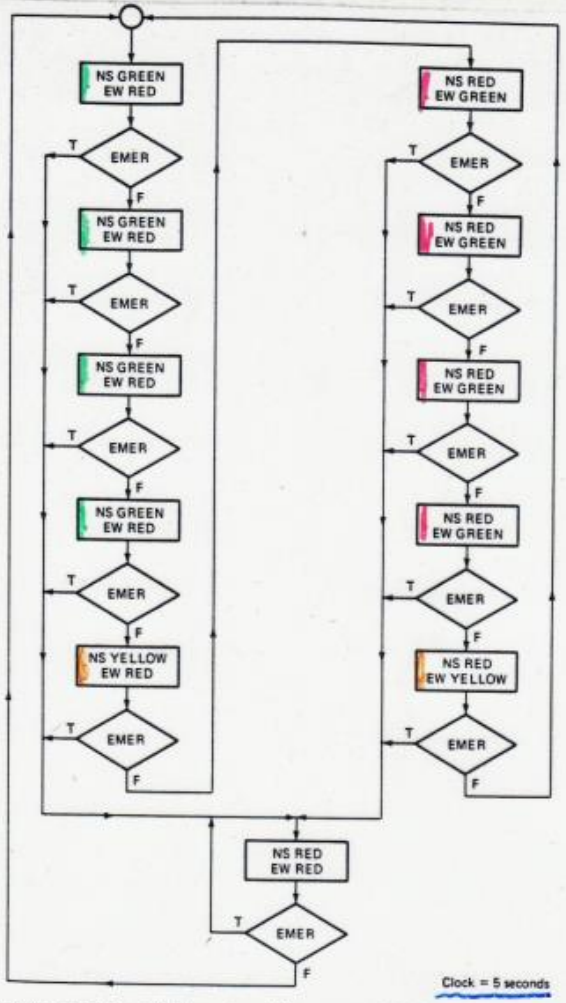
The dotted line is drawn around the state.



Flow Chart States

EVERY state must has SAME TIME PERIOD. Flow chart shows how different periods are implemented by repeating states.

Note: Slow clock.



Clock = 5 seconds

Figure 1-7 Flowchart for intersection with emergency vehicle input with a maximum delay of 5 s.

Timing Diagram

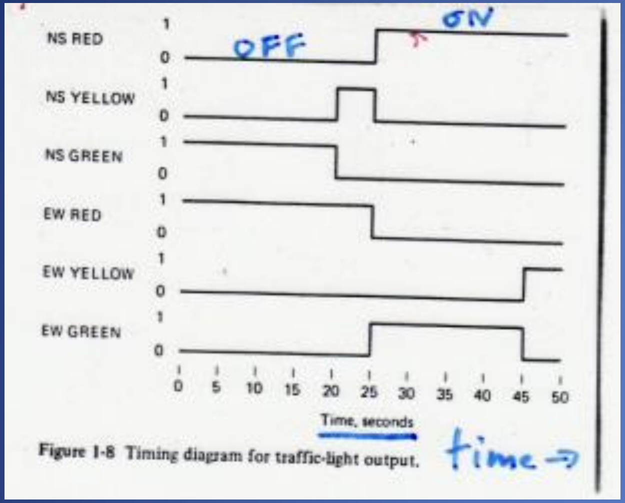


Figure 1-8 Timing diagram for traffic-light output.

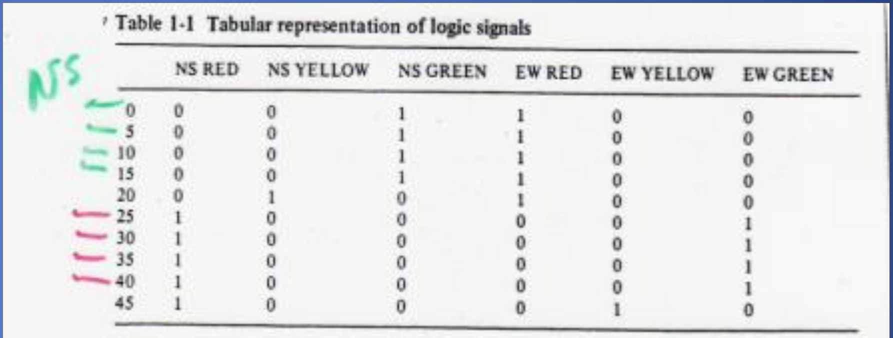


Table 1-1 Tabular representation of logic signals

NS

	NS RED	NS YELLOW	NS GREEN	EW RED	EW YELLOW	EW GREEN
0	0	0	1	1	0	0
5	0	0	1	1	0	0
10	0	0	1	1	0	0
15	0	0	1	1	0	0
20	0	1	0	1	0	0
25	1	0	0	0	0	1
30	1	0	0	0	0	1
35	1	0	0	0	0	1
40	1	0	0	0	0	1
45	1	0	0	0	1	0

Plots the signals controlling the lights.

Define: Digital 1 will turn light on, 0 off.

You can view timing diagram using oscilloscope or a logic analyzer. Also shows glitches.

If signals only change at constant time intervals, then can use logic signal table shown above right. This is convenient for large and complex systems. Can view table for a real operating circuit using a logic state analyzer. **We will use the SIGNAL TAP logic analyzer in Quartus.**

Algorithms

All algorithms can be implemented using only ROM's (Read Only Memories) and D Flip-flops.

Alternatively, ROM can be replaced with gate circuit.

For complex algorithms, the use of additional chips (timers, counter, decoders, multiplexers, registers) may be more economical and efficient .

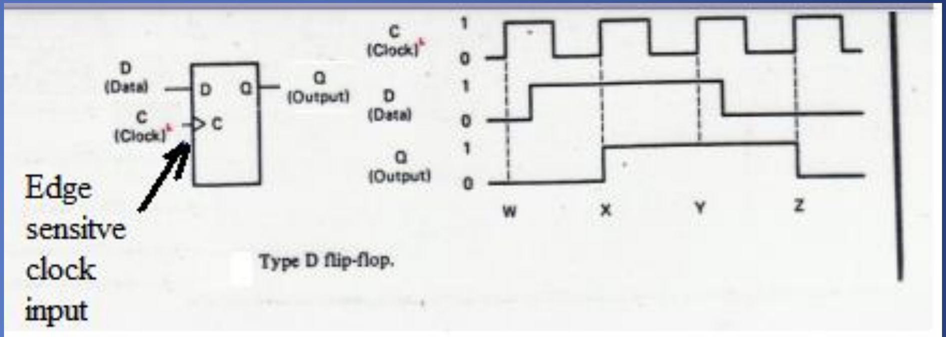
D Flip-Flops

Simple D Flip-flop (memory circuit)

C is the clock (edge sensitive)

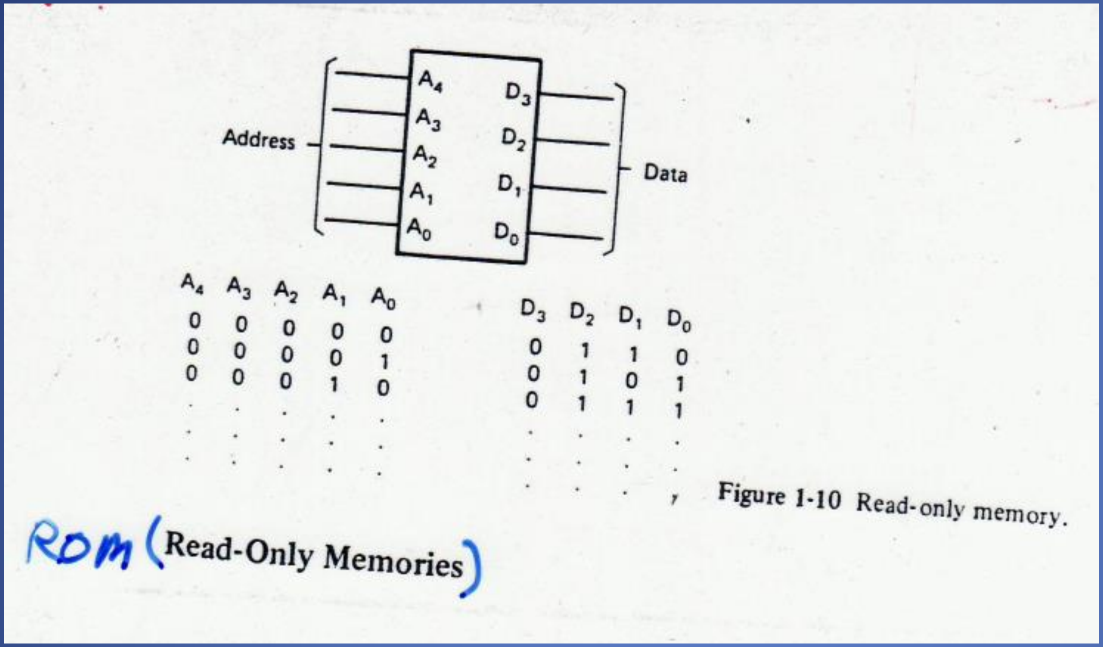
D is the Data input

Q is the output



The Q output remembers the state (0 or 1) of the D input that existed when the last 0-to-1 transition occurred on the clock line input.

Memory



A_4	A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	1	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

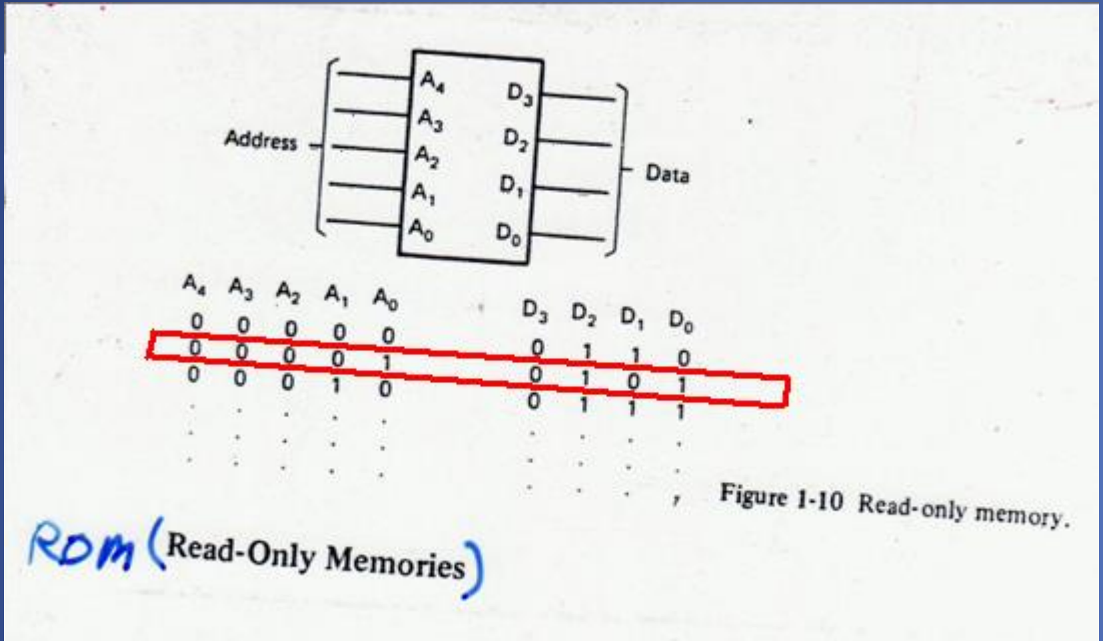
Figure 1-10 Read-only memory.

ROM (Read-Only Memories)

This ROM has 5 address lines and 4 data outputs. The number of memory locations would be $2^5 = 32$. Each of these memory locations has 4 bits of data.

If you put 00001 on the address lines, what is the output???

Memory



A_4	A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1	1	0
0	0	0	0	1	0	1	0	1
0	0	0	1	0	0	1	1	1
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Figure 1-10 Read-only memory.

ROM (Read-Only Memories)

This ROM has 5 address lines and 4 data outputs. The number of memory locations would be $2^5 = 32$. Each of these memory locations has 4 bits of data.

If you put 00001 on the address lines, what is the output???

0101

Memory (continued)

ROM

PROM

EPROM

EEPROM

Flash

Memory (continued)

ROM

PROM

EPROM   ----------- Numberd 27xx…

EEPROM -----\ __ Both numbered 28xx…

Flash ----------/

So a 2716 is an EPROM 2k memory locations each with 8 data bits.

A 2864 is an 8k by 8 EEPROM.

Memory (continued)

ROM's are memory chips which remember a pattern of data that is built into the chip permanently by the manufacturer. The 1st chip is VERY expensive , the rest are cheap. It can not be modified in the field.

PROM (Programmable ROM) can be set ONCE by the designer. Cheaper, but still can not be modified in the field.

EPROM (Erasable PROM) is a PROM which can be erased by UV light, then reprogrammed.

