61C Midterm Review Session 1!

Boolean Algebra + CALL + FSMs

Wen Cao

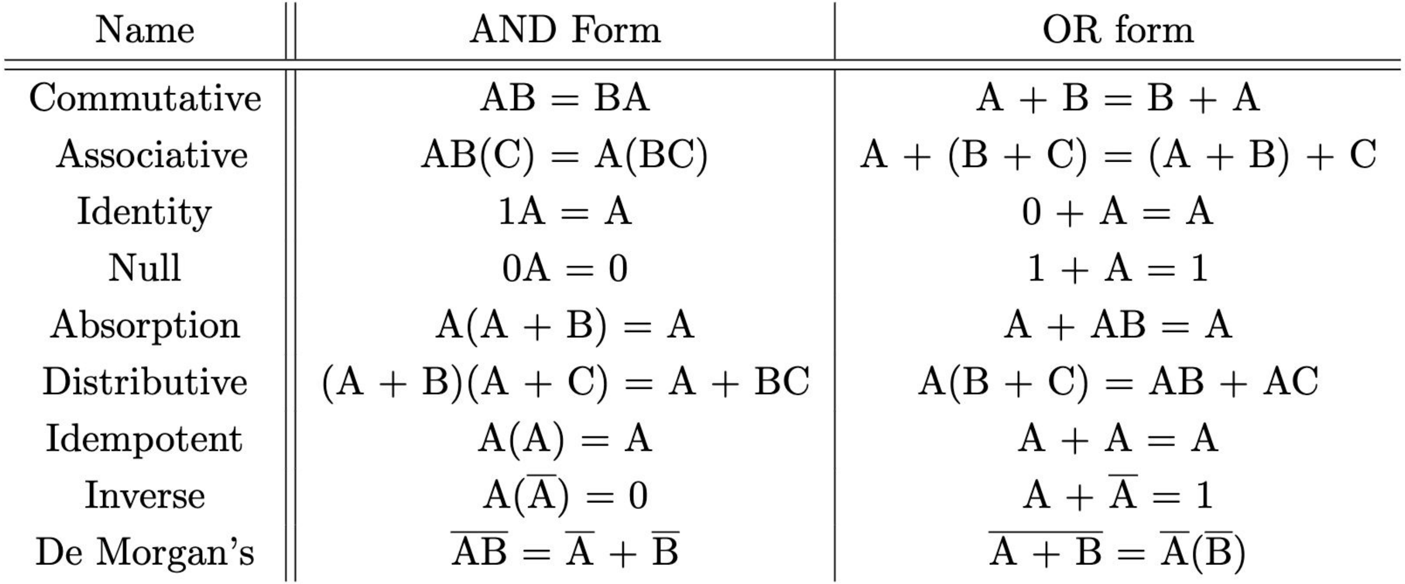
Oct 9, 2024

Boolean Algebra

Boolean Algebra Rules

One more small technique you can use:
!C + AC = !C + A

- Common rules (a lot like normal algebra):



Name	AND Form	OR form
Commutative	$AB = BA$	$A + B = B + A$
Associative	$AB(C) = A(BC)$	$A + (B + C) = (A + B) + C$
Identity	$1A = A$	$0 + A = A$
Null	$0A = 0$	$1 + A = 1$
Absorption	$A(A + B) = A$	$A + AB = A$
Distributive	$(A + B)(A + C) = A + BC$	$A(B + C) = AB + AC$
Idempotent	$A(A) = A$	$A + A = A$
Inverse	$A(\overline{A}) = 0$	$A + \overline{A} = 1$
De Morgan's	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}(\overline{B})$

Boolean Algebra Practice

- Simplify the boolean expression: $\overline{w} \cdot \overline{(wxyz)}$

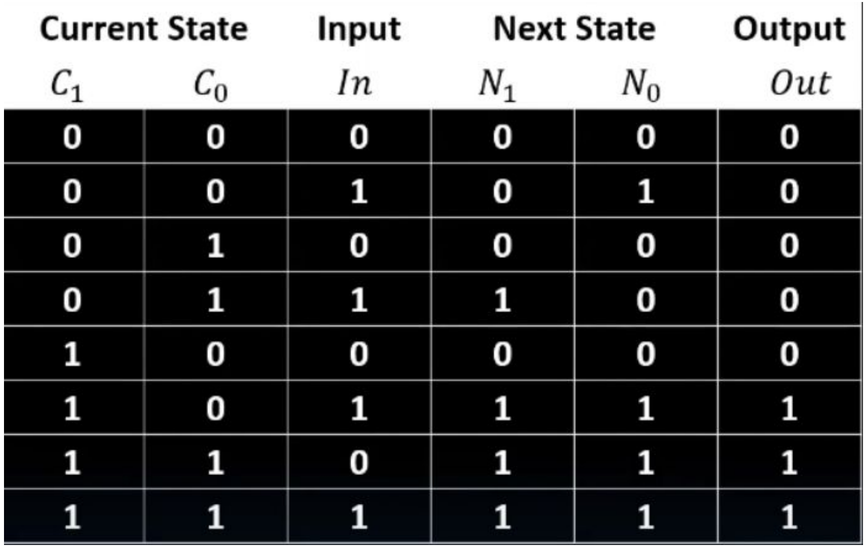
- Simplify the boolean expression: $\overline{w} \cdot \overline{(wxyz)}$

$$\begin{aligned}
\overline{w} \cdot \overline{(wxyz)} &= \overline{w} \cdot (\overline{w} + \overline{x} + \overline{y} + \overline{z}) \text{ (De Morgan's Law)} \\
&= \overline{w} \cdot \overline{w} + \overline{w}(\overline{x} + \overline{y} + \overline{z}) \\
&= \overline{w} + \overline{w}(\overline{x} + \overline{y} + \overline{z}) \\
&= \overline{w}(1 + \overline{x} + \overline{y} + \overline{z}) \\
&= \overline{w}
\end{aligned}$$

Sum of Products

Write a boolean expression that represents N_1 as a **sum of products in terms of C_1, C_0, In**.

Simply your answer.

