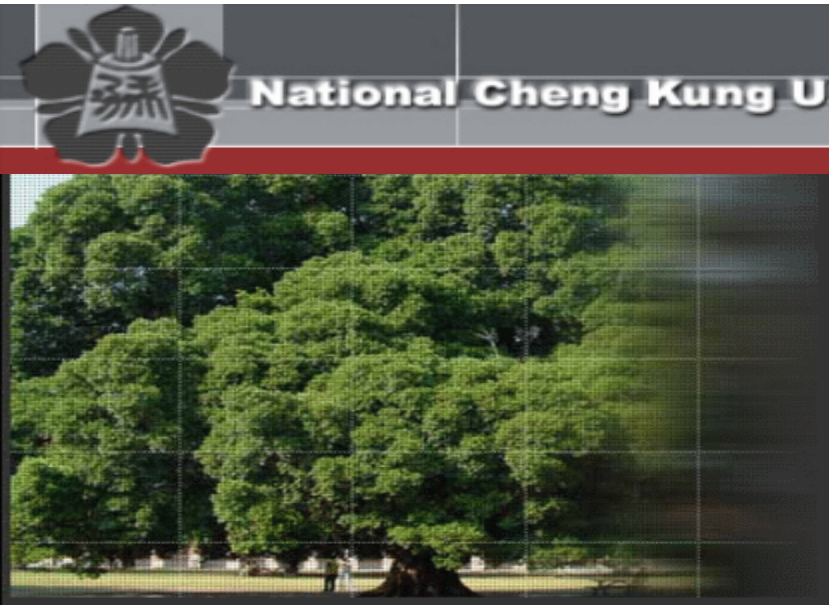
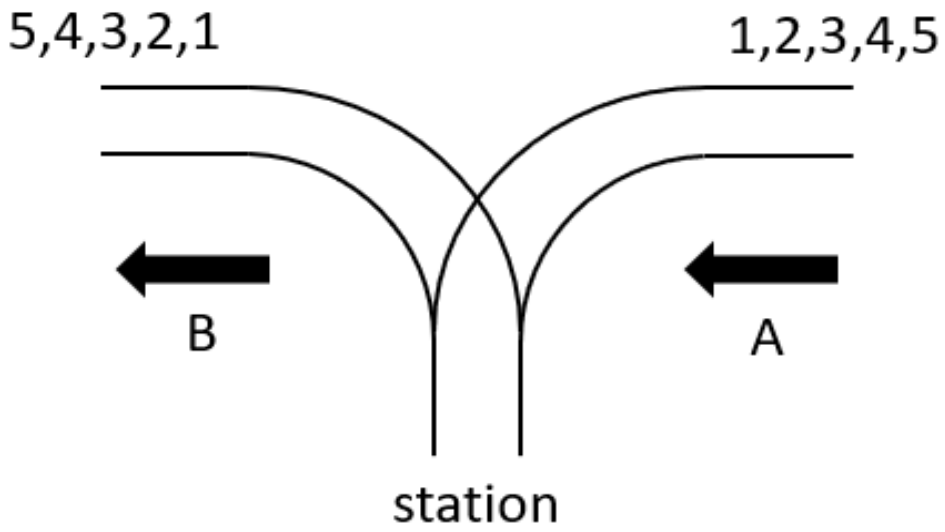


Homework 2 explanation

NCKU CSIE DICLAB

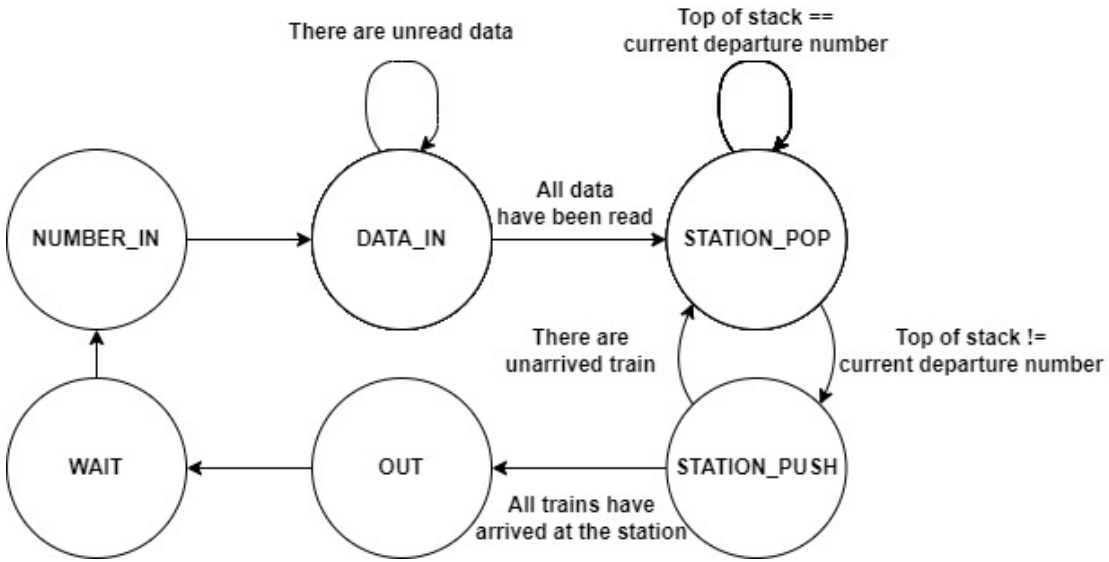
Introduction

- Every train arrives at the station from direction A, and departs the station from direction B.
- The arriving trains will be numbered in ascending order from 1 to *N*. ($3 \leq N \leq 10$)
- Given a departure order *d1*, *d2*, . . . , *dN*, the circuit has to determine if the trains can leave the station in the required order.
- For any train, there is no limit to its arrival time and staying time at the station.



