

FSM encoding

as truth table ↴

Input state			next state output		
000	0	s_0	s_3	0	00
001	1	s_0	s_1	1	01
010	0	s_1	s_2	1	10
011	1	s_1	s_3	0	00
100	0	s_2	s_3	0	00
101	1	s_2	s_1	1	01
110	0	s_3	s_3	0	00
111	1	s_3	s_3	0	00

{ Input } { Output }

Output :

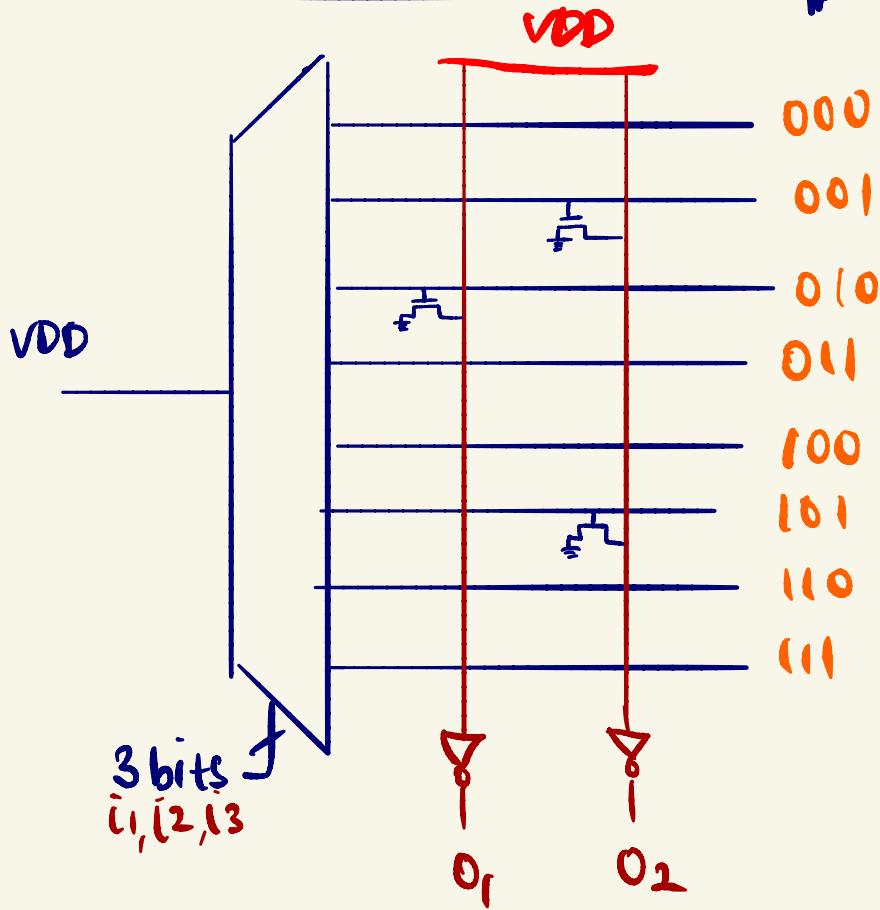
s_1	1	{ 2 bits ↓ Some other output doesn't exist, e.g.: $s_2, 0$
s_2	1	
s_3	0	

Input :

{ Input bit } { 4 states = 2 bits }

0	s_0	{ $H_2 =$ 3 bits in total to encode $I + S$
1	s_0	
0	s_1	
1	s_1	
0	s_2	
1	s_2	
0	s_3	
1	s_3	