EEPROM (Electrically Erasable PROM)

Flash memory (entire device erased quickly)

Controller

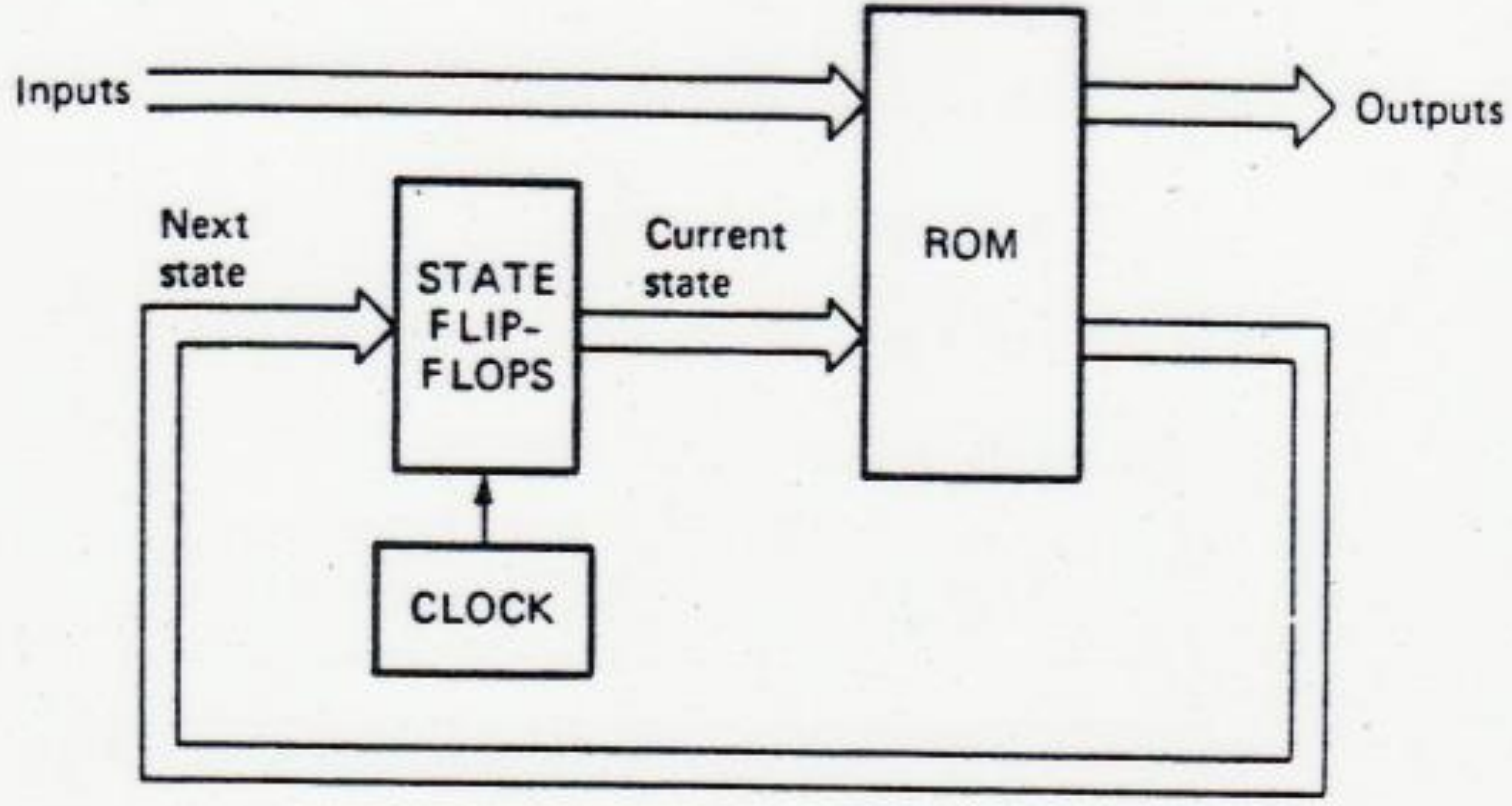


Figure 1-17 Sequential digital machine.

Controller in an example system

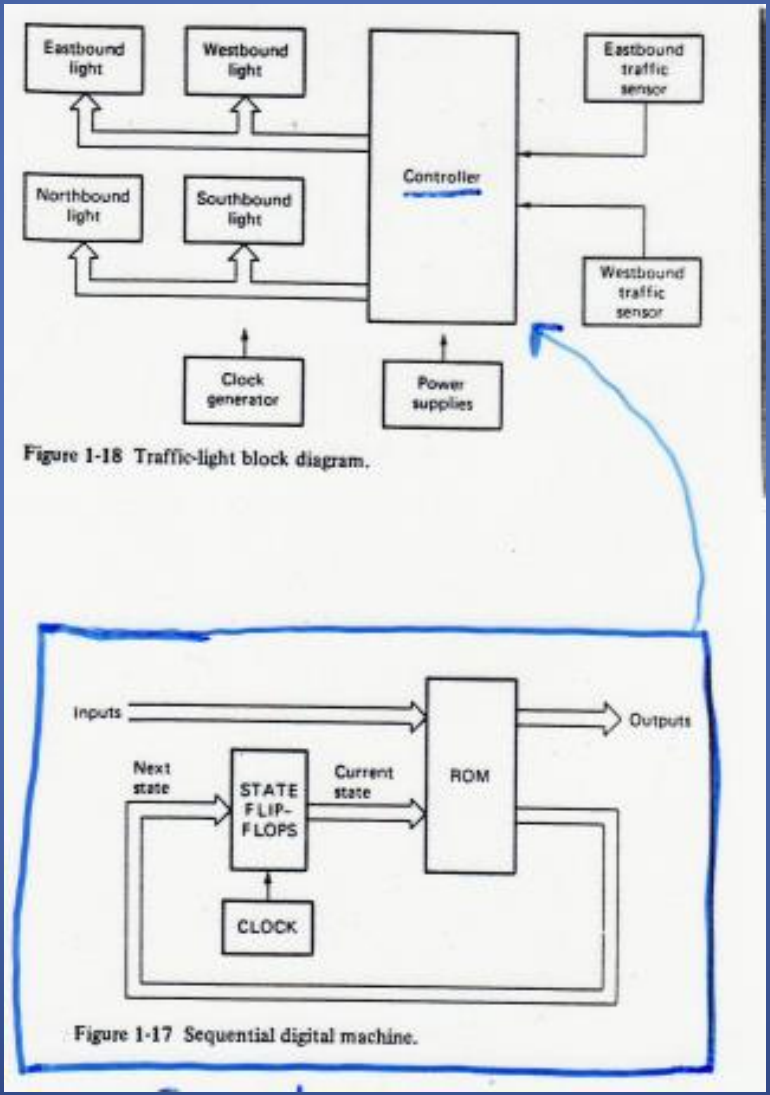
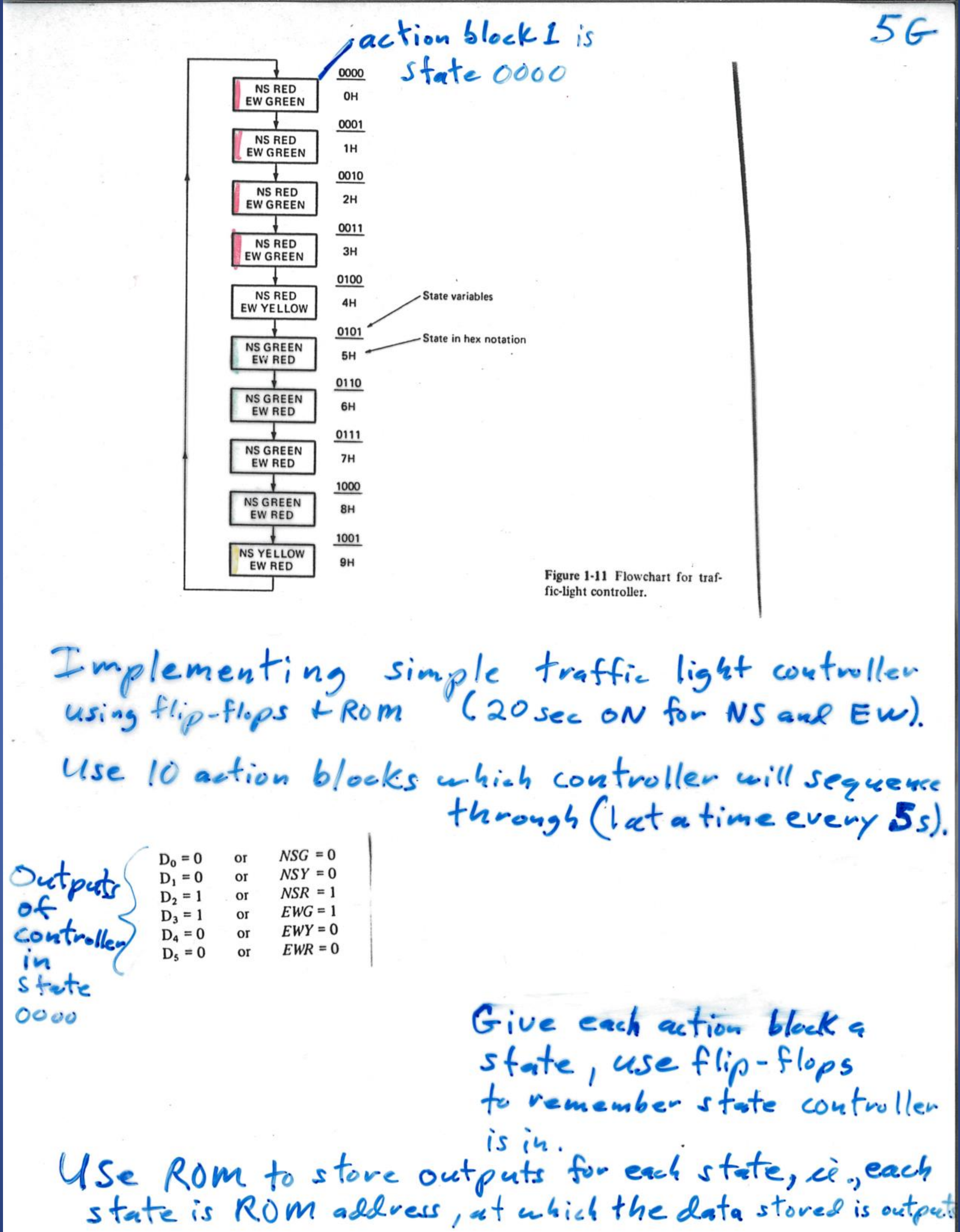


Figure 1-18 Traffic-light block diagram.

Figure 1-17 Sequential digital machine.



5G

action block 1 is state 0000

NS RED EW GREEN
0000
0H
NS RED EW GREEN
0001
1H
NS RED EW GREEN
0010
2H
NS RED EW GREEN
0011
3H
NS RED EW YELLOW
0100
4H
NS GREEN EW RED
0101
5H
NS GREEN EW RED
0110
6H
NS GREEN EW RED
0111
7H
NS GREEN EW RED
1000
8H
NS YELLOW EW RED
1001
9H
State variables
State in hex notation

Figure 1-11 Flowchart for traffic-light controller.

Implementing simple traffic light controller using flip-flops + ROM (20 sec ON for NS and EW).

Use 10 action blocks which controller will sequence through (1 at a time every 5s).

Outputs of controller in state 0000

$D_0 = 0$	or	$NSG = 0$
$D_1 = 0$	or	$NSY = 0$
$D_2 = 1$	or	$NSR = 1$
$D_3 = 1$	or	$EWG = 1$
$D_4 = 0$	or	$EWY = 0$
$D_5 = 0$	or	$EWR = 0$

Give each action block a state, use flip-flops to remember state controller is in.

Use ROM to store outputs for each state, i.e., each state is ROM address, at which the data stored is outputs

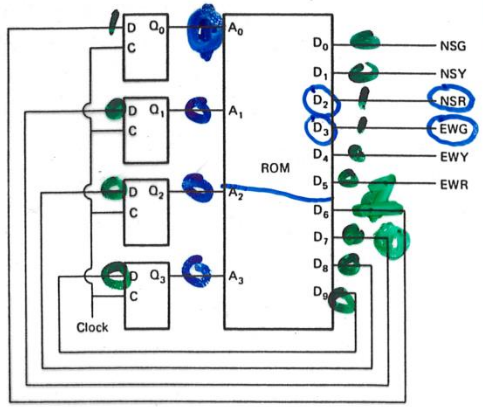


Figure 1-14 Traffic controller.

