

FSM encoding

as truth table ↴

Input state			next state output		
000	0	s_0	s_3	0	110
001	1	s_0	s_1	1	011
010	0	s_1	s_2	1	101
011	1	s_1	s_3	0	110
100	0	s_2	s_3	0	110
101	1	s_2	s_1	1	011
110	0	s_3	s_3	0	110
111	1	s_3	s_3	0	110

{ Input } { Output }

4 states : s_0 00
 s_1 01
 s_2 10
 s_3 11

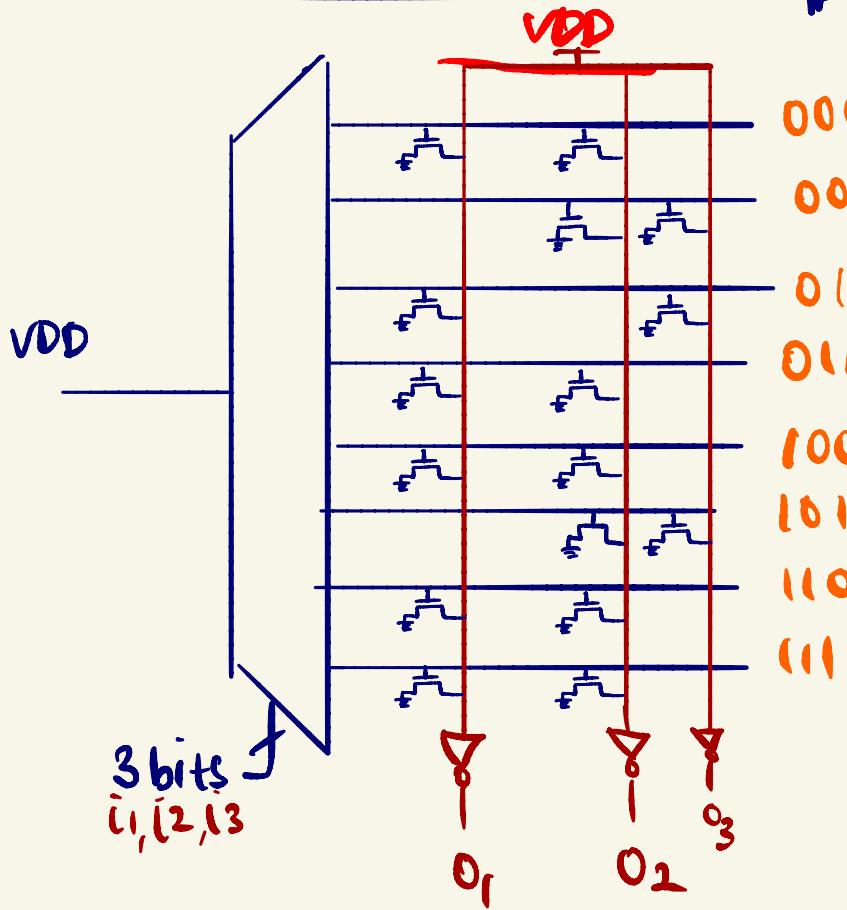
2 inputs : 1
 0

2 outputs : 1
 0



only 3 output-state combi, can be encoded into just 2 bits actually.

FSM as ROM



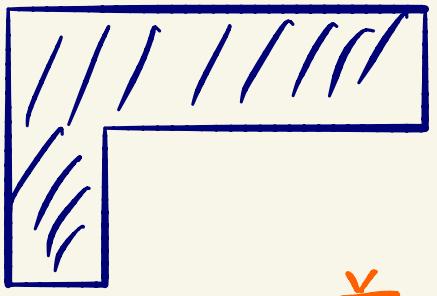
implemented as ROM

fsm is a truth table

$i_1 i_2 i_3$	$O_1 O_2 O_3$
000	110
001	011
010	101
011	110
100	110
101	011
110	110
111	110