FSM as ROM



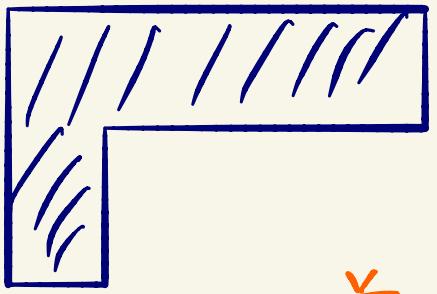
implemented as ROM

fsm is a truth table

i_1, i_2, i_3	O_1, O_2
000	00
001	01
010	10
011	00
100	00
101	01
110	00
111	00

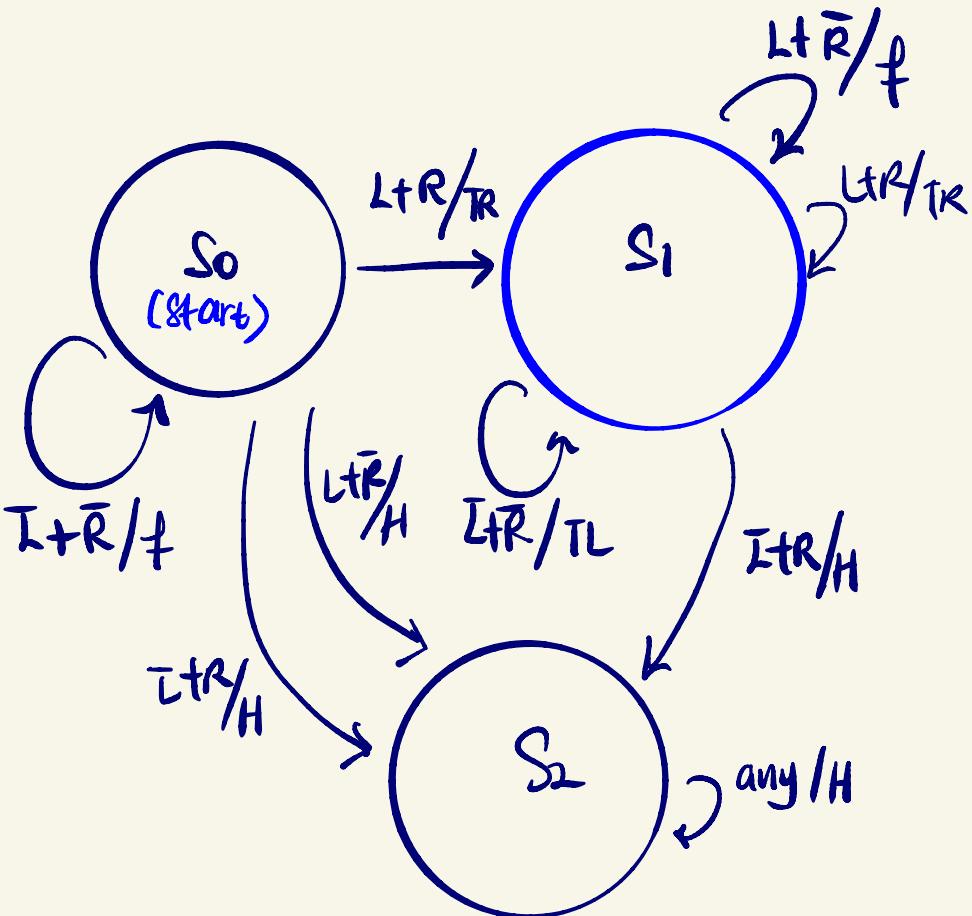
input output

Another FSM Example

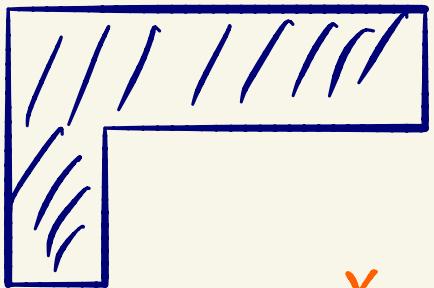


$L/R = 0 \rightarrow$ white

$L/R = 1 \rightarrow$ black



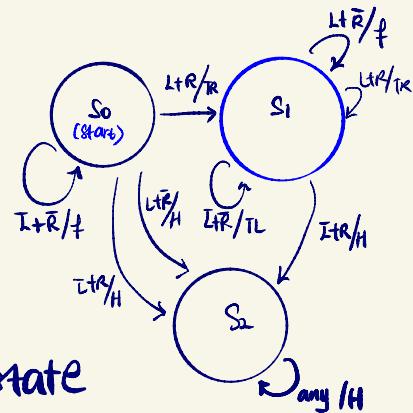
Another FSM Example



$L/R = 0 \rightarrow$ white

$L/R = 1 \rightarrow$ black

Input	State	Output	state
$L+R$	S_0	TR	S_1
$\bar{L}+\bar{R}$	S_0	F	S_0
$\bar{L}+R$	S_0	H	S_2
$L+\bar{R}$	S_0	H	S_2
$L+R$	S_1	TR	S_1
$\bar{L}+R$	S_1	H	S_2
$\bar{L}+\bar{R}$	S_1	TL	S_1
$L+\bar{R}$	S_1	F	S_1
any	S_2	H	S_2



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	$\bar{L} + R$	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
10 01	$\bar{L} + R$	S_1	H	S_2 11 10
11 01	$\bar{L} + \bar{R}$	S_1	TL	S_1 10 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				111 0
10 10				111 0
11 10				111 0