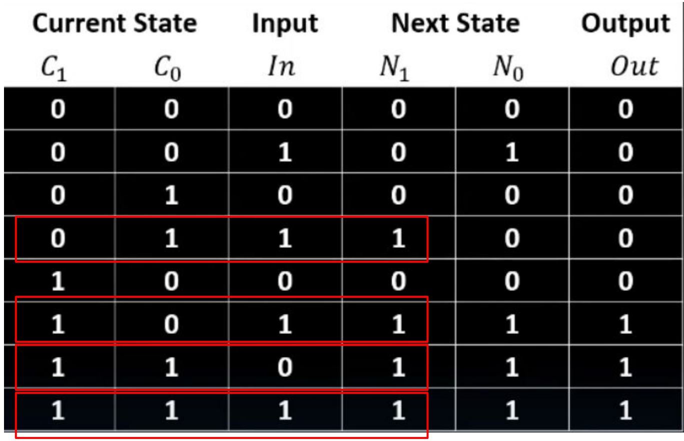


Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Sum of Products

Write a boolean expression that represents N_1 as a **sum of products**. Simply your answer.

$!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In}$



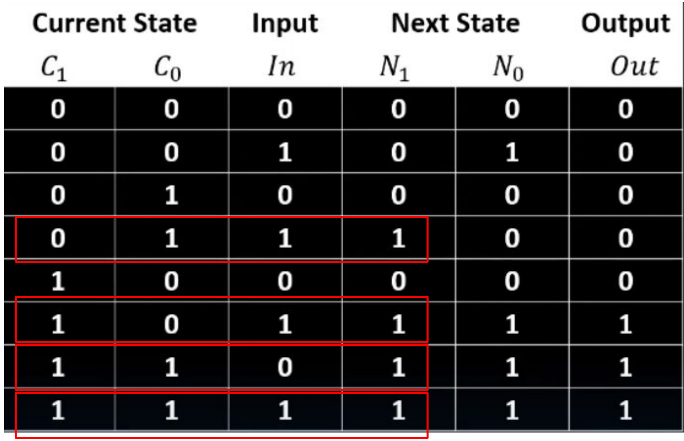
Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Sum of Products

Write a boolean expression that represents N$_1$ as a **sum of products**. Simply your answer.

$$!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In}$$

$$=!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In+C_1C_0In+C_1C_0In}$$



Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

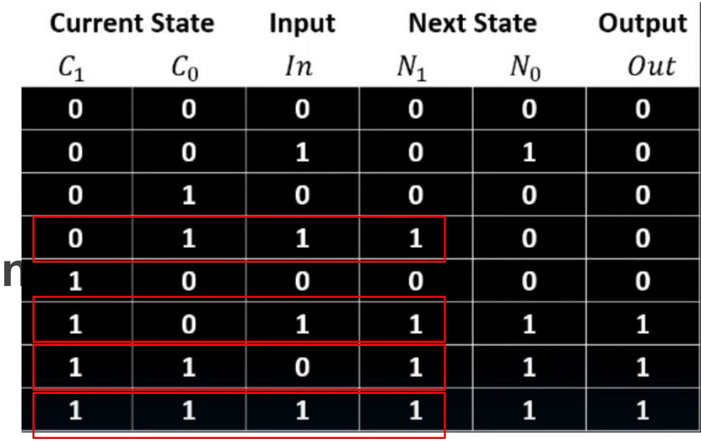
Sum of Products

Write a boolean expression that represents N_1 as a **sum of products**. Simply your answer.

$!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In}$

$=!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In+C_1C_0In+C_1C_0In}$

$=(!C_1C_0In+\mathbf{C_1C_0In})+(C_1!C_0In+\mathbf{C_1C_0In})+(C_1C_0!In+\mathbf{C_1C_0In}$



Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Sum of Products

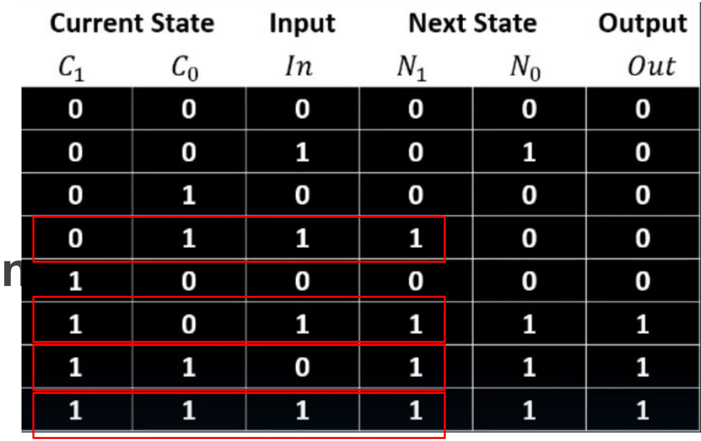
Write a boolean expression that represents N_1 as a **sum of products**. Simply your answer.

$$!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In}$$

$$=!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In+C_1C_0In+C_1C_0In}$$

$$=(!C_1C_0In+\mathbf{C_1C_0In})+(C_1!C_0In+\mathbf{C_1C_0In})+(C_1C_0!In+\mathbf{C_1C_0In}$$

$$=(!C_1+C_1)C_0In+(!C_0+C_0)C_1In+(!In+In)C_1C_0$$



Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

Sum of Products

Write a boolean expression that represents N_1 as a **sum of products**. Simply your answer.

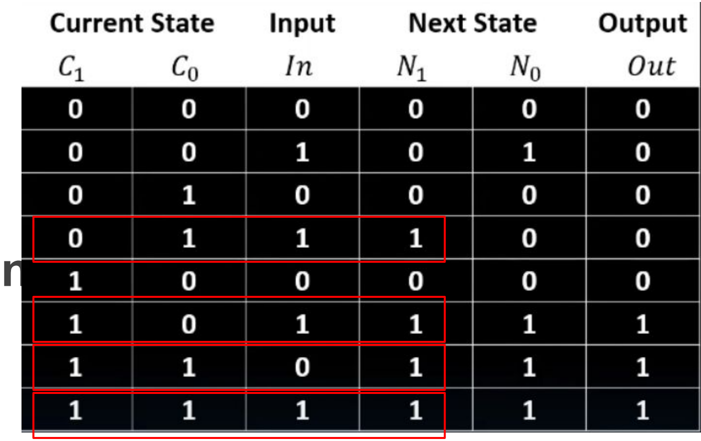
$$!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In}$$

$$=!C_1C_0In+C_1!C_0In+C_1C_0!In+\mathbf{C_1C_0In+C_1C_0In+C_1C_0In}$$

$$=(!C_1C_0In+\mathbf{C_1C_0In})+(C_1!C_0In+\mathbf{C_1C_0In})+(C_1C_0!In+\mathbf{C_1C_0In}$$

$$=(!C_1+C_1)C_0In+(!C_0+C_0)C_1In+(!In+In)C_1C_0$$

$$=C_0In+C_1In+C_1C_0$$



Current State		Input	Next State		Output
C_1	C_0	In	N_1	N_0	Out
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