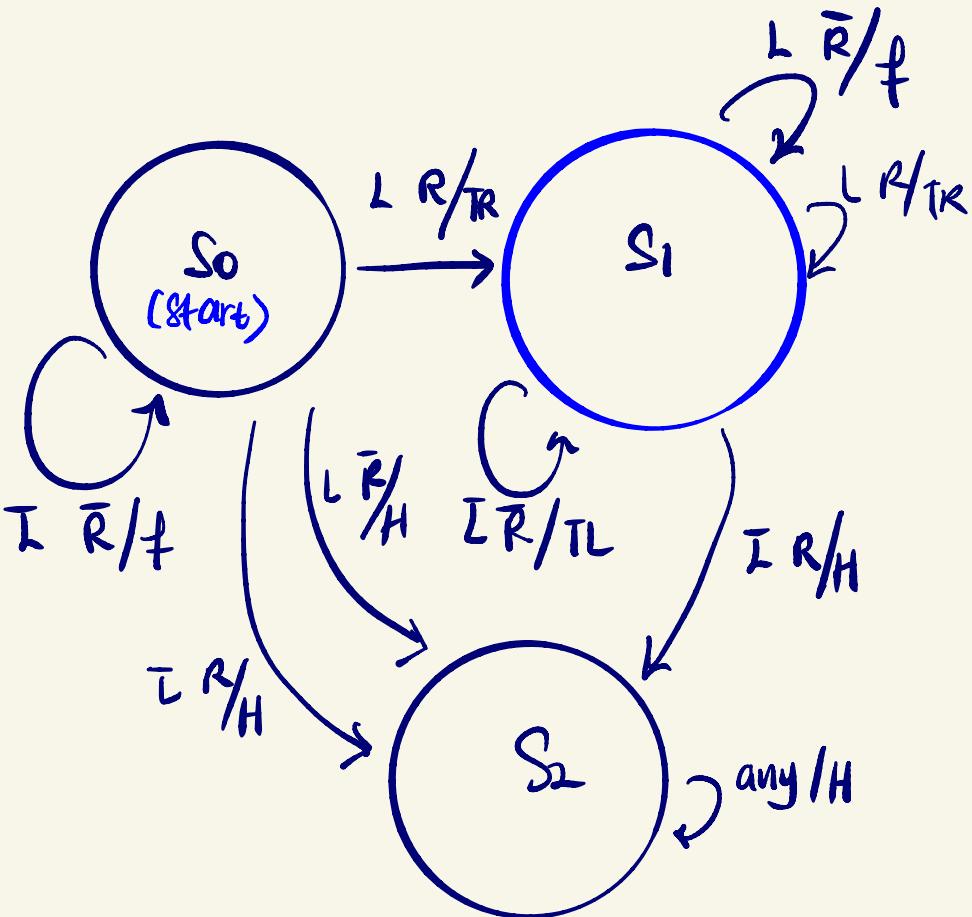
Input Output
(of the next state)

Another FSM Example

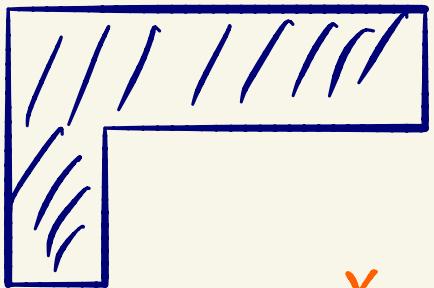


$\bar{L}/\bar{R} = 0 \rightarrow$ white

$\bar{L}/\bar{R} = 1 \rightarrow$ black



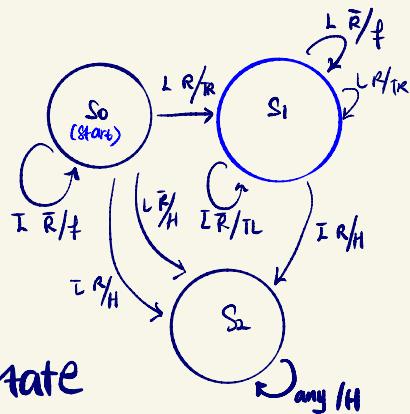
Another FSM Example



$\bar{L}/\bar{R} = 0 \rightarrow$ white

$\bar{L}/\bar{R} = 1 \rightarrow$ black

Input	State	Output	state
L R	S0	TR	S1
$\bar{L} \bar{R}$	S0	F	S0
L R	S0	H	S2
$\bar{L} \bar{R}$	S0	H	S2
L R	S1	TR	S1
$\bar{L} \bar{R}$	S1	H	S2
L R	S1	TL	S1
$\bar{L} \bar{R}$	S1	F	S1
any	S2	H	S2