Table 1-2 Output signal definitions (D0 – D5) 5I

Signal abbreviation	If signal is 0	If signal is 1
NSG	NS green is off	NS green is on
NSY	NS yellow is off	NS yellow is on
NSR	NS red is off	NS red is on
EWG	EW green is off	EW green is on
EWY	EW yellow is off	EW yellow is on
EWR	EW red is off	EW red is on

$A_3 A_2 A_1 A_0$

EWG ON

NSR ON

Table 1-4 Traffic controller ROM contents

Current state ROM address	Next state D_9	D_8	D_7	D_6	Outputs D_5	D_4	D_3	D_2	D_1	D_0	ROM contents
0H	0	0	0	1	0	0	1	1	0	0	04CH
1H	0	0	1	0	0	0	1	1	0	0	08CH
2H	0	0	1	1	0	0	1	1	0	0	0CCH
3H	0	1	0	0	0	0	1	1	0	0	10CH
4H	0	1	0	1	0	1	0	1	0	0	154H
5H	0	1	1	0	1	0	0	0	0	1	1A1H
6H	0	1	1	1	1	0	0	0	0	1	1E1H
7H	1	0	0	0	1	0	0	0	0	1	221H
8H	1	0	0	1	1	0	0	0	0	1	261H
9H	0	0	0	0	1	0	0	0	1	0	022H
AH	0	0	0	0	0	0	0	0	0	0	000H
BH	0	0	0	0	0	0	0	0	0	0	000H
CH	0	0	0	0	0	0	0	0	0	0	000H
DH	0	0	0	0	0	0	0	0	0	0	000H
EH	0	0	0	0	0	0	0	0	0	0	000H
FH	0	0	0	0	0	0	0	0	0	0	000H

01H

Not used

We know order of states from flow chart shown previously. Store next state in Rom memory (D6-D9). Output from these ROM bits is connected to flip flop inputs.

SUMMARY: Current state register (the flip-flops) is sequenced to new state every 5 seconds by clock sig. ROM has outputs for each state, also has next state to occur.

Gate Implementation of Traffic Controller

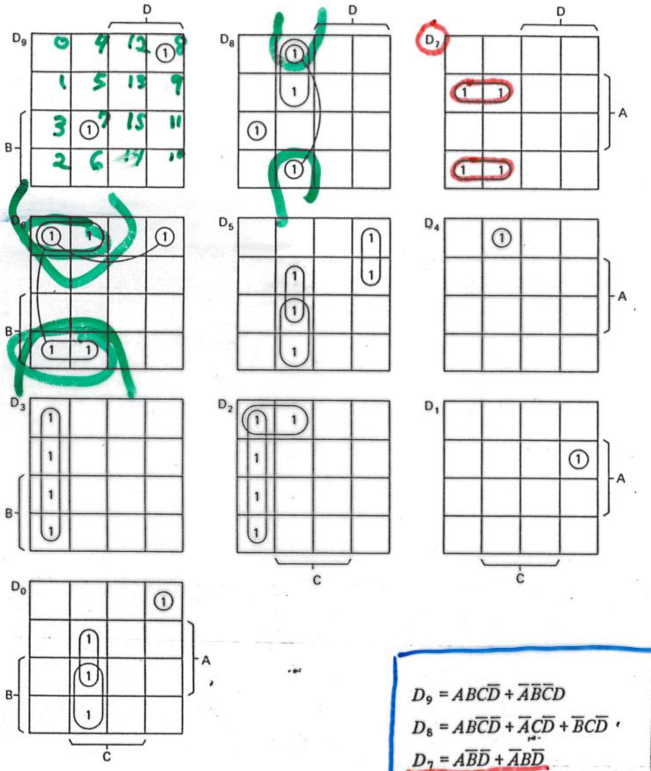


Figure 2-16 Reduced functions for traffic controller.

$D_9 = ABC\overline{D} + \overline{A}\overline{B}\overline{C}D$

$D_8 = AB\overline{C}\overline{D} + \overline{A}C\overline{D} + \overline{B}C\overline{D}$

$D_7 = A\overline{B}\overline{D} + \overline{A}B\overline{D}$

$D_6 = \overline{A}\overline{D} + \overline{A}B\overline{C}$

$D_5 = BC\overline{D} + AC\overline{D} + \overline{B}\overline{C}D$

$D_4 = \overline{A}\overline{B}C\overline{D}$

$D_3 = \overline{C}\overline{D}$

$D_2 = \overline{C}\overline{D} + \overline{A}\overline{B}\overline{D}$

$D_1 = A\overline{B}\overline{C}D$

$D_0 = BC\overline{D} + AC\overline{D} + \overline{A}\overline{B}\overline{C}D$

SOP functions from above KM's.

KM's from ROM table 2-1.

Note: We worked through output D7 already.

SOP functions describe operation of the ROM.

Now we need to replace ROM with GATES • • •

6V

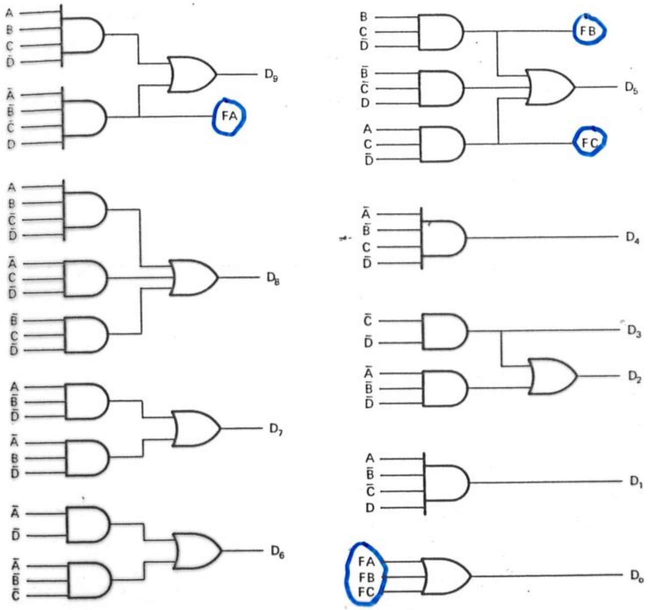


Figure 2-17 Gate realizations of logic expressions for traffic light.

Each output expression (D0-D9) has been implemented with gates.

Notice:

SOP → AND to OR

FA, FB, FC → simplifies schematic

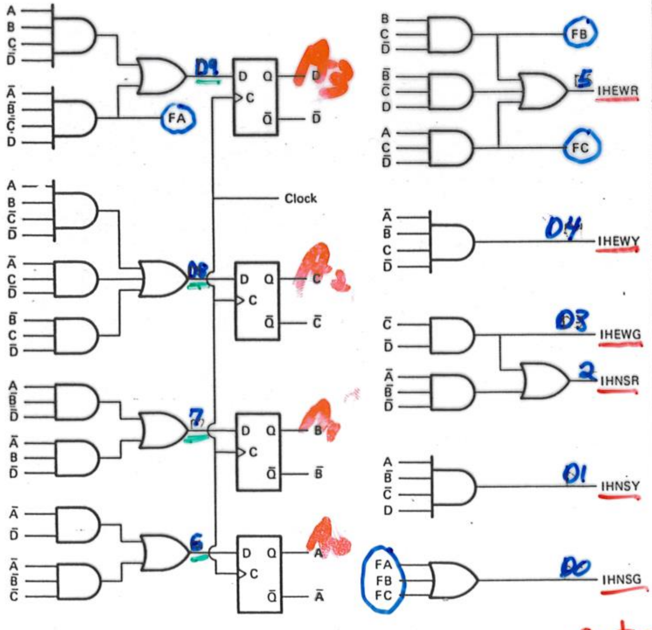


Figure 2-18 Complete traffic-light controller using gates.

Outputs

Above circuits are now combined with flip-flops to form controller.

D0 - D5 outputs

D6 - D9 next state

Compare to ROM controller, Fig. 1-14.

5 J

Another EXAMPLE

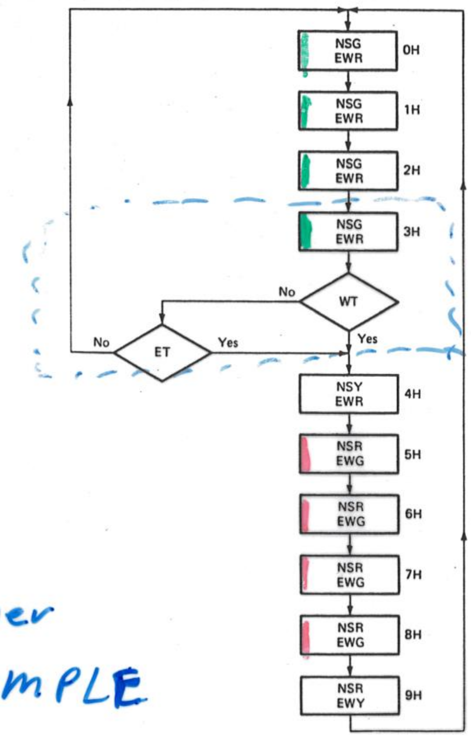


Figure 1-15 Flowchart of traffic-light controller with traffic sensor inputs.

Implementing a traffic controller with inputs.

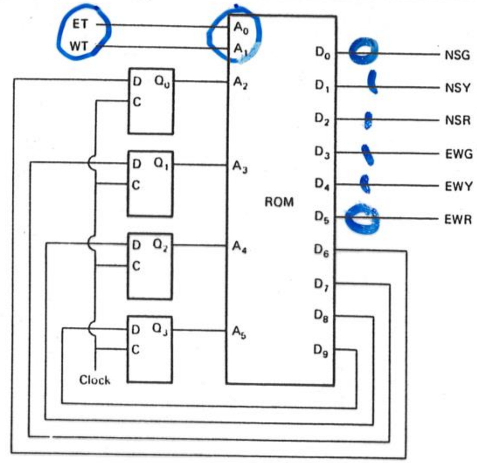
WT, ET are the traffic sensors.

Flowchart shown earlier (Fig. 1-5) is modified so action blocks are equal time (clock signal can now be used to sequence a controller).

Construct controller as before.

Problem. Next state after 3H is either state 0H or 4H, depending on inputs.

5K



• **Figure 1-16** Traffic-light controller with traffic sensor inputs.

going are CS

Table 1-6 Abbreviated ROM contents

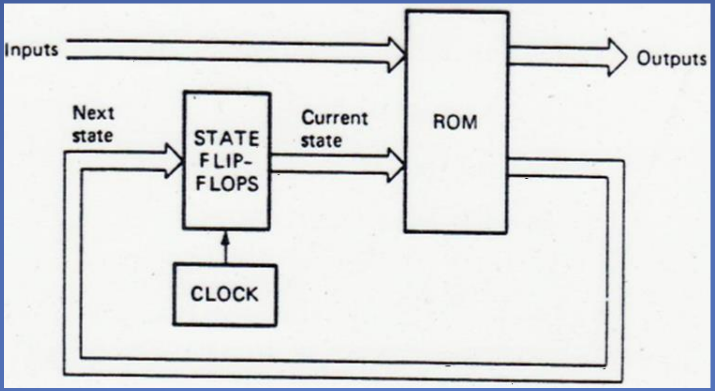
	Current State				WT	ET	Next state				Outputs					
	A_5	A_4	A_3	A_2	A_1	A_0	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0H	0	0	0	0	×	×	0	0	0	1	1	0	0	0	0	1
1H	0	0	0	1	×	×	0	0	1	0	1	0	0	0	0	1
2H	0	0	1	0	×	×	0	0	1	1	1	0	0	0	0	1
3H	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	1
	0	0	1	1	0	1	0	1	0	0	1	0	0	0	0	1
	0	0	1	1	1	0	0	1	0	0	1	0	0	0	0	1
	0	0	1	1	1	1	0	1	0	0	1	0	0	0	0	1
4H	0	1	0	0	×	×	0	1	0	1	1	0	0	0	1	0
5H	0	1	0	1	×	×	0	1	1	0	0	0	1	1	0	0
6H	0	1	1	0	×	×	0	1	1	1	0	0	1	1	0	0
7H	0	1	1	1	×	×	1	0	0	0	0	0	1	1	0	0
8H	1	0	0	0	×	×	1	0	0	1	0	0	1	1	0	0
9H	1	0	0	1	×	×	0	0	0	0	0	1	0	1	0	0
Not used	1	0	1	0	×	×	0	0	0	0	0	0	0	0	0	0
	1	0	1	1	×	×	0	0	0	0	0	0	0	0	0	0
	1	1	0	0	×	×	0	0	0	0	0	0	0	0	0	0
	1	1	0	1	×	×	0	0	0	0	0	0	0	0	0	0
	1	1	1	0	×	×	0	0	0	0	0	0	0	0	0	0
	1	1	1	1	×	×	0	0	0	0	0	0	0	0	0	0

Solution: Use Rom with 6 address lines,
extra two are for inputs. Now next
address (state) depends on inputs also.
Rember: D6-D9 connected to A2-A5
AlSo, state 3H has 4 possible conditions depending
on the 2 inputs. If both inputs =0, next state
(D6-D9) is 0H, otherwise it is 4H (0100).