    [distributive property]

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

    [distributive property] = A * C + B * C + !A * !B + !A * C

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] = **A** * **C** + B * C + !A * !B + **!A** * **C**

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] $= \mathbf{A} * \mathbf{C} + \boxed{B * C + !A * !B} + \mathbf{!A} * \mathbf{C}$

[drag out common C term] $= \mathbf{C} * (\mathbf{A + !A}) + \boxed{B * C + !A * !B}$

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] $= \mathbf{A} * \mathbf{C} + \boxed{B * C + !A * !B} + \mathbf{!A} * \mathbf{C}$

[drag out common C term] $= \mathbf{C} * (\mathbf{A + !A}) + \boxed{B * C + !A * !B}$

= C + B * C + !A * !B

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] = **A** * **C** + $\boxed{B * C + !A * !B}$ + **!A** * **C**

[drag out common C term] = **C** * (**A + !A**) + $\boxed{B * C + !A * !B}$

= C + B * C + !A * !B

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] $= \mathbf{A} * \mathbf{C} + \boxed{B * C + !A * !B} + \mathbf{!A} * \mathbf{C}$

[drag out common C term] $= \mathbf{C} * (\mathbf{A + !A}) + \boxed{B * C + !A * !B}$

$= C + B * C + !A * !B$

$= C * (1 + B) + !A * !B$

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] = **A** * **C** + $\boxed{B * C + !A * !B}$ + **!A** * **C**

[drag out common C term] = **C** * (**A + !A**) + $\boxed{B * C + !A * !B}$

= C + B * C + !A * !B

= C * (1 + B) + !A * !B

= C + !A!B

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] $= \mathbf{A} * \mathbf{C} + \boxed{B * C + !A * !B} + \mathbf{!A} * \mathbf{C}$

[drag out common C term] $= \mathbf{C} * (\mathbf{A + !A}) + \boxed{B * C + !A * !B}$

$= C + B * C + !A * !B$

$= C * (1 + B) + !A * !B$

$= C + !A!B$

Done?

More Boolean Algebra

- Simplify the boolean expression: C * (A + B) + !A * (!B + C) (hint: 3 operators)

[distributive property] $= \mathbf{A} * \mathbf{C} + \boxed{B * C + !A * !B} + \mathbf{!A} * \mathbf{C}$

[drag out common C term] $= \mathbf{C} * (\mathbf{A + !A}) + \boxed{B * C + !A * !B}$

$= C + B * C + !A * !B$

$= C * (1 + B) + !A * !B$

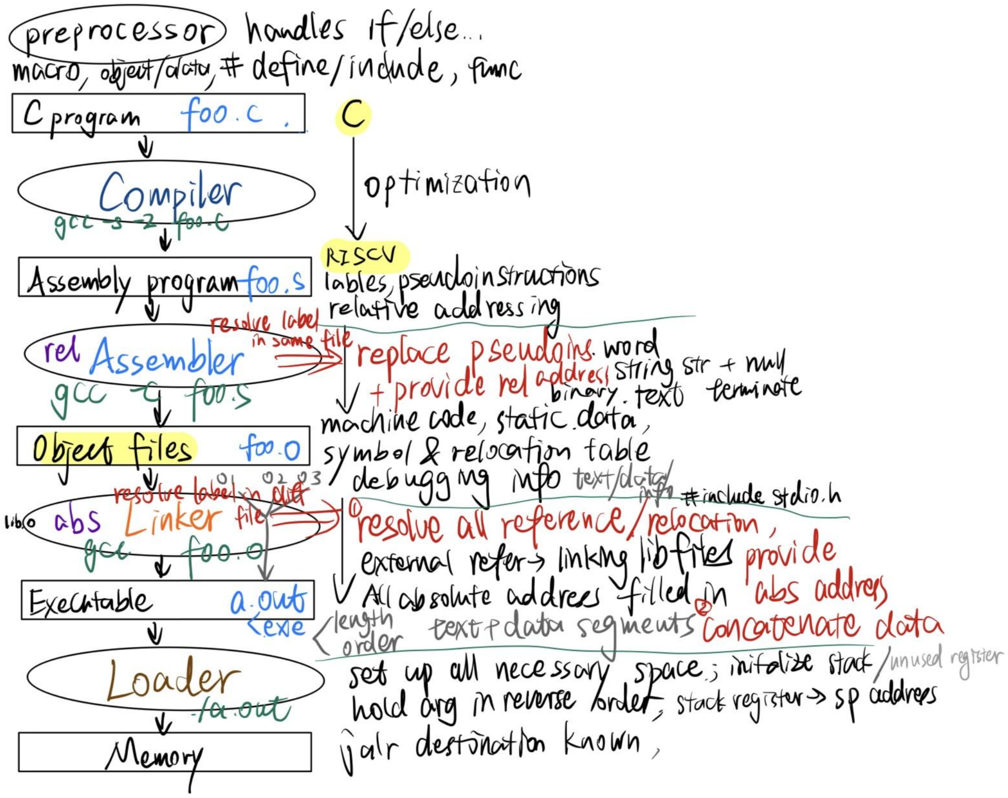
$= C + !A!B$

Done? Oooops nope, C + !A*!B has 4 operators :((((

C + !(A+B)

CALL

CALL



preprocessor handles if/else...
macro, object/data, # define/include, func

C program foo.c

C

optimization

RISCV

Compiler
gcc -s -2 foo.c

Assembly program foo.s
lables, pseudoinstructions
relative addressing

rel Assembler
resolve label in same file
gcc -c foo.s

replace pseudoins. word
string str + null terminate
+ provide rel address binary .text
machine code, static data,
symbol & relocation table
debugging info text/data/info