Another FSM Example

S_0 00
 S_1 01
 S_2 10

Input	State	Output state				
00 00	L R S_0	TR	S_1	01 01		
11 00	$\bar{L} \bar{R}$ S_0	F	S_0	00 00		
10 00	T R S_0	H	S_2	11 10		
01 00	L \bar{R} S_0	H	S_2	11 10		
00 01	L R S_1	TR	S_1	01 01		
00 01	$\bar{L} R$ S_1	H	S_2	11 10		
10 01	$\bar{L} \bar{R}$ S_1	TL	S_1	00 01		
01 01	L \bar{R} S_1	F	S_1	00 01		
00 {0}	any S_2	H	S_2	11 10		
01 10				11 10		
10 10				11 10		
11 10				11 10		

$L \ R$ 00
 $L \ \bar{R}$ 01
 $\bar{L} \ R$ 10
 $\bar{L} \ \bar{R}$ 11

 F 00
 TR 01
 TL 10
 H 11

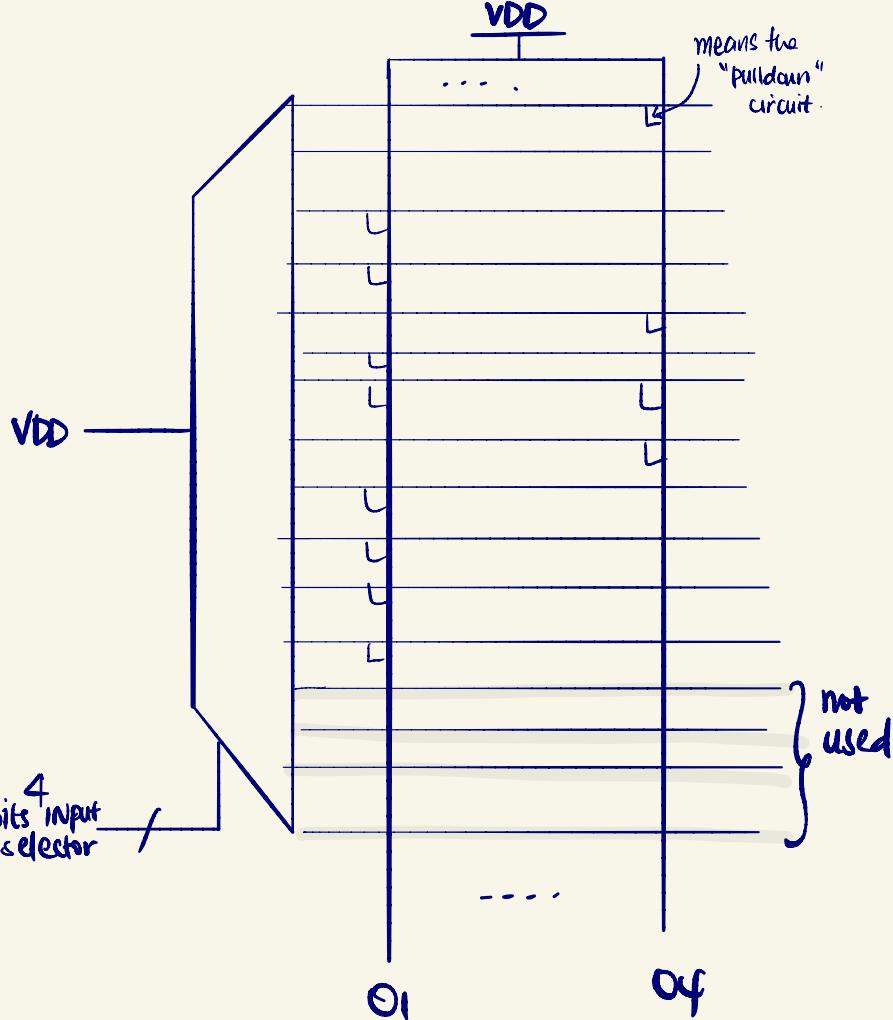
Another FSM Example

4 bits input selector

	Input	State		Output state
00 00	L + R	S ₀	TR	S ₁ 01 01
11 00	L + R	S ₀	F	S ₀ 00 00
10 00	L + R	S ₀	H	S ₂ 11 10
01 00	L + R	S ₀	H	S ₂ 11 10
00 01	L + R	S ₁	TR	S ₁ 01 01
10 01	L + R	S ₁	H	S ₂ 11 10
11 01	L + R	S ₁	TL	S ₁ 00 01
11 01	L + R	S ₁	F	S ₁ 00 01
01 01	L + R	S ₁	H	S ₂ 11 10
00 10	any	S ₂		11 10
01 10				11 10
10 10				11 10
11 11				11 10