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

 F 00
 TR 01
 TL 10
 H 11

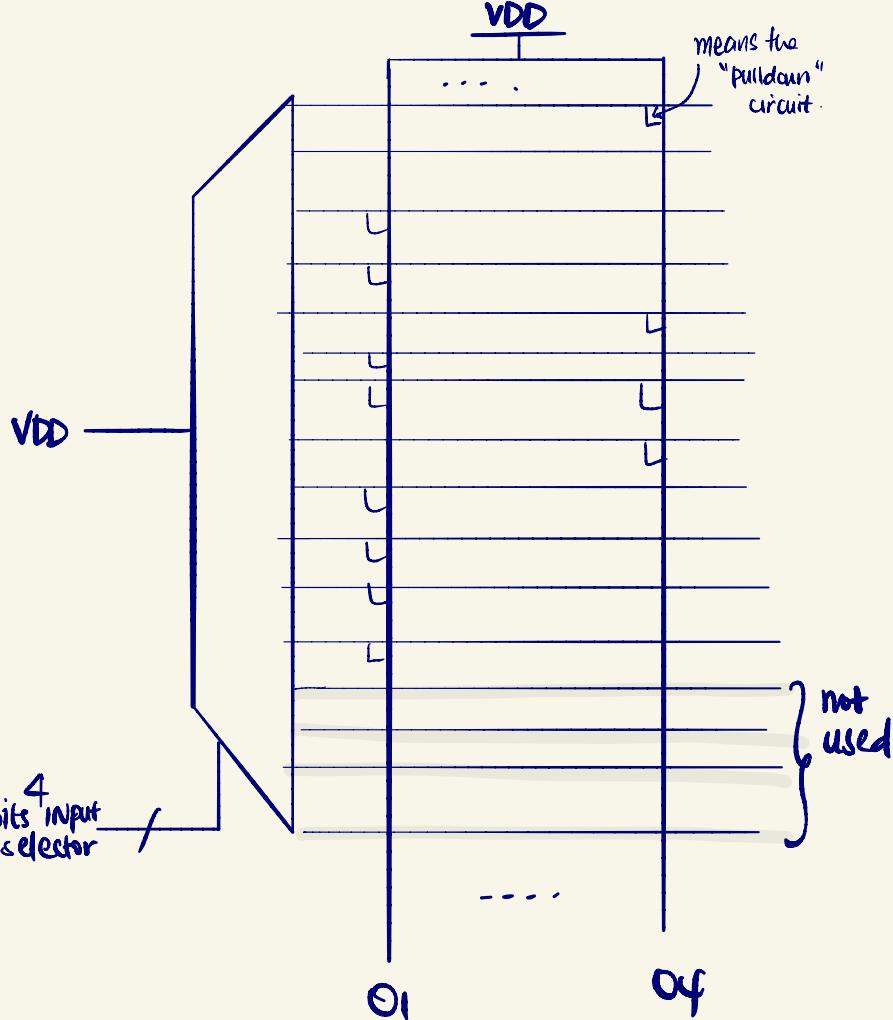
Another FSM Example

4 bits input selector

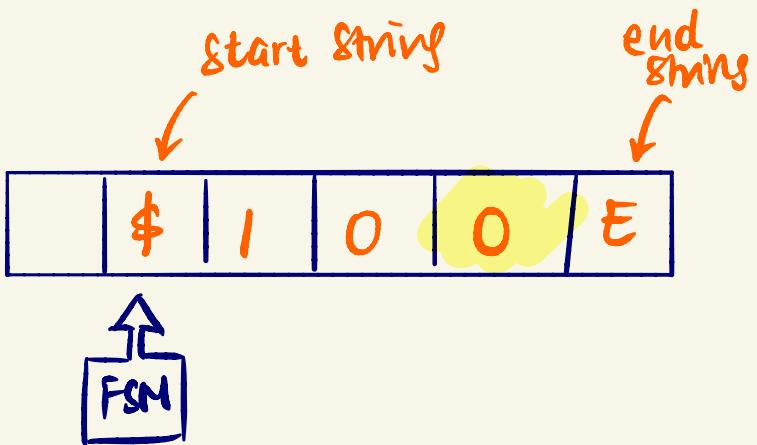
	Input	State		Output state
00 00	L + R	S ₀	TR	S ₁ 01 01
11 00	L + R	S ₀	F	S ₀ 00 00
10 00	L + R	S ₀	H	S ₂ 11 10
01 00	L + R	S ₀	H	S ₂ 11 10