Object files foo.o

01 02 03

lib.o abs Linker
resolve label in diff file
gcc foo.o

#include stdio.h

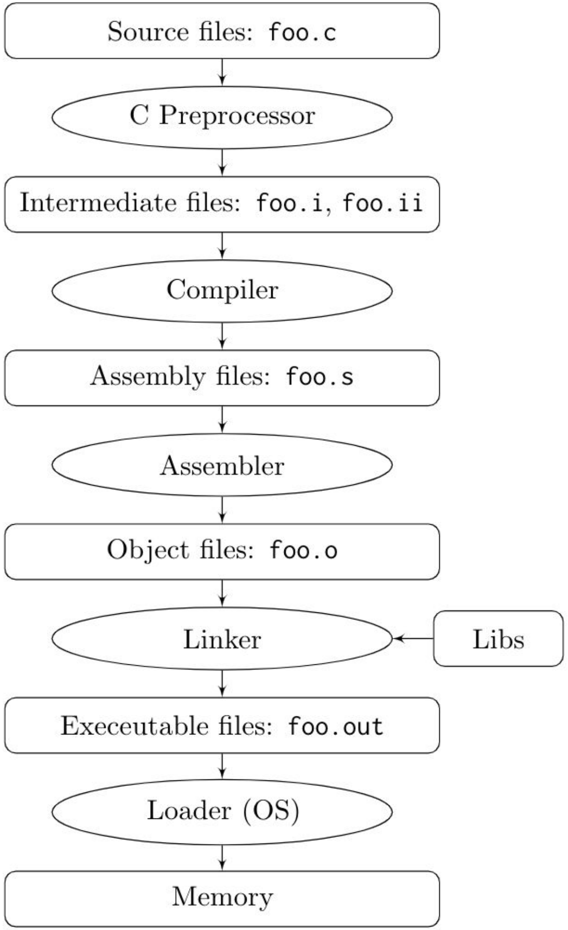
① resolve all reference/relocation,
external refer→ linking lib files provide abs address
All absolute address filled in
② concatenate data
text + data segments
length order

Executable a.out
<exe

Loader
./a.out

set up all necessary space; initalize stack /unused register
hold arg in reverse order, stack register→ sp address
jalr destination known,

Memory



Source files: foo.c

C Preprocessor

Intermediate files: foo.i, foo.ii

Compiler

Assembly files: foo.s

Assembler

Object files: foo.o

Linker

Libs

Exeecutable files: foo.out

Loader (OS)

Memory

Compiler / Compilation

C or other low level language → RISC-V or other assembly language

This does not happen for high level languages like Python

Optimizations occur at this level (gcc will make your code better and faster)

Output will contain:

Labels, pseudoinstructions, relative addressing

Assembler / Assembly

RISC-V or other assembly language → Object file

Assembler must take two passes over the assembly code

Generates two tables for future use:

Symbol Table: labels and their relative addresses where they’re defined

Relocation table: indicates parts of the code that will need to be calculated and changed later. (external labels, data in static section, etc.)

2-Pass Assembler Example

file_a.s

```
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing

Relocation Table for File A

Label Name	State

First Pass

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start

Relocation Table for File A

Label Name	State

First Pass

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start

Relocation Table for File A

Label Name	State
loop	?

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start

Relocation Table for File A

Label Name	State
loop	?

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start

Relocation Table for File A

Label Name	State
loop	?
malloc	?

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + (4 * 11)

Relocation Table for File A

Label Name	State
loop	?
malloc	?

2-Pass Assembler Example

```
file_a.s

00 func_name:

       addi sp, sp, -4

01     beq a0, x0, loop

       # <8 instructions here>

10     jal ra, malloc

11 loop: bneq t0, x0, done

       # <4 instructions here>

15     j loop

16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + (4 * 11)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

2-Pass Assembler Example

First Pass

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + (4 * 11)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j loop
16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + (4 * 11)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

2-Pass Assembler Example

```
file_a.s

00 func_name:

       addi sp, sp, -4

01     beq a0, x0, loop

       # <8 instructions here>

10     jal ra, malloc

11 loop: bneq t0, x0, done

       # <4 instructions here>

15     j ~~loop~~ fa_start + 44

16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + (4 * 11)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

2-Pass Assembler Example

```
file_a.s

00 func_name:

       addi sp, sp, -4

01     beq a0, x0, loop

       # <8 instructions here>

10     jal ra, malloc

11 loop: bneq t0, x0, done

       # <4 instructions here>

15     j ~~loop~~ fa_start + 44

16 done: ret
```

First Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

Second Pass

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, loop
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j ~~loop~~ fa_start + 44
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	?
malloc	?
done	?

2-Pass Assembler Example

Second Pass

```
file_a.s

00 func_name:
       addi sp, sp, -4
01     beq a0, x0, ~~loop~~
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j ~~loop~~ fa_start + 44
16 done: ret
```

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	fa_start + 44
malloc	?
done	?

2-Pass Assembler Example

```
file_a.s

00 func_name:
       addi sp, sp, -4
01     beq a0, x0, ~~loop~~
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, done
       # <4 instructions here>
15     j ~~loop~~ fa_start + 44
16 done: ret
```

Second Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	fa_start + 44
malloc	?
done	?

2-Pass Assembler Example

```
file_a.s
00 func_name:
        addi sp, sp, -4
01      beq a0, x0, ~~loop~~
        # <8 instructions here>
10      jal ra, malloc
11 loop: bneq t0, x0, ~~done~~
        # <4 instructions here>
15      j ~~loop~~ fa_start + 44
16 done: ret
```

Second Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	fa_start + 44
malloc	?
done	fa_start + 64

2-Pass Assembler Example