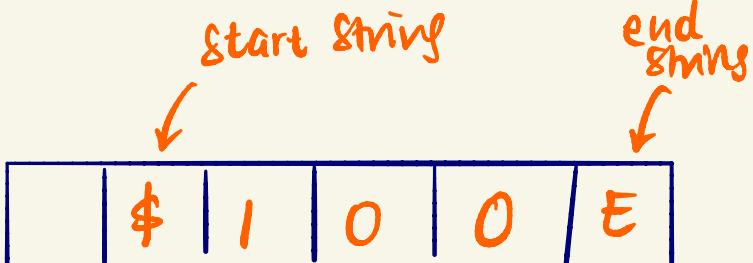
4 output bits

4
bits
input
selector



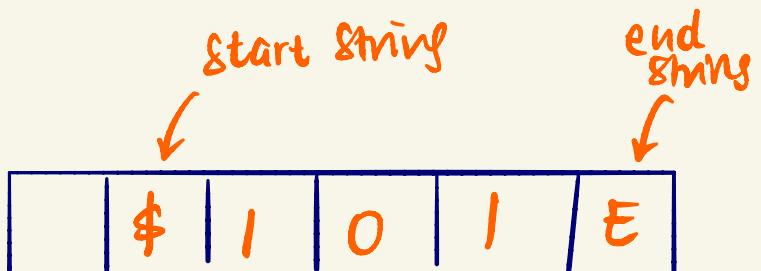
means the
"pulldown"
circuit.

} not
used



Case 1:
LSB is 0

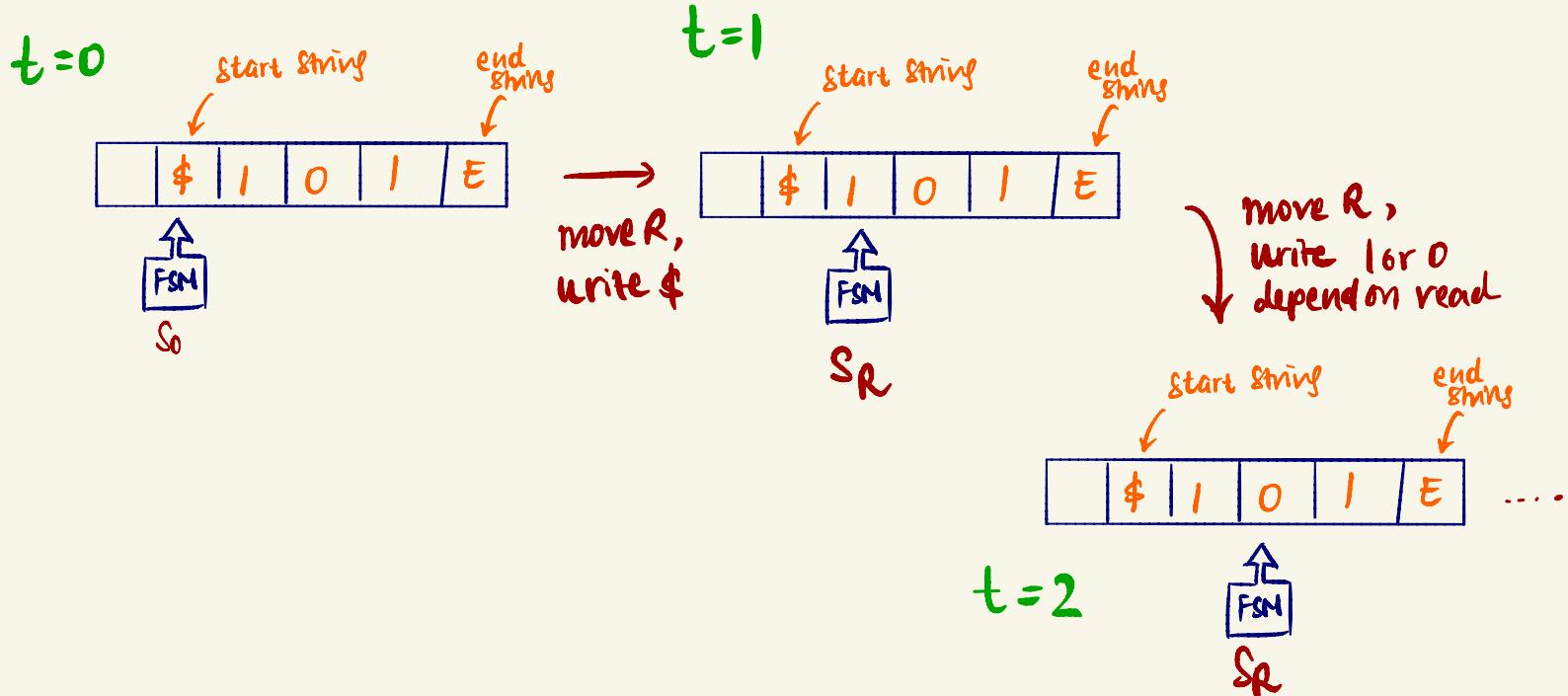
S_{in}	Input	S_{next}	write	move
S_0	\$	S_R	\$	R
S_R	0/1	S_R	0/1	R
S_R	E	S_A	E	L
S_A	0	S_L	1	L
S_L	0/1	S_L	0/1	L
S_L	\$	HALT	\$	R