00 01	L + R	S ₁	TR	S ₁ 01 01
10 01	L + R	S ₁	H	S ₂ 11 10
11 01	L + R	S ₁	TL	S ₁ 00 01
11 01	L + R	S ₁	F	S ₁ 00 01
01 01	L + R	S ₁	H	S ₂ 11 10
00 10	any	S ₂		11 10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

4 output bits

4
bits
input
selector



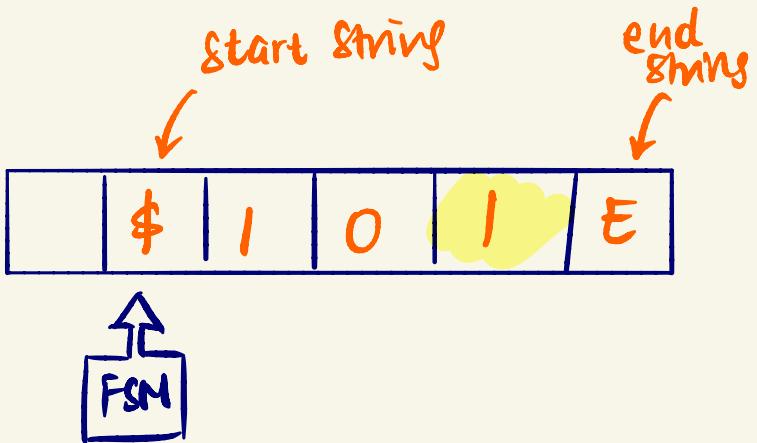
Task: take input N
add 1 in binary
can overwrite \$ if overflow