ROM Table (pg 23)

Current State				Inputs			Next State				Outputs					.
A3	A2	A1	A0	WT	ET		D9	D8	D7	D6	EWR	EWY	EWG	NSR	NSY	NSG
0	0	0	0	X	X	|	0	0	0	1	1	0	0	0	0	1
				.		|					.					
				.		|					.					
0	0	1	1	0	0	|	0	0	0	0	1	0	0	0	0	1
0	0	1	1	0	1	|	0	1	0	0	1	0	0	0	0	1
0	0	1	1	1	0	|	0	1	0	0	1	0	0	0	0	1
0	0	1	1	1	1	|	0	1	0	0	1	0	0	0	0	1
				.		|					.					
				.		|					.					

State machine



How many Flip-Flops are needed for a state machine with 10 states?

How many data pins would a ROM need if this state machine had its 10 states and required 3 outputs?

Flow Chart States

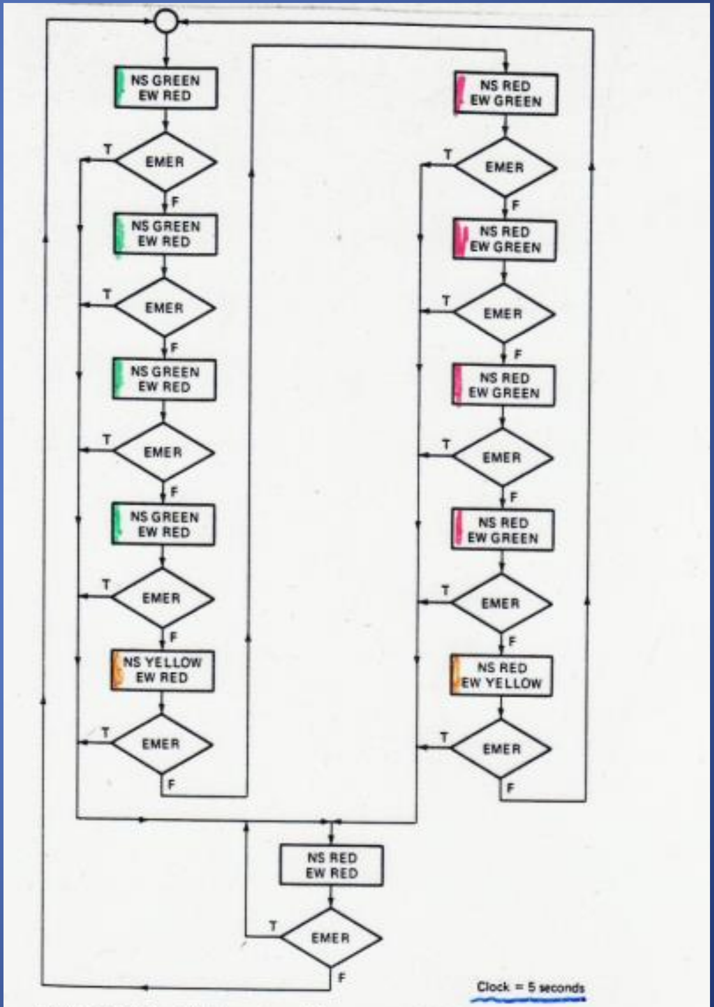
MOORE Machine

Not efficient

Can’t stay in same state when over.

Better way……..

MEALY Machine



Clock = 5 seconds

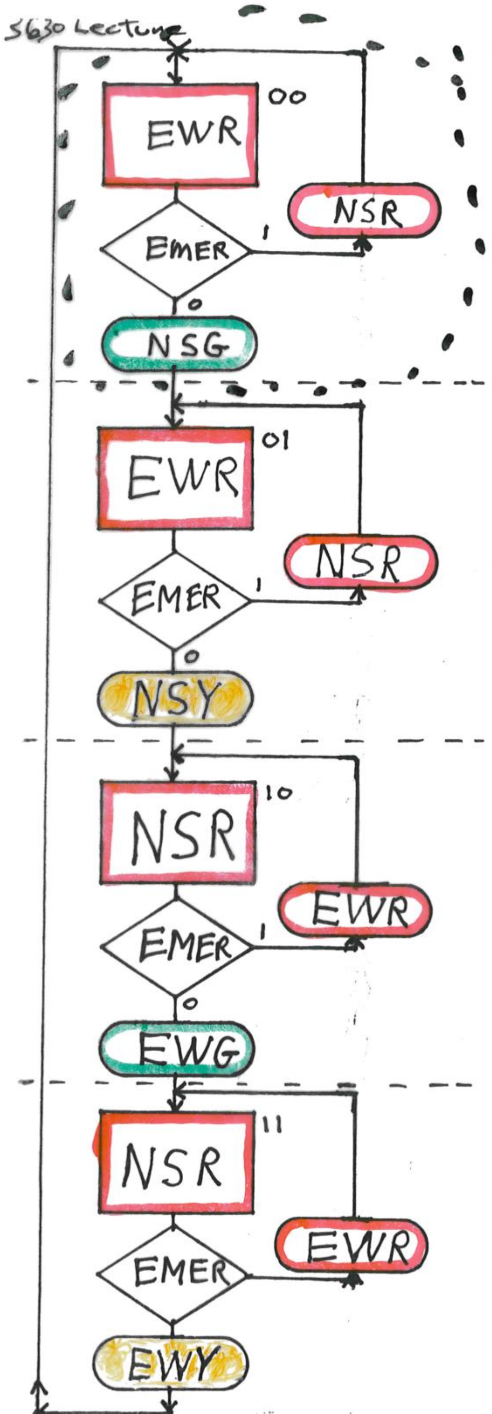
Figure 1-7 Flowchart for intersection with emergency vehicle input with a maximum delay of 5 s.

5630 Lecture

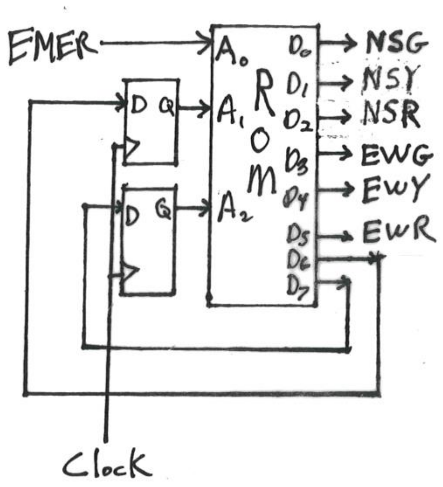
1 of 2

Rom ASM - Mealy

Traffic Light, Emer input
(returns to same state)

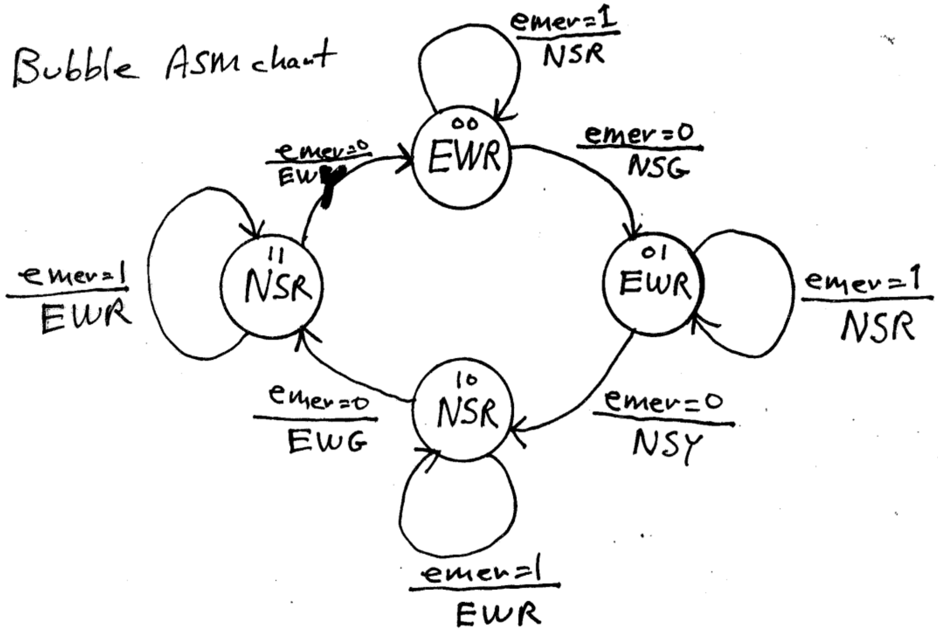


Flowchart



Logic Circuit Diagram

Bubble ASM chart



C.S		Emer	NS		NSG	NSY	NSR	EWG	EWY	EWR
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	1	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0	0	1
0	1	0	1	0	0	1	0	0	0	1
0	1	1	0	1	0	0	1	0	0	1
1	0	0	1	1	0	0	1	1	0	0
1	0	1	1	0	0	0	1	0	0	1
1	1	0	0	0	0	0	1	0	1	0
1	1	1	1	1	0	0	1	0	0	1

Notation… IMPORTANT

Reference Wiatrowski page 28:

Y is for active high inputs (Ex: YINPUT1)
N is for active low inputs.

H is for active high outputs
L is for active low outputs.

Can have “immediate” outputs IHout1 or ILout2.

Notation…(continued)

ASM formal definitions:

Inputs:

YWT = WT sensor ON if logic 1

NWT = WT sensor ON if logic 0.

Outputs:

HEWG = EWG is ON if logic 1

LEWG = EWG is ON if logic 0

Immediate Outputs: IHEWG and ILEWG (lights, etc.)

Synchronous inputs are synchronized with clock pulse. Asynchronous inputs may cause many problems (covered in chapter 5) such as missed signals between clock pulses, or violated setup/hold times!

Asynch Input Problem

a) Synch input,
NRT=1,
LOUT=OFF, NS=AE

b) Synch input,
LRT=0,
LOUT=ON, NS=
AD

c)Asynch input,
NRT=pulse,
LOUT=pulse,
NS=AE

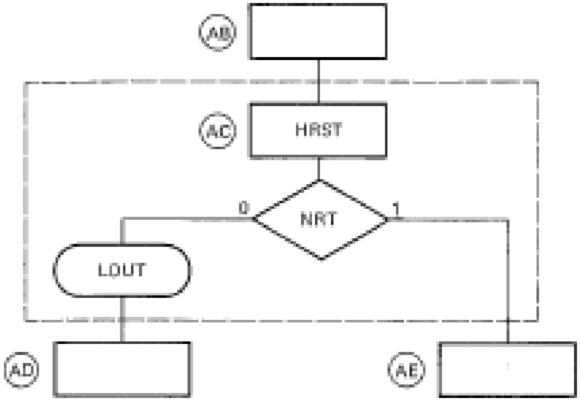
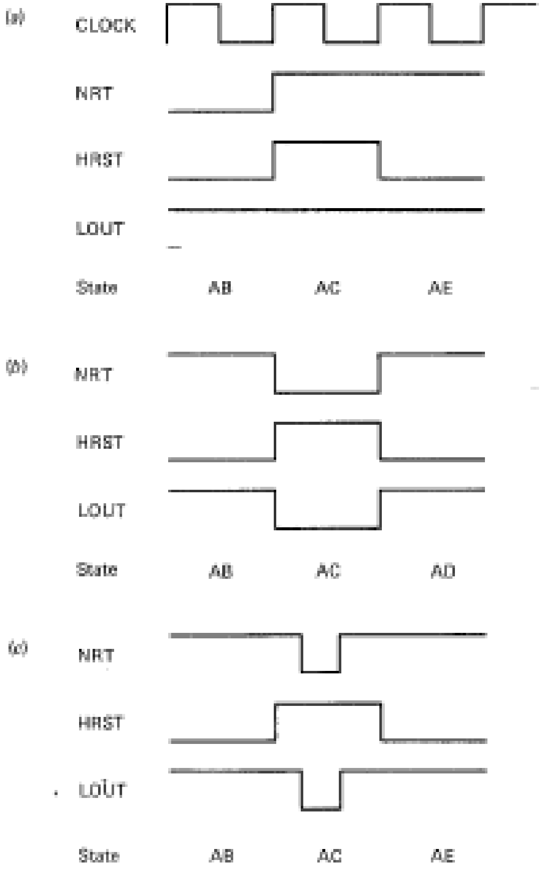


Figure 1-26 ASM block with conditional output.