```
file_a.s
00 func_name:
       addi sp, sp, -4
01     beq a0, x0, ~~loop~~
       # <8 instructions here>
10     jal ra, malloc
11 loop: bneq t0, x0, ~~done~~
       # <4 instructions here>
15     j ~~loop~~ fa_start + 44
16 done: ret
```

Second Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	fa_start + 44
malloc	?
done	fa_start + 64

2-Pass Assembler Example

Q: what does it mean that malloc's state in the relocation table is still "?"

A: this tells us that malloc wasn't defined in File A; thus it must be defined in a separate user file OR in a separate library. Both of these are considered "external references" and should be resolved in the linking stage.

Second Pass

Symbol Table for File A

Label Name	Relative Addressing
func_name	fa_start
loop	fa_start + 44
done	fa_start + (4 * 16)

Relocation Table for File A

Label Name	State
loop	fa_start + 44
malloc	?
done	fa_start + 64

Linker / Linking

Multiple object files → .exe file (executable)

Categorizes code segments from object files and puts them together

(the linker decides on the order)

Resolves all references

- References from user tables are resolved using symbol tables
- External references resolved with static/dynamic linking of lib files
- Goes through all relocation table entries

<u>All</u> absolute addresses are filled in!

Loader / Loading

.exe file → Puts the program on memory and gets it ready to run.

Sets up necessary space on memory for text and data.

Initializes stack to hold arguments from the user.

Initializes registers

Tips

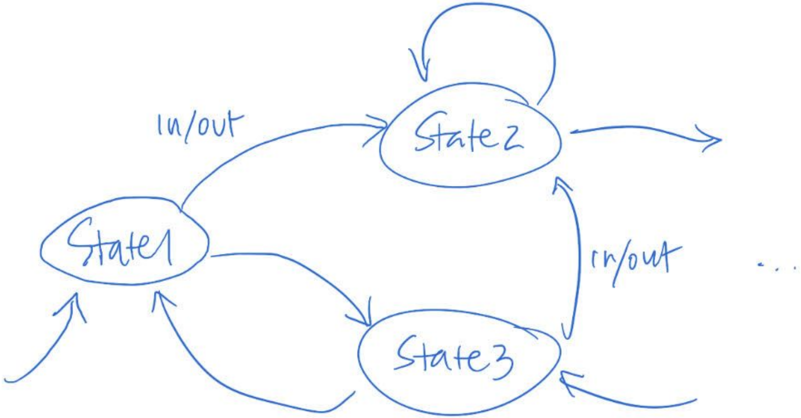
Review the homework 4 CALL question!

It’s randomized! Try different variances!

FSM

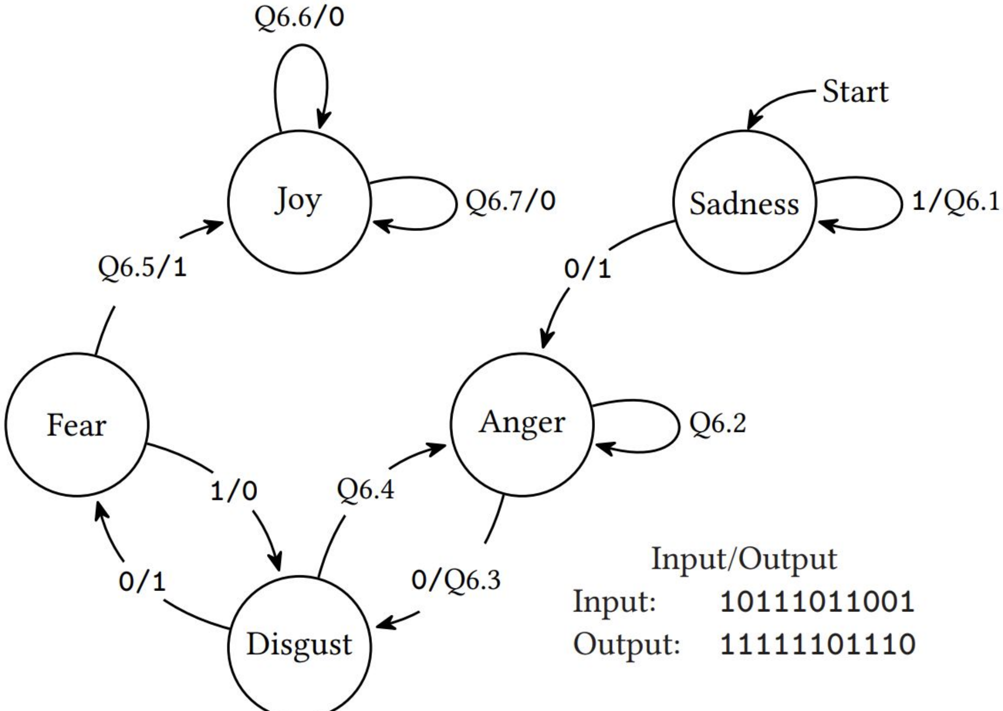
FSM (Finite State Machine)