Sin	Input	Snext	write	move

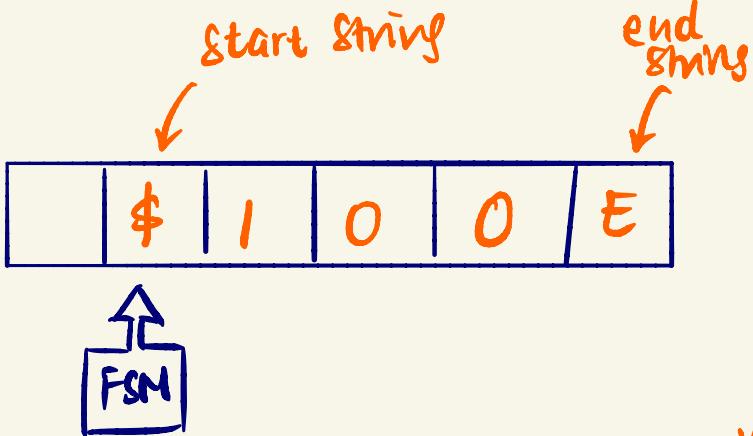
Case 1

Task: take input N
add 1 in binary
can overwrite \$ if overflow



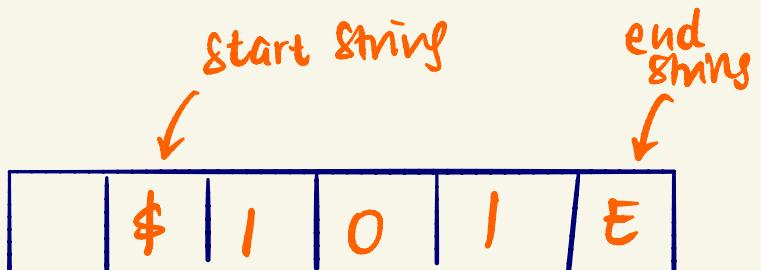
Sin	Input	Snext	write	move

Case 2



Case 1:
LSB is 0

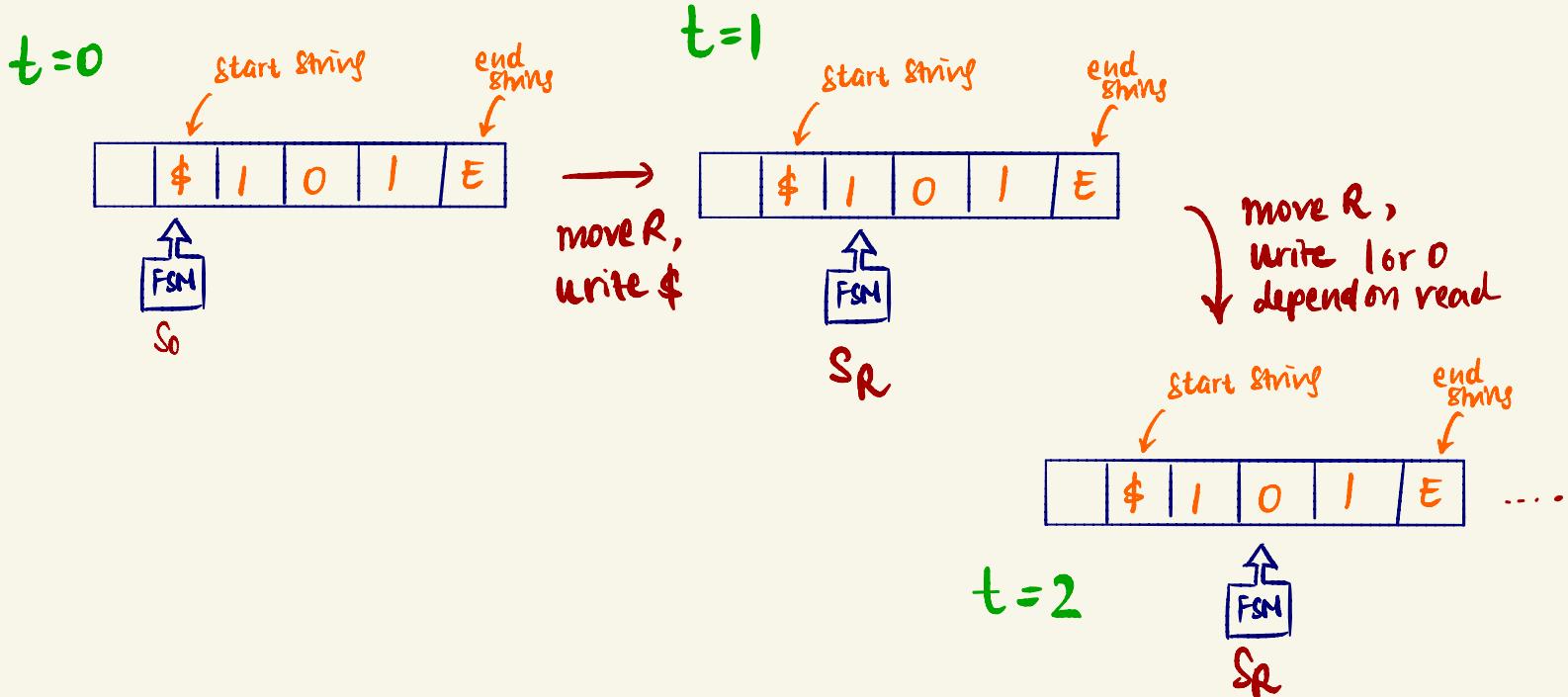
Sin	Input	Snext	write	move
S ₀	\$	S _R	\$	R
S _R	0/1	S _R	0/1	R
S _R	E	S _W	E	L
S _A	0	S _L	1	L
S _L	0/1	S _L	0/1	L
S _L	\$	HALT	\$	R