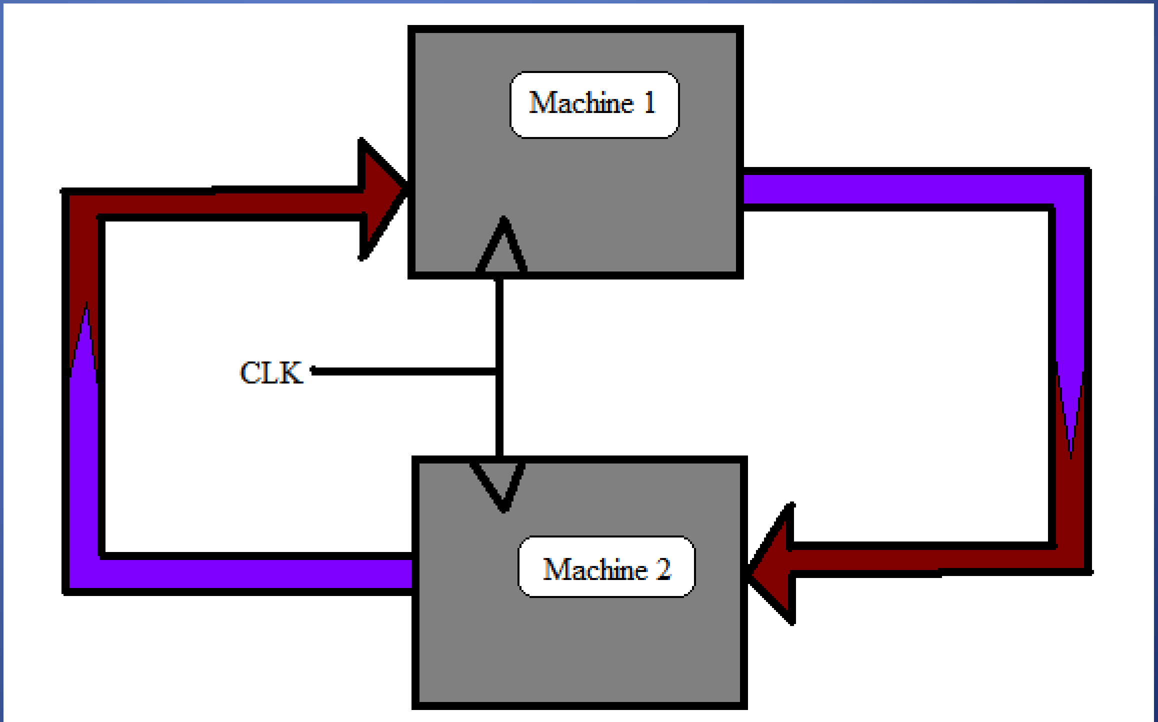


LOUT ON and
goes to AE ???

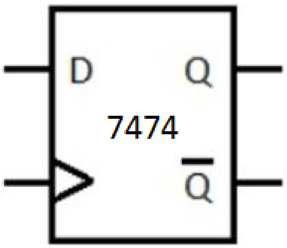
Figure 1-27 Conditional output timing diagrams.

Notation… (continued)

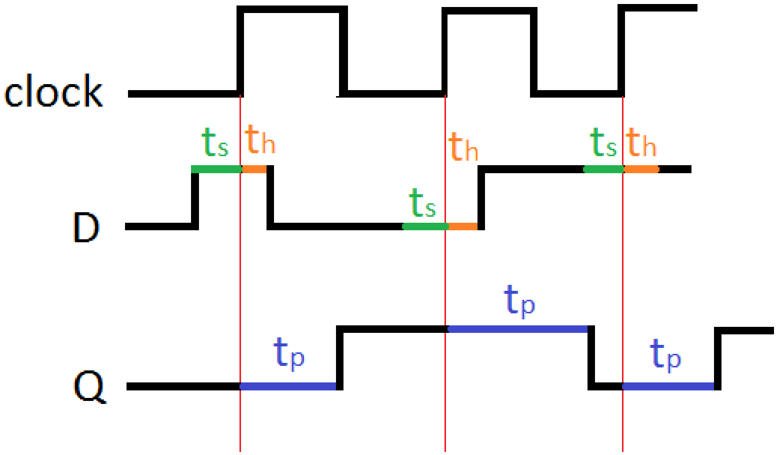
Keep in mind that most outputs from one state machine are inputs to another state machine or to something on the same clock!!



5-5 Clocked Flip Flop Timing



D flip flop timing parameters



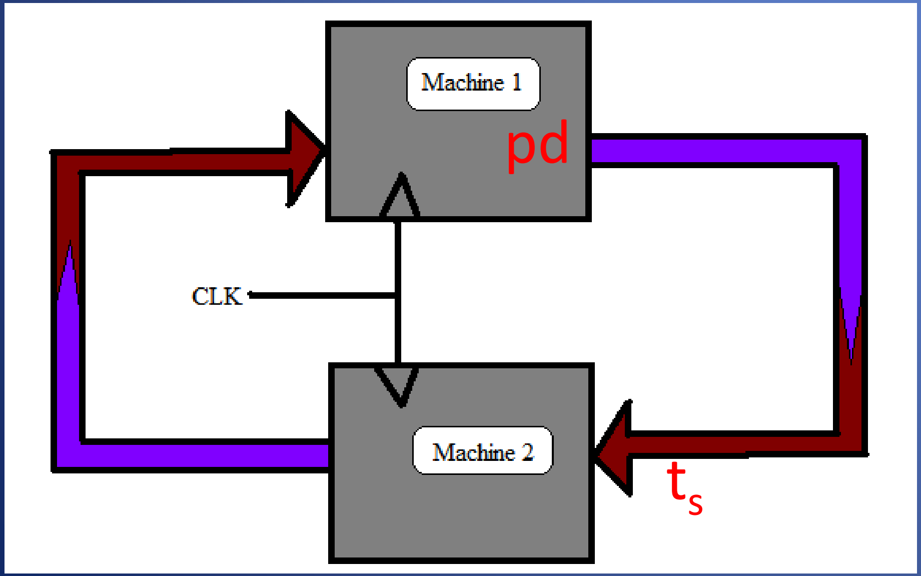
t_p - propagation delay = 25 ns (LH) or 40 ns (HL) for 7474

t_s - setup time = 20 ns for 7474

t_h - hold time = 5 ns for 7474

Delayed by one CLK cycle

IMPORTANT NOTE: Systems will often be connected using the same clock signal. This arrangement shown below will add a DELAY to the overall system! It will take an extra clock cycle for a machine's outputs to be read by the other machine's inputs!!!



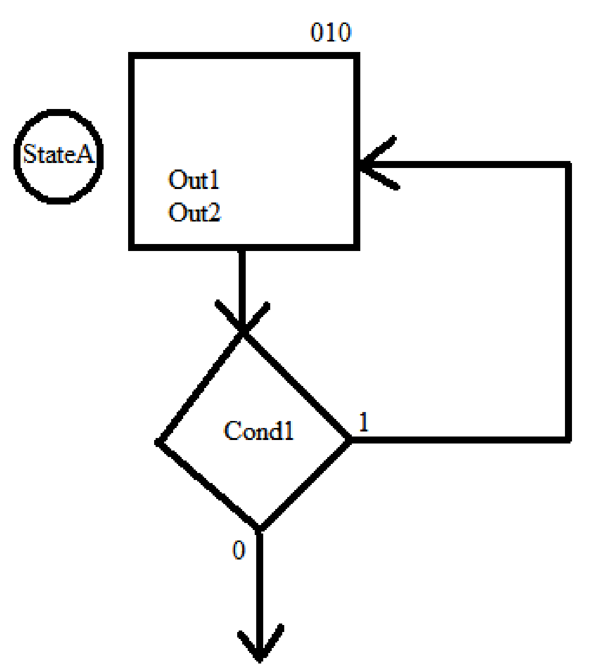
Machine 1 output is delayed by the propagation delay of Machine 1, but Machine 2 needs it to arrive before the clock (setup time).

More about State Machines

List of rules for state diagrams:

1. Must have a state box. You can only enter a state into the state box.
2. Outputs of a state are listed inside the state box (unless they are conditional).
3. State Name goes inside a circle outside the state box.
4. State code (number) on top of the box (in Hex or binary).
5. Decision diamonds must have a 0 or 1 on their exit paths.

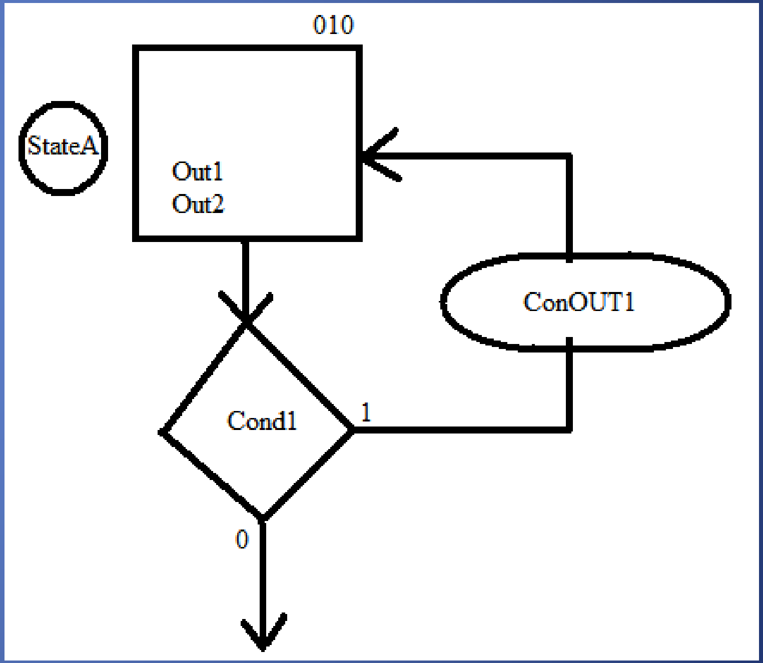
NOTE AGAIN:
Everything in a state happens at the same time!!!



Conditional Outputs

Mealy Machines.

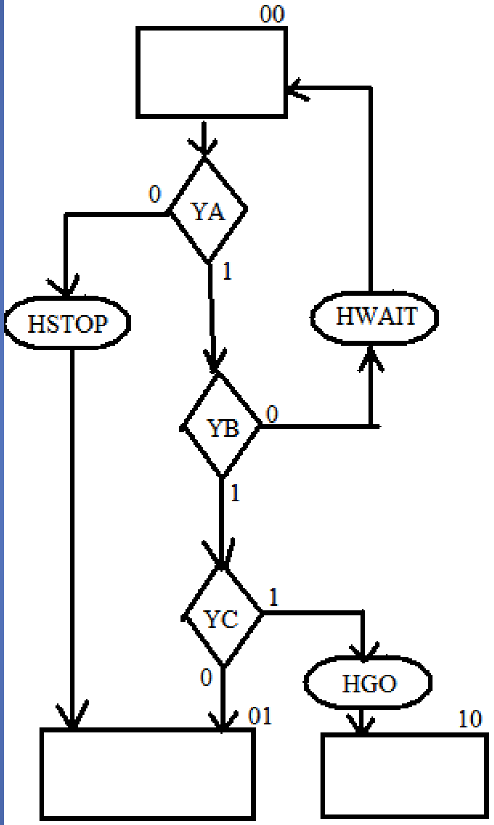
In the example shown here, the output “ConOUT1” will be asserted when the ASM is in StateA and the input “Cond1” is true.



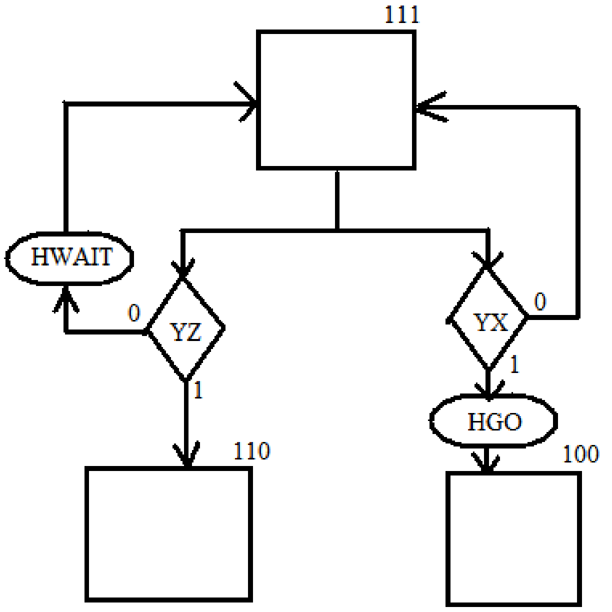
Decision diamonds have no time associated with them. All things in the state happen simultaneously! (CondOUT1 happens at same time as Out1 and Out2)!

Conditional Outputs (continued)

Since the decision diamonds have no time associated with them, a long serial connection of diamonds does not impact the timing of the state machine.



OK!