- A Finite State Machine is a simplified version of a computer.
- It takes in as input a sequence of characters, and outputs a sequence of characters.
- Visually, we represent a FSM by a number of states, plus transitions (arrows) between the states
  - One state is denoted the start state
  - Each state has one arrow exiting it for every possible input
  - Each arrow has a sign "X/Y" on it. Intuitively, we follow this arrow if our input is X, and we output Y when we take this arrow.



FSM (Su24) Tip: be patient :)))

Uh oh! Riley’s emotions are scrambled, and she’s forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.



Input/Output

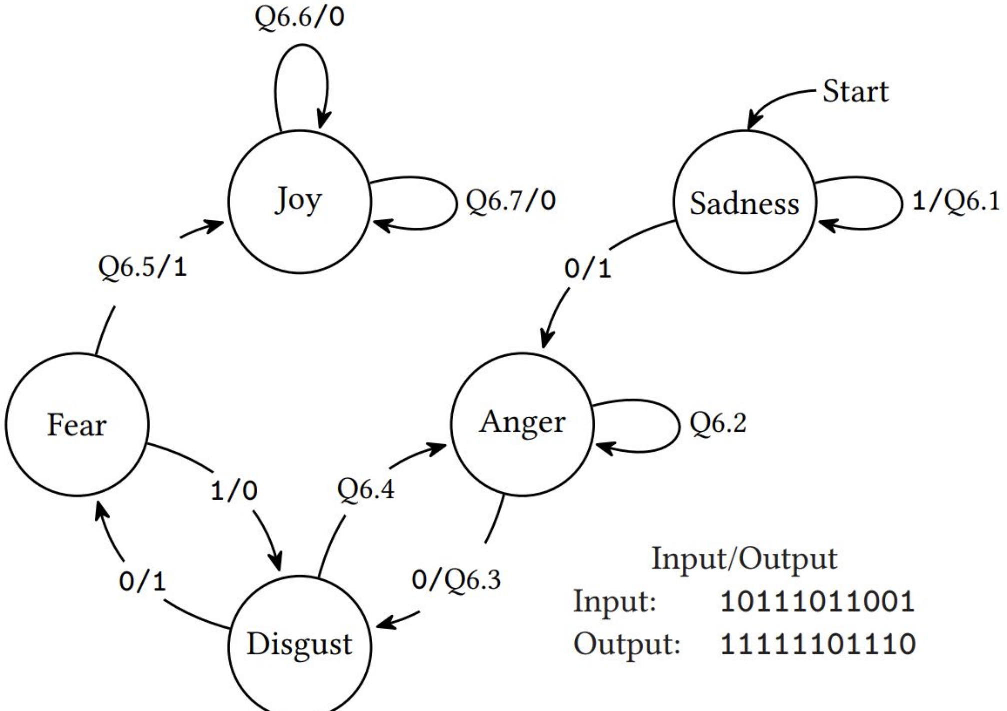
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.1:



Input/Output

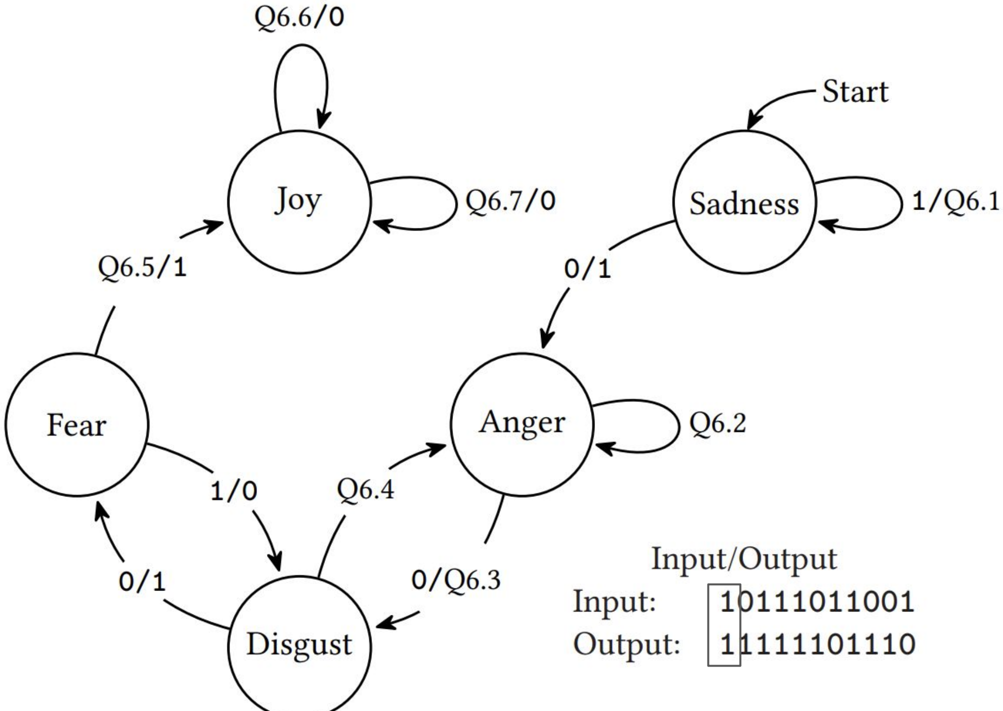
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.1:



Input/Output