Finite State Machine

1. NUMBER_IN：Read the number of coming trains
2. DATA_IN：Read the sequence of departure order
3. STATION_POP：Examine whether the top of stack is equal to current departure number
4. STATION_PUSH：Push number into the stack
5. OUT：Output determination result
6. WAIT：Reset registers and go back to the initial state (NUMBER_IN)



Data Registers

- num：record the number of coming trains
- index：index of sequence of departure order
- order：data array to store departure order
- station_index：index of the stack
- station：data array that acts as the stack
- sequence_index：count of arrived train

order

index	0	1	2	3	4	5	6	7	8	9
data										

stack

index	0	1	2	3	4	5	6	7	8	9
data										

num: 0
sequence_index: 1

NUMBER_IN & DATA_IN


```
70              NUMBER_IN:begin//read number
71                  num <= data;
72              end
73              DATA_IN:begin//read data
74                  order[index] <= data;
75                  if(index == num - 1) index <= 4'd0;
76                  else index <= index + 1;
77              end
```

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data										

num: 5
sequence_index: 1

STATION_POP


```
78              STATION_POP:begin//compare top with order
79                  if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                      index <= index + 1;
81                      station_index <= station_index - 1;
82                  end
83              end
```

station_index == 0，the stack is empty
go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data										

num: 5
sequence_index: 1

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

Train 1 arrives at the station, push 1 into the stack.

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1									

num: 5
sequence_index: 2

STATION_POP


```
78              STATION_POP:begin//compare top with order
79                  if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                      index <= index + 1;
81                      station_index <= station_index - 1;
82                  end
83              end
```

The top of stack != current departure number

go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1									

num: 5

sequence_index: 2

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

Train 2 arrives at the station, push 2 into the stack.

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2								

num: 5
sequence_index: 3

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack != current departure number

go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2								

num: 5

sequence_index: 3

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

Train 3 arrives at the station, push 3 into the stack.

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2	3							

num: 5
sequence_index: 4

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack != current departure number

go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2	3							

num: 5

sequence_index: 4

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

Train 4 arrives at the station, push 4 into the stack.

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2	3	4						

num: 5
sequence_index: 5

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack == current departure number

Pop an element from stack and stay at STATION_POP state

Increase the index of order array

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2	3	4						

num: 5

sequence_index: 5

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack == current departure number

Pop an element from stack and stay at STATION_POP state

Increase the index of order array

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2	3							

num: 5

sequence_index: 5

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack == current departure number

Pop an element from stack and stay at STATION_POP state

Increase the index of order array

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	2								

num: 5

sequence_index: 5

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack != current departure number
go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1									

num: 5
sequence_index: 5

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

Train 3 arrives at the station, push 3 into the stack.

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	5								

num: 5
sequence_index: 6

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack == current departure number
Pop an element from stack and stay at STATION_POP state
Increase the index of order array

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1	5								

num: 5
sequence_index: 6

STATION_POP


```
78          STATION_POP:begin//compare top with order
79              if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                  index <= index + 1;
81                  station_index <= station_index - 1;
82              end
83          end
```

The top of stack == current departure number
Pop an element from stack and stay at STATION_POP state
Increase the index of order array

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	1									

num: 5
sequence_index: 6

STATION_POP


```
78              STATION_POP:begin//compare top with order
79                  if((station_index > 4'd0) && (station[station_index_minus_one] == order[index]))begin
80                      index <= index + 1;
81                      station_index <= station_index - 1;
82                  end
83              end
```

station_index == 0，the stack is empty
go to STATION_PUSH state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data										

num: 5
sequence_index: 6

STATION_PUSH


```
84              STATION_PUSH:begin//pop ascending sequence
85                  station[station_index] <= sequence_index;
86                  station_index <= station_index + 1;
87                  sequence_index <= sequence_index + 1;
88              end
```

```
40          STATION_PUSH:begin
41              if(sequence_index == num + 1) nextState = OUT;
42              else nextState = STATION_POP;
43          end
```

Push 6 into the stack.
sequence_index == num + 1，
go to OUT state

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	6									

num: 5
sequence_index: 6 + 1

OUT


```
89 ▼            OUT:begin//output result
90                  valid <= 1;
91                  if(index == num) result <= 1;
92              end
```

Pull up valid to 1

index == 5，result = 1

order

index	0	1	2	3	4	5	6	7	8	9
data	4	3	2	5	1					

stack

index	0	1	2	3	4	5	6	7	8	9
data	6									

num: 5

sequence_index: 7

WAIT


```
93 ▼        WAIT:begin//reset register
94              for(i = 0; i < 10; i = i + 1) station[i] <= 4'b1111;
95              valid <= 0;
96              result <= 0;
97              index <= 0;
98              station_index <= 4'd0;
99              sequence_index <= 4'd1;
100         end
```

Pull down valid to 0

Reset registers

order

index	0	1	2	3	4	5	6	7	8	9
data										

stack

index	0	1	2	3	4	5	6	7	8	9
data										

num: 5

sequence_index: 1