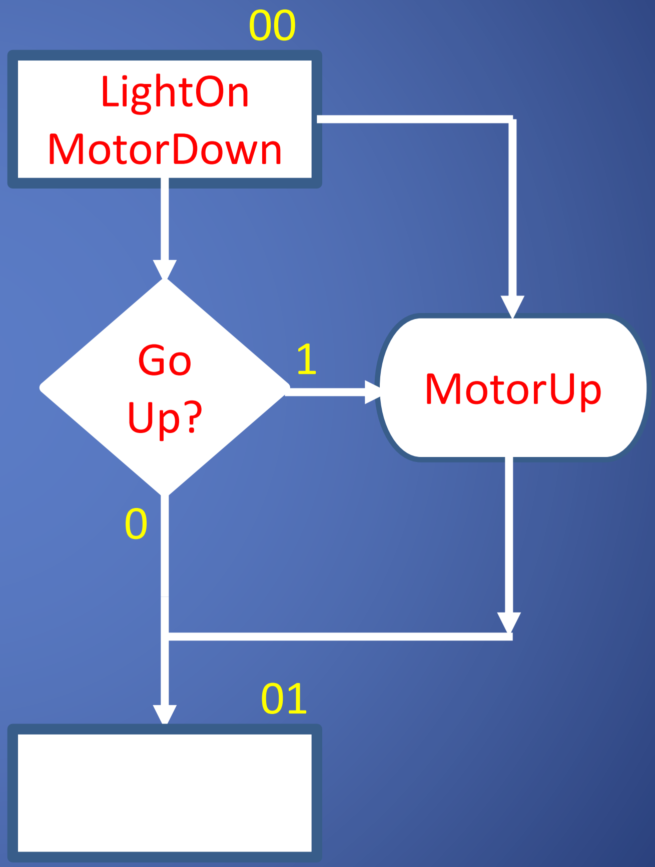


WRONG

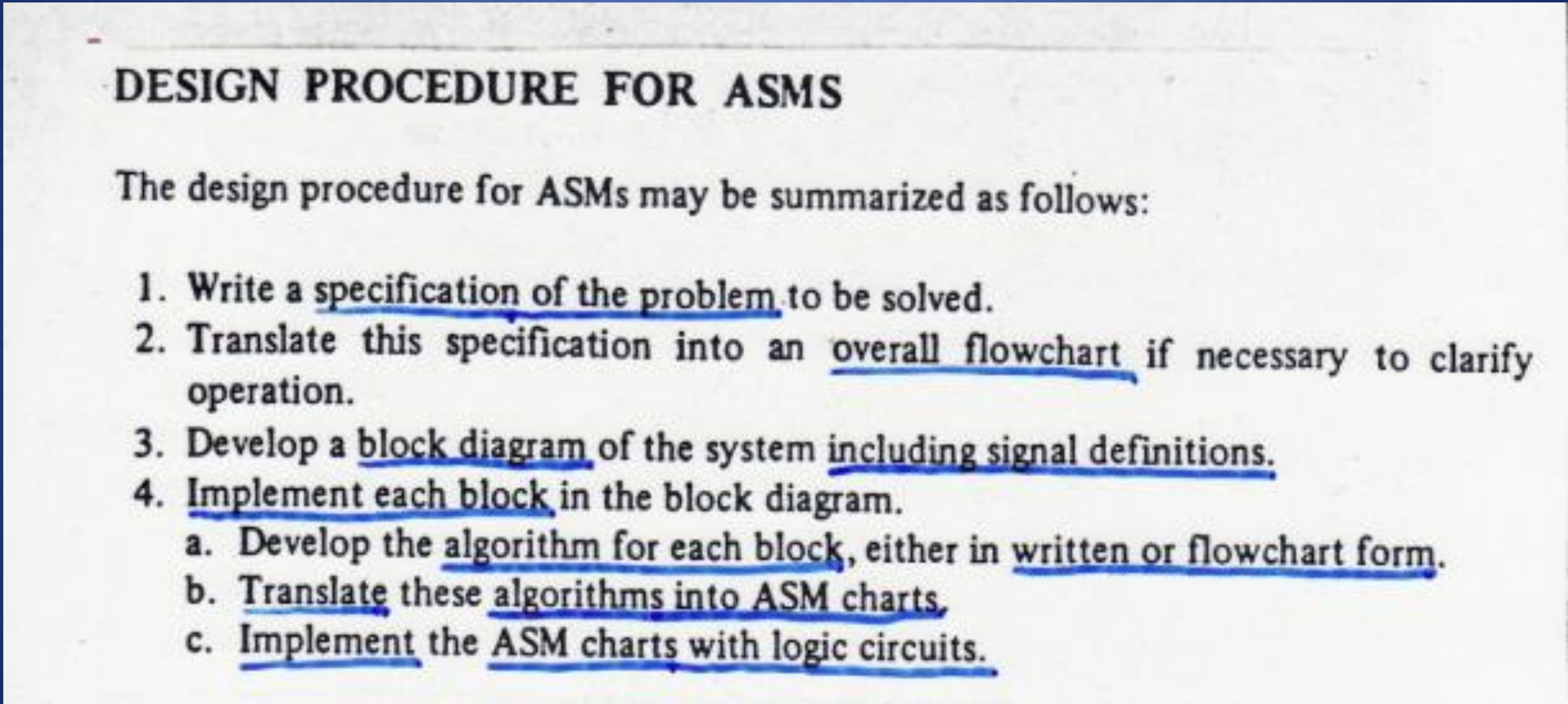
Conditional Outputs

Mealy Machines

Is there anything wrong with this ASM chart?



Design Procedure

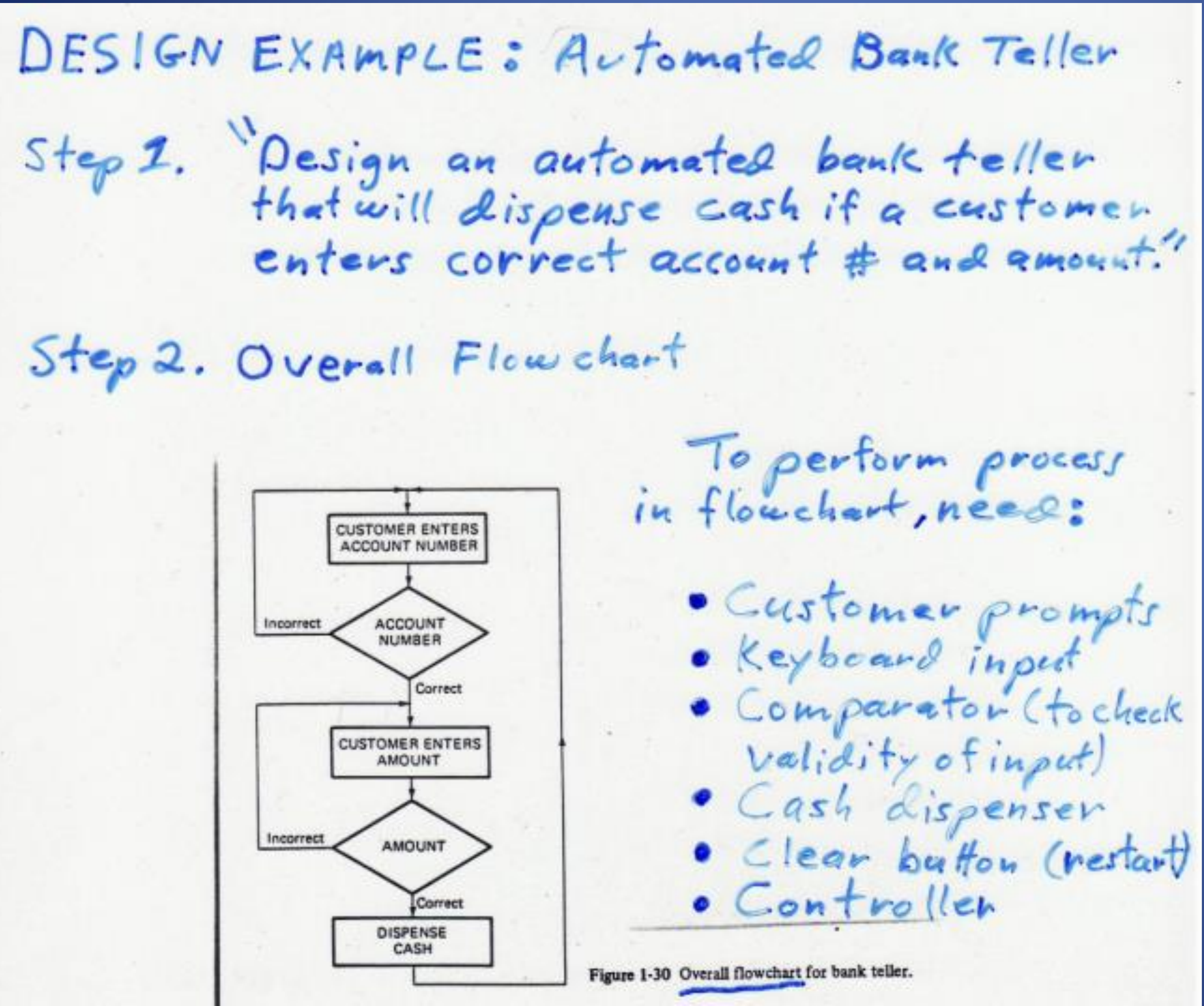


DESIGN PROCEDURE FOR ASMS

The design procedure for ASMs may be summarized as follows:

1. Write a specification of the problem to be solved.
2. Translate this specification into an overall flowchart if necessary to clarify operation.
3. Develop a block diagram of the system including signal definitions.
4. Implement each block in the block diagram.
   a. Develop the algorithm for each block, either in written or flowchart form.
   b. Translate these algorithms into ASM charts.
   c. Implement the ASM charts with logic circuits.

Design Ex: Automated Bank Teller



DESIGN EXAMPLE: Automated Bank Teller

Step 1. "Design an automated bank teller that will dispense cash if a customer enters correct account # and amount."

Step 2. Overall Flowchart

CUSTOMER ENTERS ACCOUNT NUMBER
Incorrect
ACCOUNT NUMBER
Correct
CUSTOMER ENTERS AMOUNT
Incorrect
AMOUNT
Correct
DISPENSE CASH

To perform process in flowchart, need:

- Customer prompts
- Keyboard input
- Comparator (to check validity of input)
- Cash dispenser
- Clear button (restart)
- Controller

Figure 1-30 Overall flowchart for bank teller.

Reference the bottom of page 37 for notation.

A comparator makes sure you have the correct numbers.

ATM Design (continued)

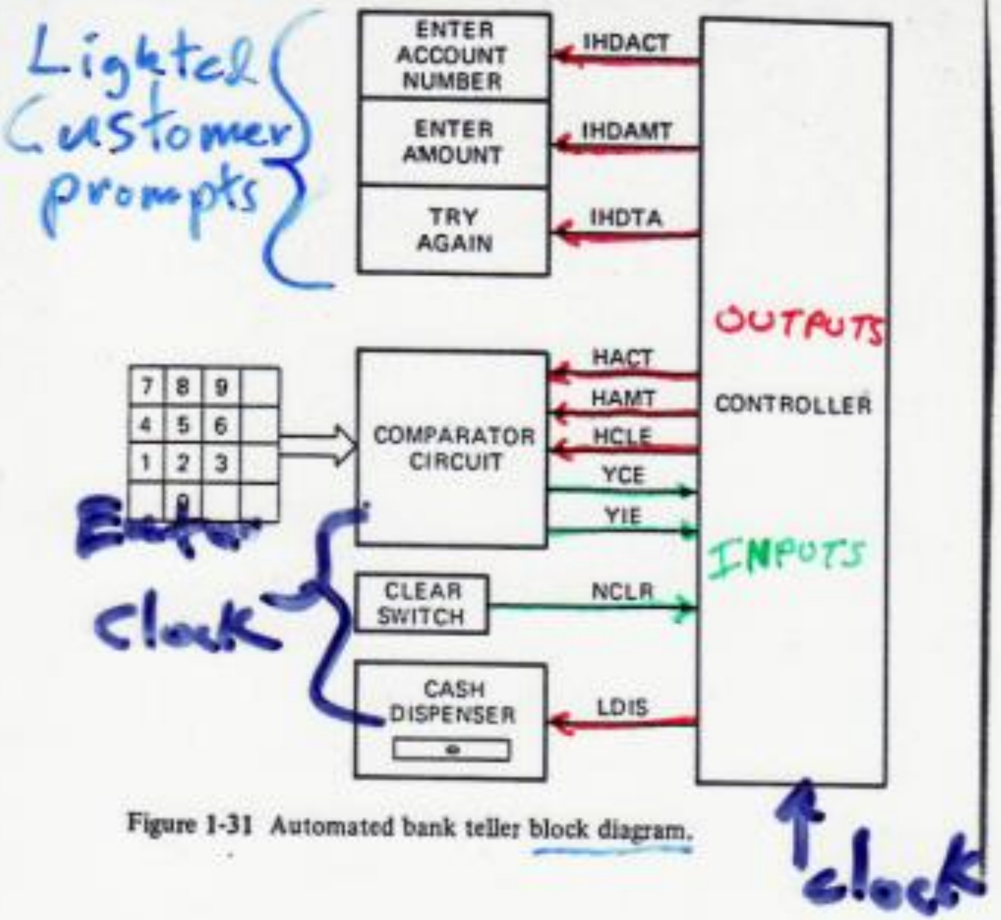


Figure 1-31 Automated bank teller block diagram.

5Q

IHDACT. Display "Enter Account Number."
IHDAMT. Display "Enter Amount."
IHDTA. Display "Try Again."
HACT. Set comparator to accept the account.
HAMT. Set comparator to accept the amount.
YCE. A correct entry has been made.
YIE. An incorrect entry has been made.
HCLE. Resets both *YCE* and *YIE* to 0.
LDIS. Dispense the cash.
NCLR. The clear switch has been pressed.