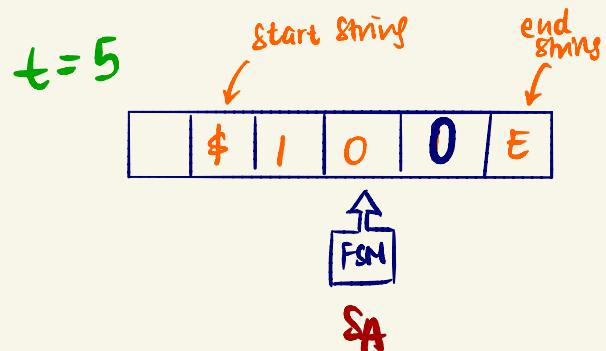
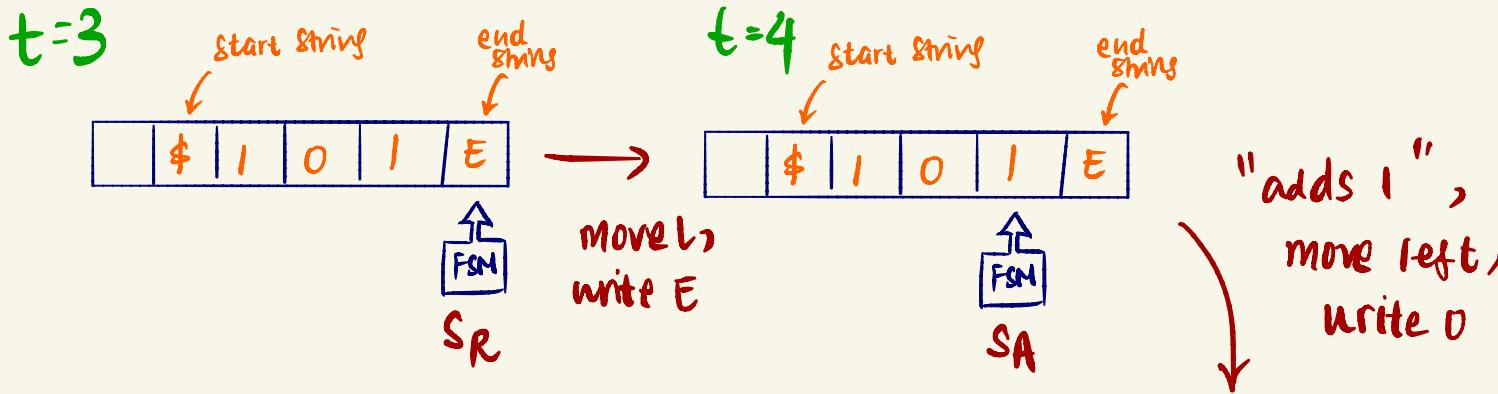
case 2:
LSB is 1