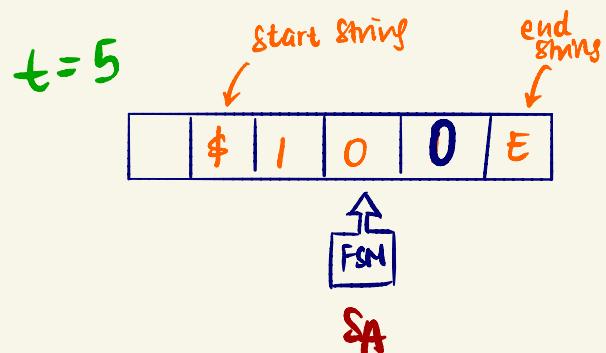
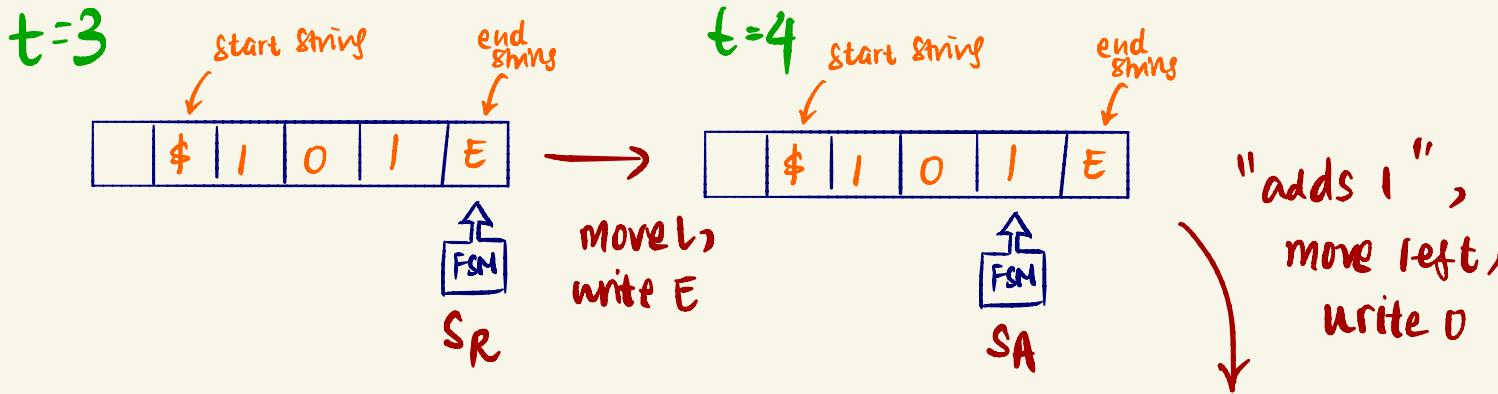
Sin	Input	Snext	write	move
S ₀	\$	S _R	\$	R
S _R	0/1	S _R	0/1	R
S _R	E	S _A	E	L
S _A	I	S _A	O	L
S _A	\$	HALT	I	R