Controller block directs operation of input blocks, output blocks, and processing block. Very common configuration.

ATM Design (continued)

Step 3: Block diagram with signal definitions.

All outputs must be on as long as that action is needed, i.e., HAMT must be 1 until the customer is finished entering amount.

Need to use short clock period (~50ms) so message can be lighted “simultaneously” with corresponding control signal. For example, IHDACT and HACT (HACT is delayed output, but 50ms is negligible in this case). HACT will take effect before customer can push first number key.

Step 4: Implement each block. We will implement controller block…

ATM Design (continued)

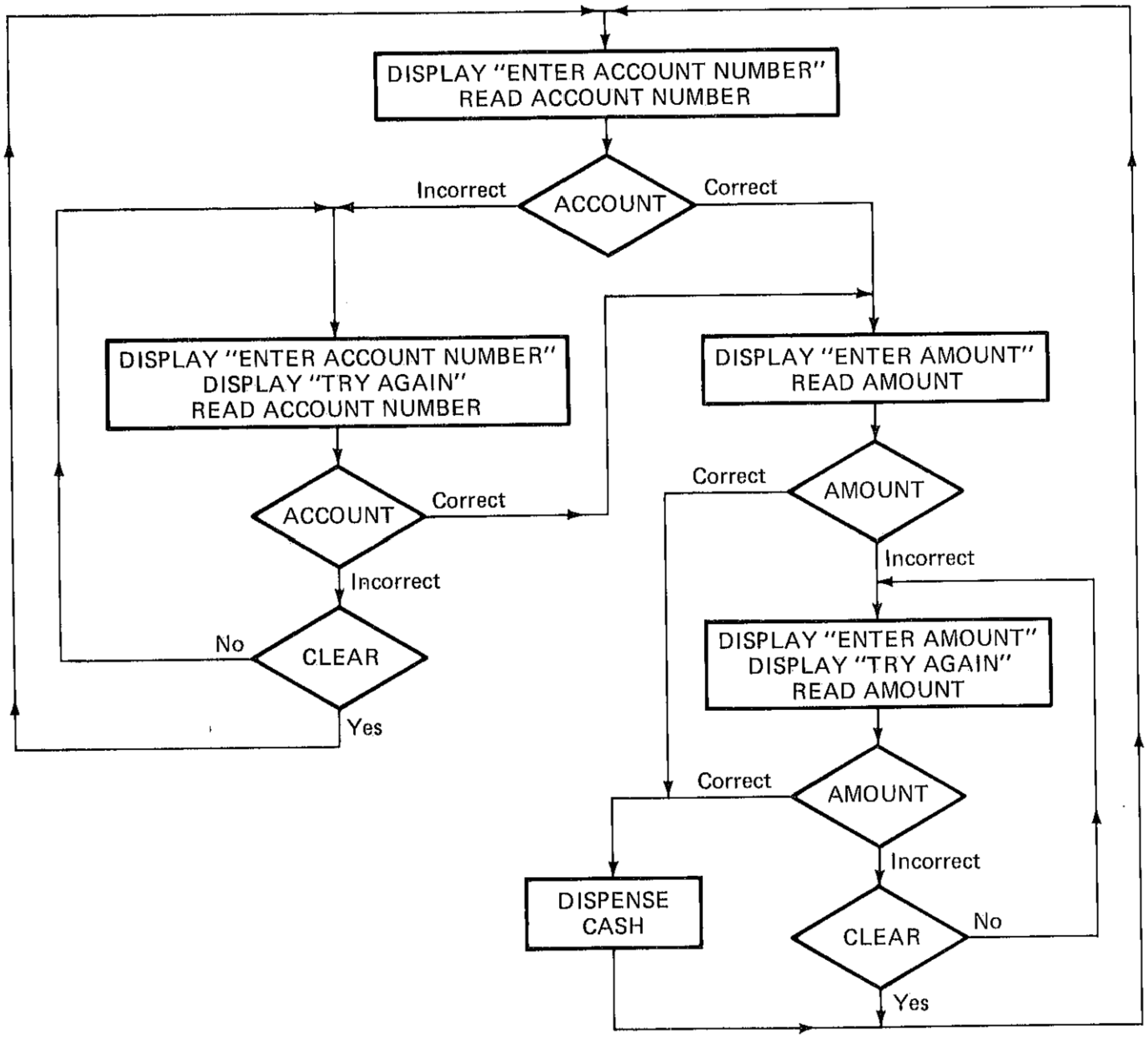


Figure 1-32 Flowchart for bank teller controller.

ATM Design (Moore Machine)

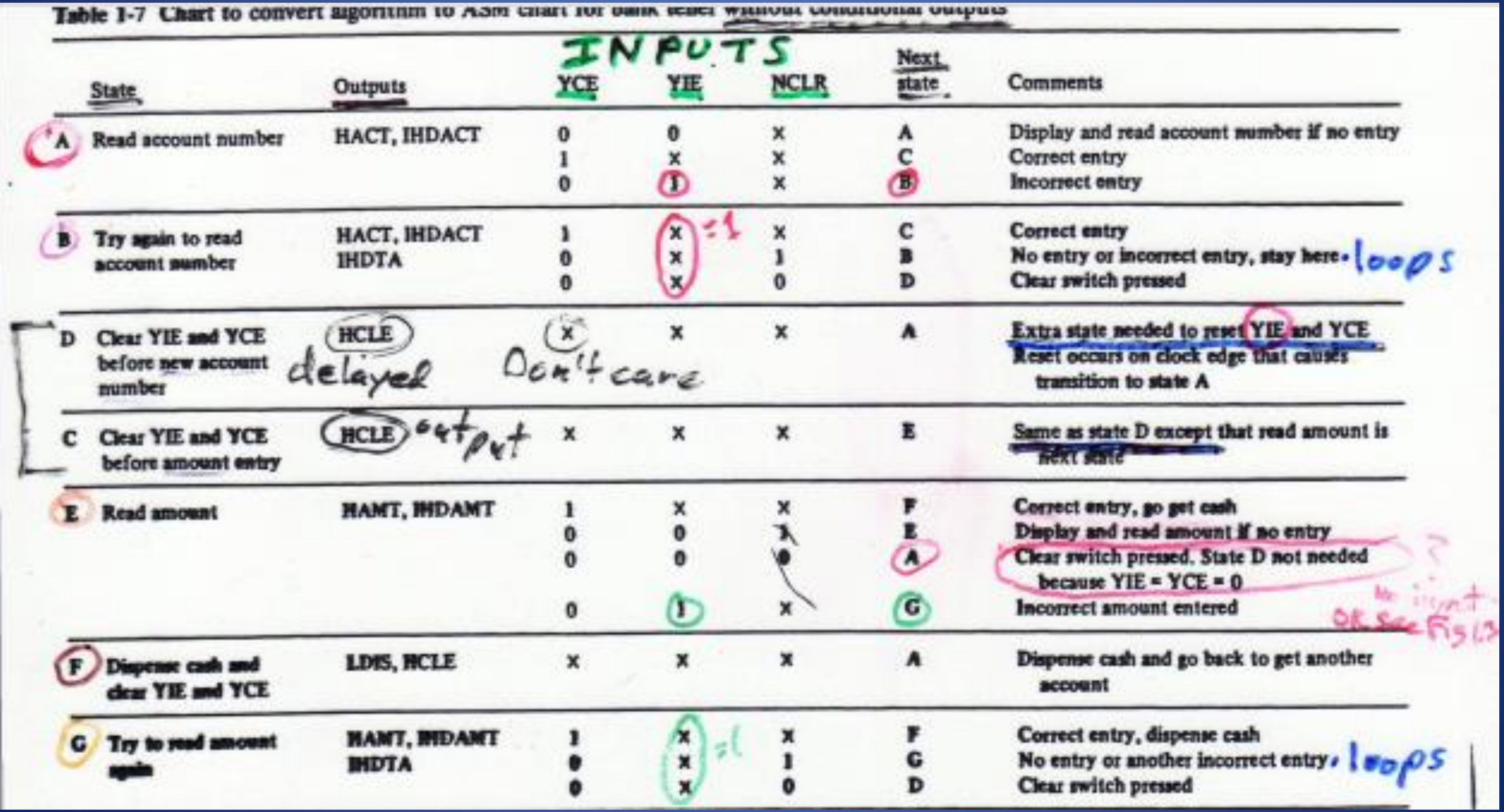


Table 1-7 Chart to convert algorithm to ASM chart for bank teller without conditional outputs

State	Outputs	INPUTS YCE	 YIE	 NCLR	Next state	Comments
A Read account number	HACT, IHDACT	0	0	x	A	Display and read account number if no entry
		1	x	x	C	Correct entry
		0	1	x	B	Incorrect entry
B Try again to read account number	HACT, IHDACT IHDTA	1	x =1	x	C	Correct entry
		0	x	1	B	No entry or incorrect entry, stay here. loops
		0	x	0	D	Clear switch pressed
D Clear YIE and YCE before new account number	HCLE delayed	x Don't care	x	x	A	Extra state needed to reset YIE and YCE Reset occurs on clock edge that causes transition to state A
C Clear YIE and YCE before amount entry	HCLE output	x	x	x	E	Same as state D except that read amount is next state
E Read amount	HAMT, IHDAMT	1	x	x	F	Correct entry, go get cash
		0	0	1	E	Display and read amount if no entry
		0	0	0	A	Clear switch pressed. State D not needed because YIE = YCE = 0 ?
		0	1	x	G	Incorrect amount entered OK See Fig 1.3
F Dispense cash and clear YIE and YCE	LDIS, HCLE	x	x	x	A	Dispense cash and go back to get another account
G Try to read amount again	HAMT, IHDAMT IHDTA	1	x =1	x	F	Correct entry, dispense cash
		0	x	1	G	No entry or another incorrect entry. loops
		0	x	0	D	Clear switch pressed

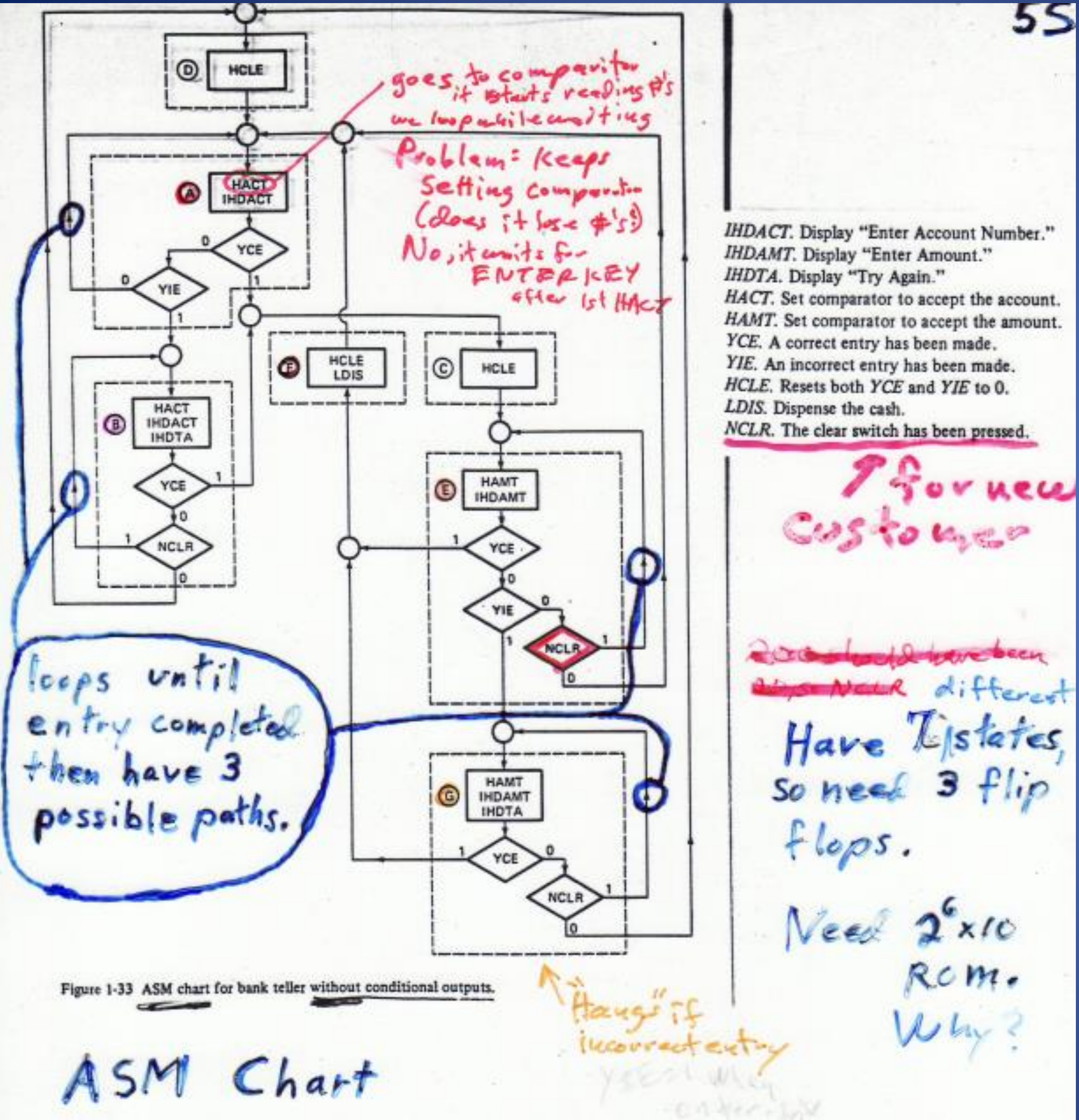

D
HCLE
A
HACT
IHDACT
YCE
YIE
0
1
B
HACT
IHDACT
IHDTA
YCE
NCLR
F
HCLE
LDIS
C
HCLE
E
HAMT
IHDAMT
YCE
YIE
NCLR
G
HAMT
IHDAMT
IHDTA
YCE
NCLR
goes to comparitor
it starts reading #'s
we loop while waiting
Problem: Keeps Setting comparator (does it lose #'s?)
No, it waits for ENTER KEY after 1st HACT
loops until entry completed then have 3 possible paths.
"Hangs" if incorrect entry

Figure 1-33 ASM chart for bank teller without conditional outputs.