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

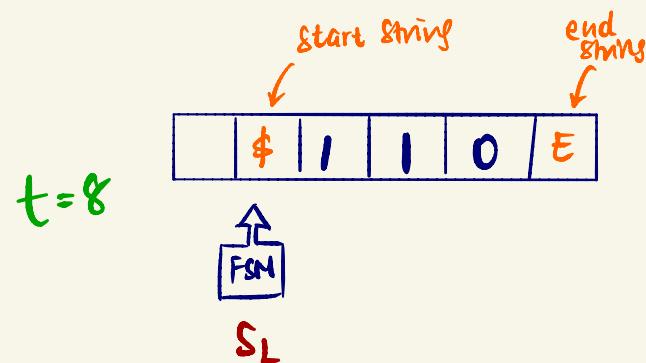
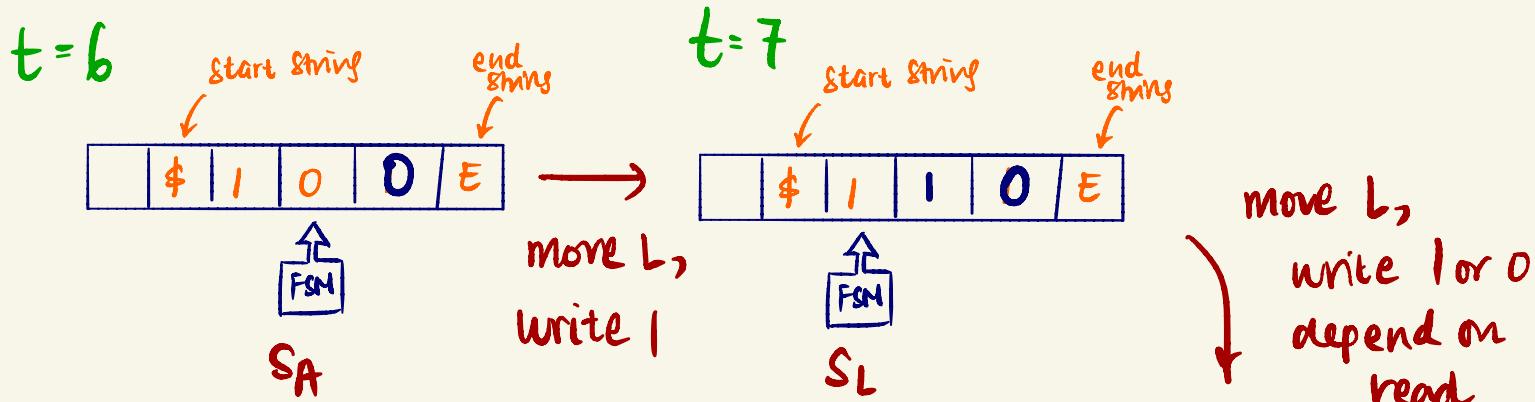
lets run it !



lets run it !

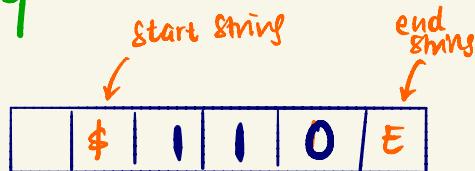


lets run it !



lets run it !

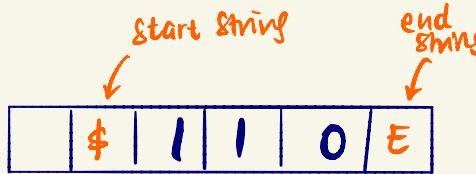
$t=9$



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