case 2:
LSB is 1

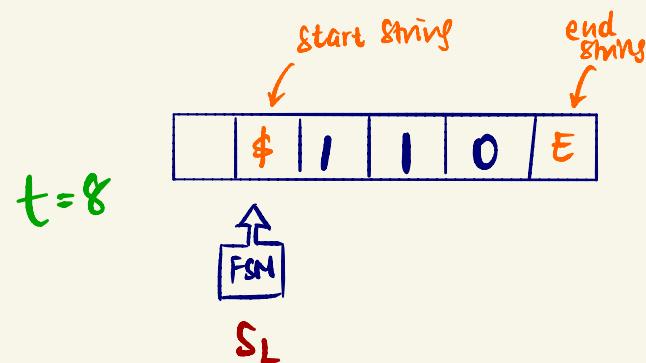
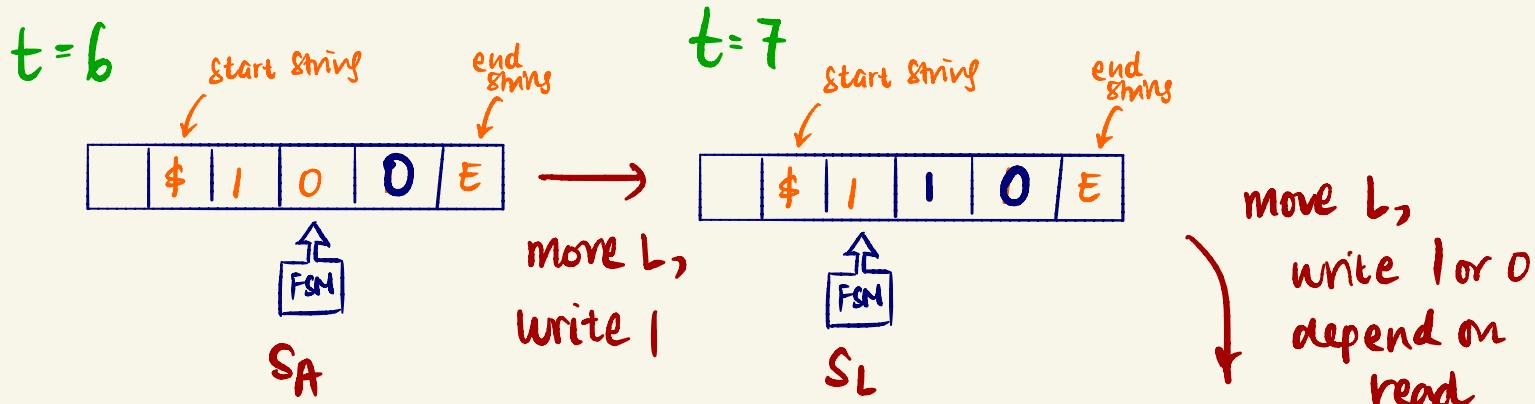
lets run it !



lets run it !

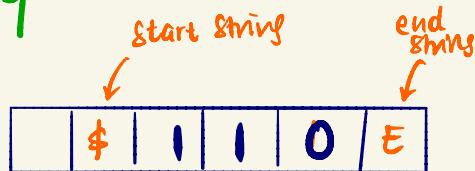


lets run it !



lets run it !

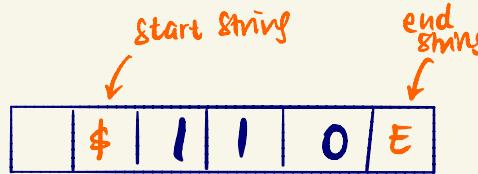
$t=9$



move R,

Halt

(task finished)



$t = 10$

What about other cases ?

Sim	Input	Snext	wire	more
S ₀	\$	S _R	\$	R
S _R	0/1	S _R	0/1	R
S _R	E	S _A	E	L
S _A	1	S _A	0	L
S _A	\$	HALT	1	R
S _A	0	S _R	1	L
S _L	0/1	S _L	0/1	L
S _L	\$	HALT	\$	R

case 2:
LSB is 1

case 1: LSB is 0

So with input % or E?

S_R with input \$?

S_A with Input E?

etc ...

These will all result in
HALT(), basically
because it is an
invalid input