IHDACT. Display "Enter Account Number."
IHDAMT. Display "Enter Amount."
IHDTA. Display "Try Again."
HACT. Set comparator to accept the account.
HAMT. Set comparator to accept the amount.
YCE. A correct entry has been made.
YIE. An incorrect entry has been made.
HCLE. Resets both *YCE* and *YIE* to 0.
LDIS. Dispense the cash.
NCLR. The clear switch has been pressed.

↑ for new customer

~~[illegible] should have been~~
~~[illegible] NCLR~~ different

Have 7 states, so need 3 flip flops.

Need $2^6 \times 10$ ROM.
Why?

ASM Chart

Waits for "enter" key.

- Loops if not incorrect entry, or if no button is pressed.
- If correct, then we go into another state.

If cleared, then starts over.

Note: States C and D do nothing except assert HCLE one clock cycle ahead of state in which YIE and YCE must be reset because HCLE is delayed output.

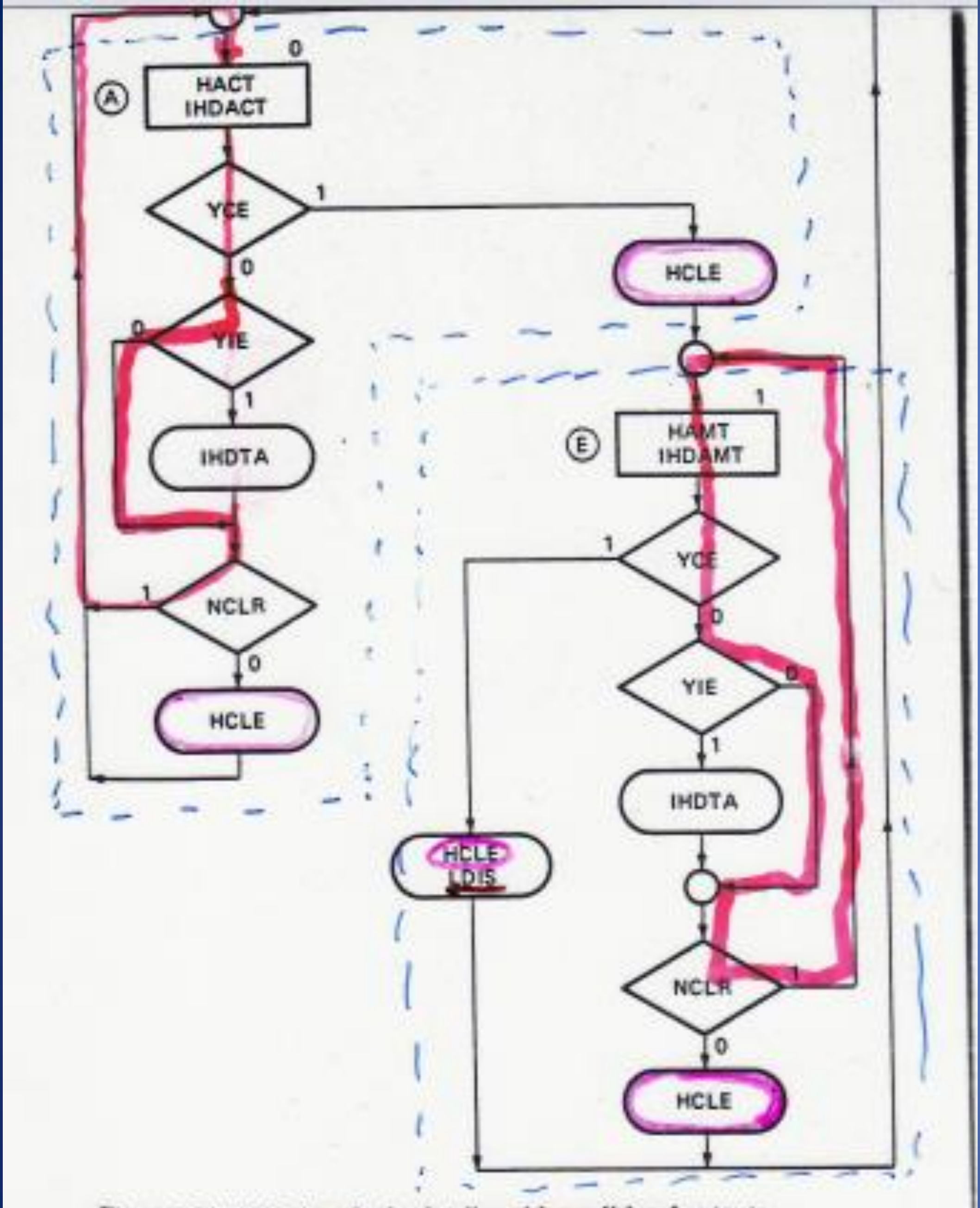


Figure 1-34 ASM chart for bank teller with conditional outputs.

This is the same design with a Mealy Machine.

Red path is if they don’t press anything. User hits enter key at end of entry.

Conditional Outputs: IHDTA, HCLE, LDIS.

Using HCLE as conditional output saves 2 states (C and D) because HCLE can be asserted (when needed) in same clock cycle that causes next state to occur. Why couldn’t this be done without conditional outputs?

Note that this uses 2 states.

The previous design used 7.

ATM Design (continued)

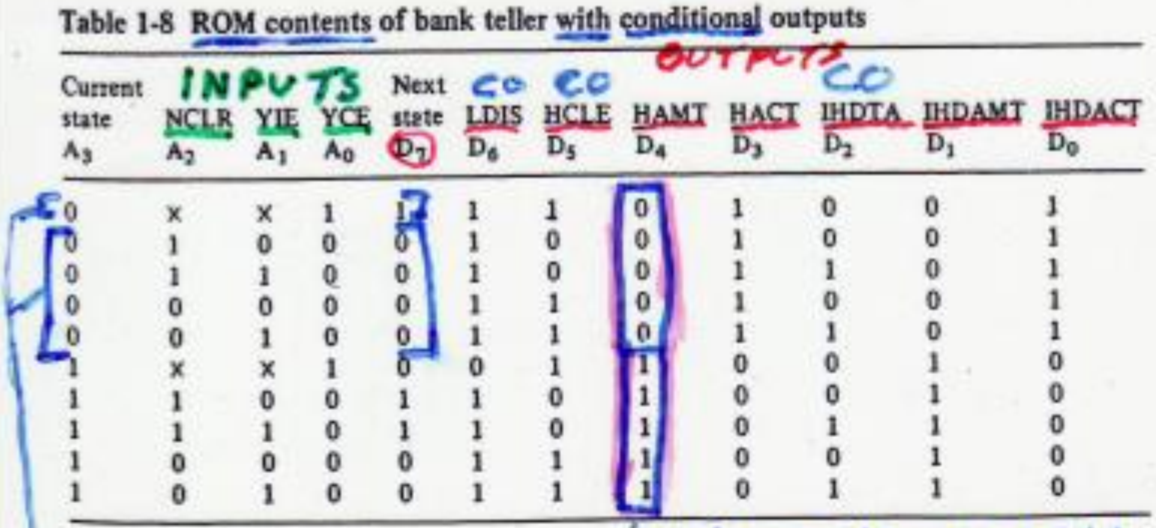


Table 1-8 ROM contents of bank teller with conditional outputs

	INPUTS				OUTPUTS						
					CO	CO			CO		
Current state A_3	NCLR A_2	YIE A_1	YCE A_0	Next state D_7	LDIS D_6	HCLE D_5	HAMT D_4	HACT D_3	IHDTA D_2	IHDAMT D_1	IHDACT D_0
0	x	x	1	1	1	1	0	1	0	0	1
0	1	0	0	0	1	0	0	1	0	0	1
0	1	1	0	0	1	0	0	1	1	0	1
0	0	0	0	0	1	1	0	1	0	0	1
0	0	1	0	0	1	1	0	1	1	0	1
1	x	x	1	0	0	1	1	0	0	1	0
1	1	0	0	1	1	0	1	0	0	1	0
1	1	1	0	1	1	0	1	0	1	1	0
1	0	0	0	0	1	1	1	0	0	1	0
1	0	1	0	0	1	1	1	0	1	1	0

E

not conditional output!

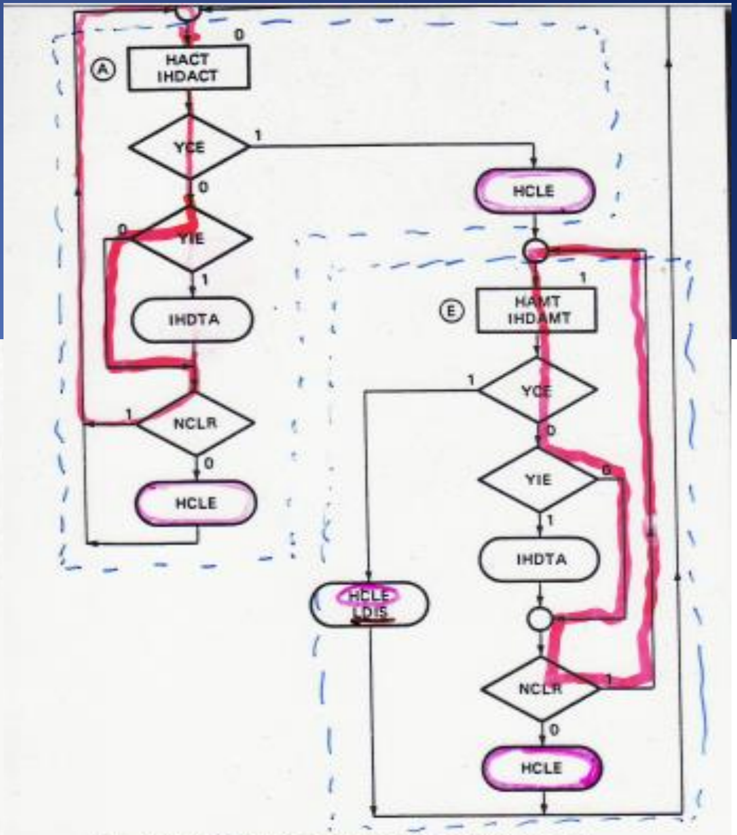


Figure 1-34 ASM chart for bank teller with conditional outputs.

Next state is 1 or 0, depending on inputs.

LDIS, HCLE, IHDTA also depend on inputs,
other outputs don't (HAMT).

Rom = $2^4 \times 8$ only! why?

Only have 2 states 0,1 (1 flip flop needed)!

Compare Moore and Mealy Designs

Moore

7 states (2^3=8)

thus 3 address lines

+3 inputs = 6 total.

3 lines for “next state”

+ 7 outputs = 10 data lines.

So, 2^6x10=64x10 ROM

But would have to buy 64x16.

Mealy

Compare Moore and Mealy Designs

Moore

7 states ($2^3=8$)

thus 3 address lines

+3 inputs = 6 total.

3 lines for “next state”

+ 7 outputs = 10 data lines.

So, 2^6x10=64x10 ROM

But would have to buy 64x16.

Mealy

2 states ($2^1=2$)

thus 1 address line

+3 inputs = 4 total.

Compare Moore and Mealy Designs

Moore	Mealy
7 states (2^3=8)	2 states (2^1=2)
thus 3 address lines	thus 1 address line
+3 inputs = 6 total.	+3 inputs = 4 total.
3 lines for “next state”	1 line for “next state”
+ 7 outputs = 10 data lines.	+7 outputs=8 data.
So, 2^6x10=64x10 ROM	2^4x8=16x8 ROM
But would have to buy 64x16.	

Compare Moore and Mealy Designs

Moore	Mealy
64x16 ROM	16x8 ROM
	Smaller, Cheaper, More Powerful… Has glitch issues – see Chu