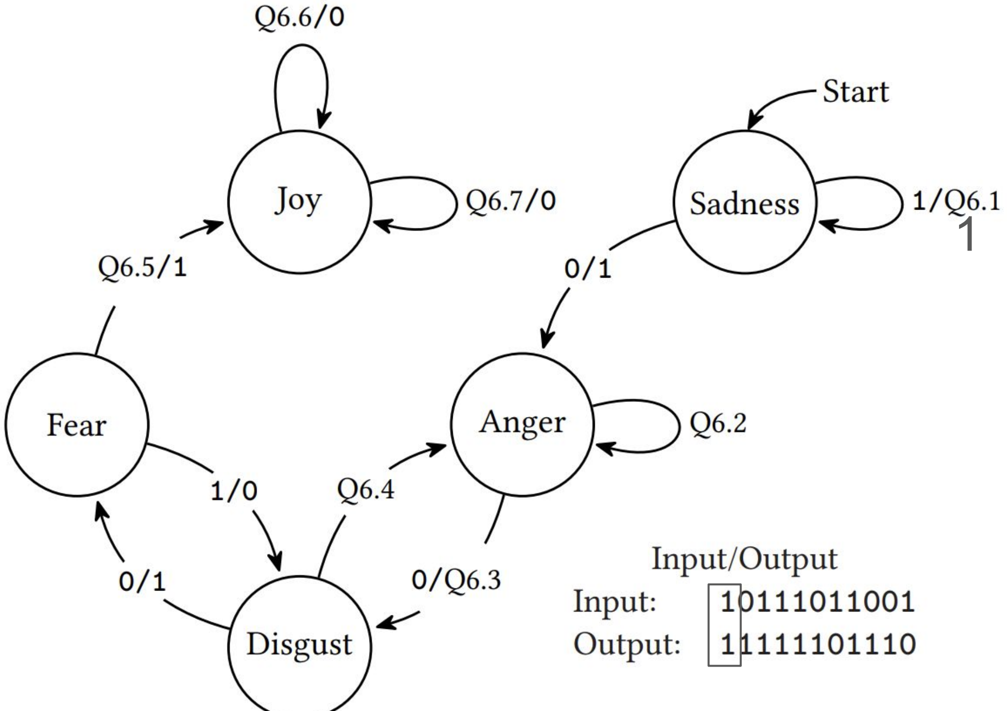
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.1: 1



Input/Output

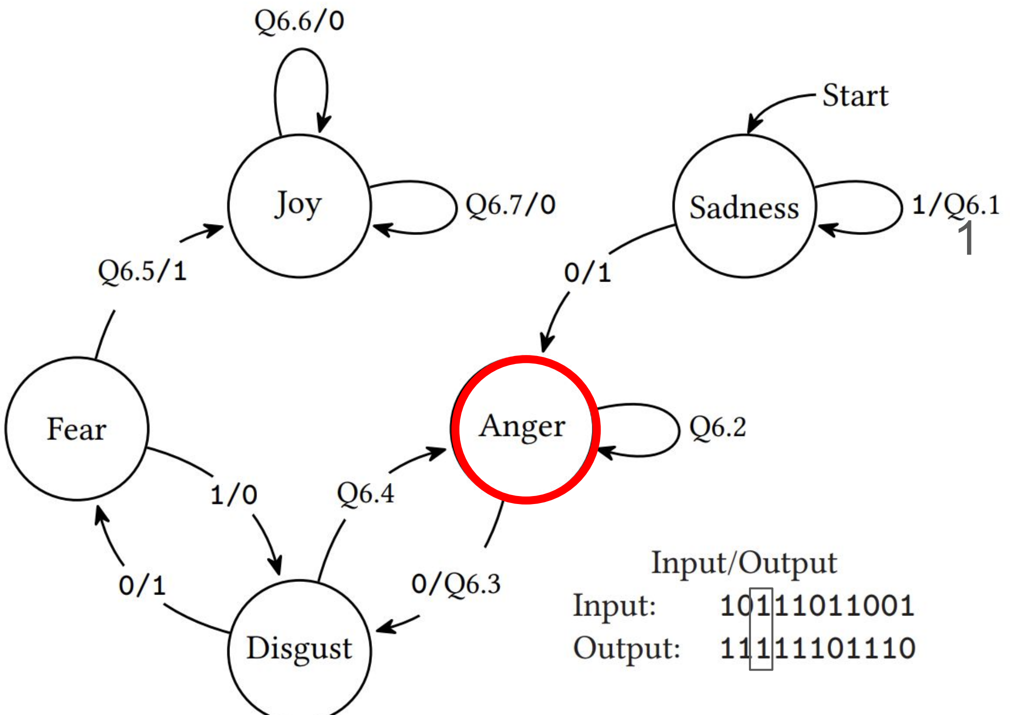
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

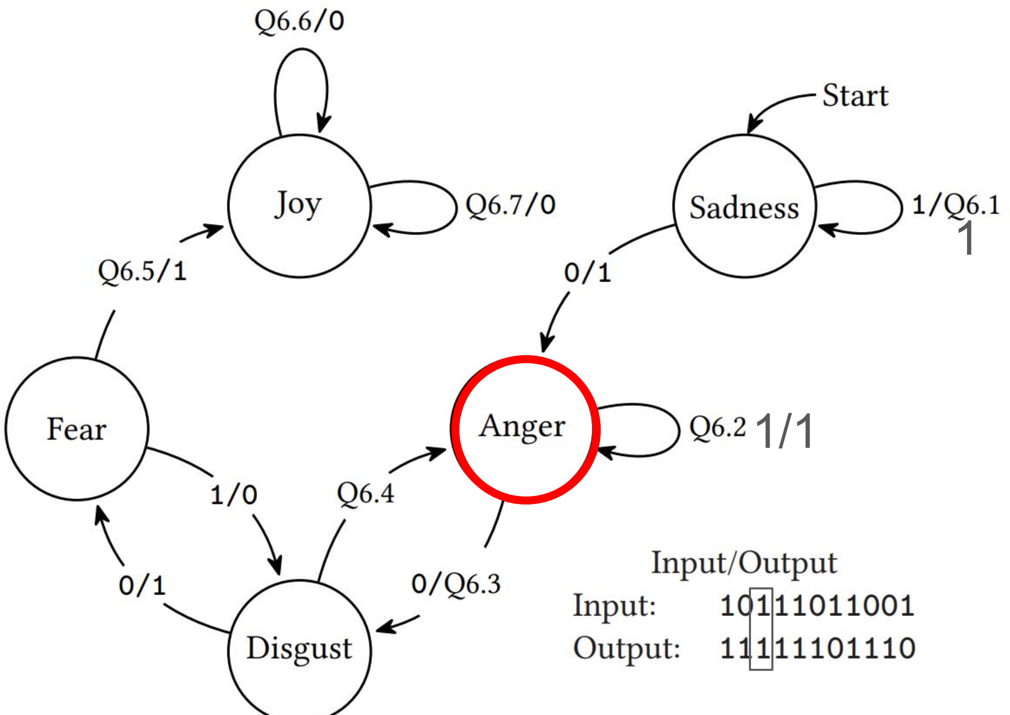
Q6.2:



FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.2: 1/1



Input/Output

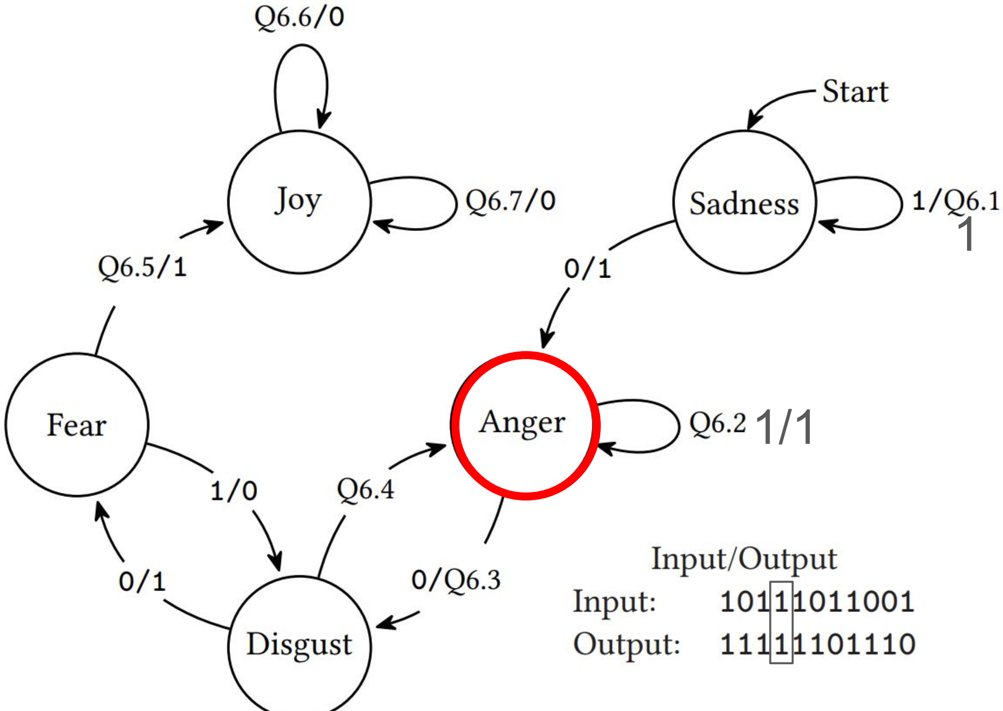
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.2: 1/1



Input/Output

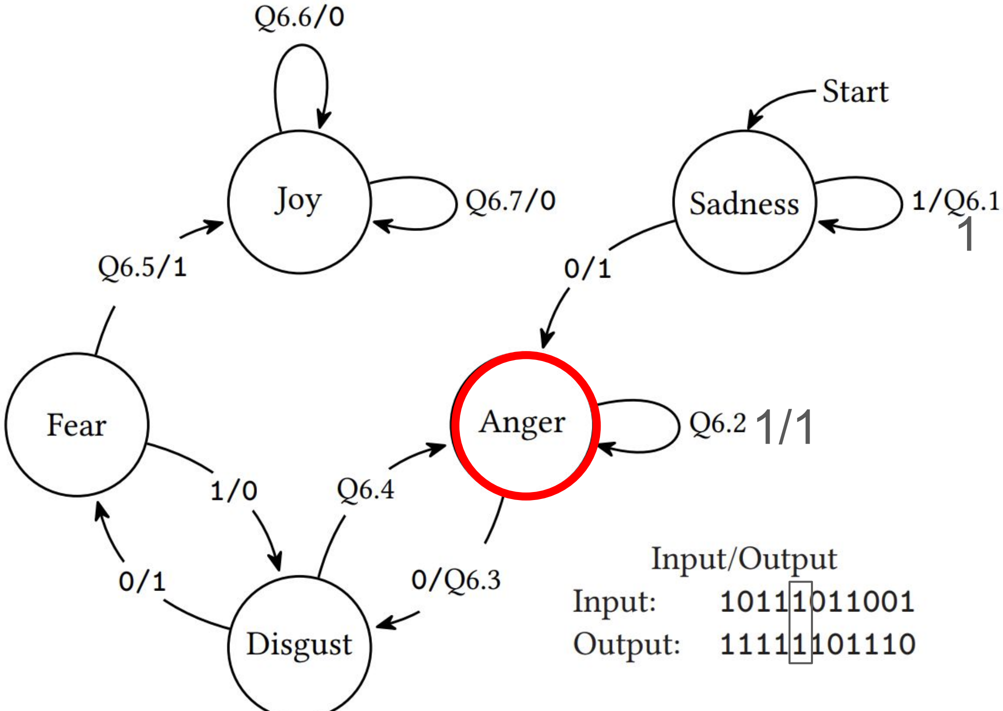
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.2: 1/1



Input/Output

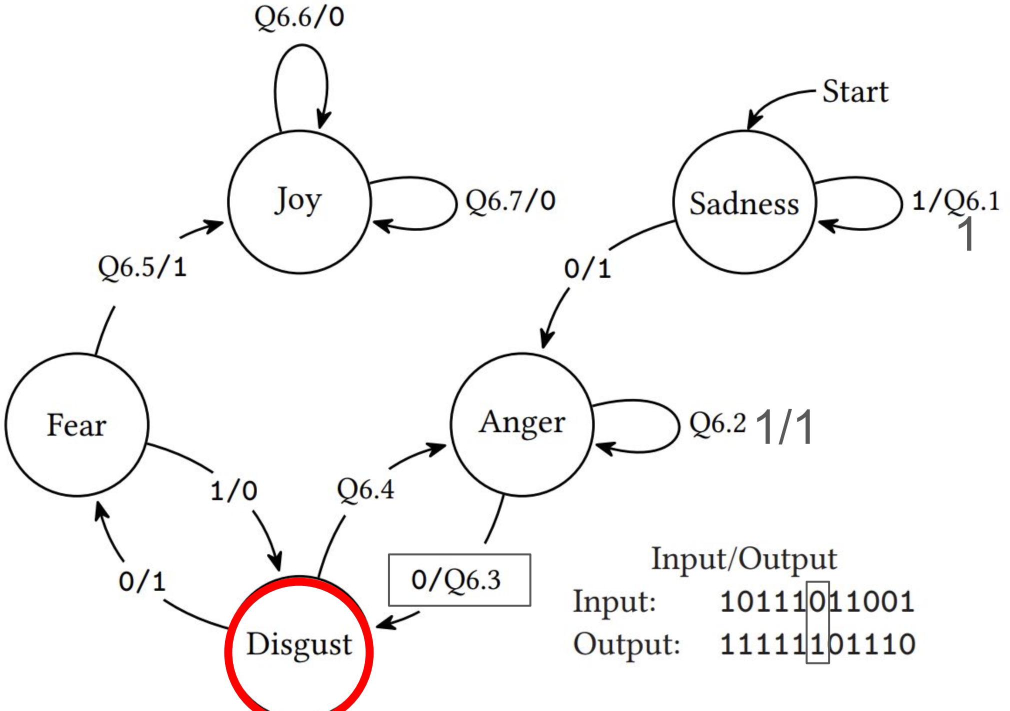
Input: 101110011001

Output: 111111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.3:



Input/Output

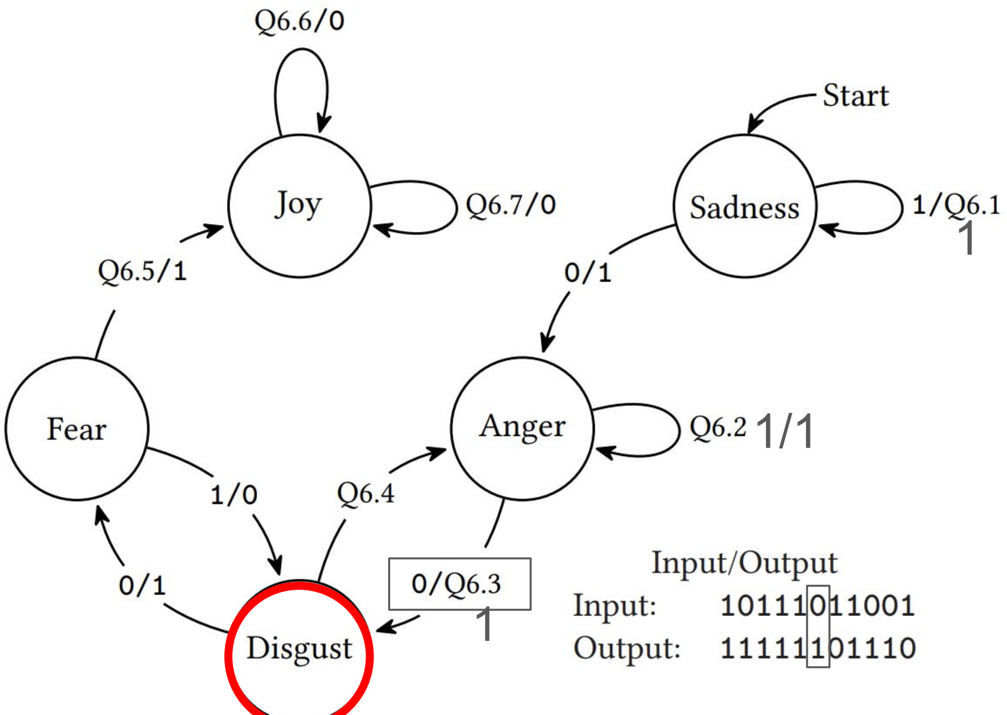
Input: 1011101̲1001

Output: 1111110̲1110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.3: 1



Input/Output

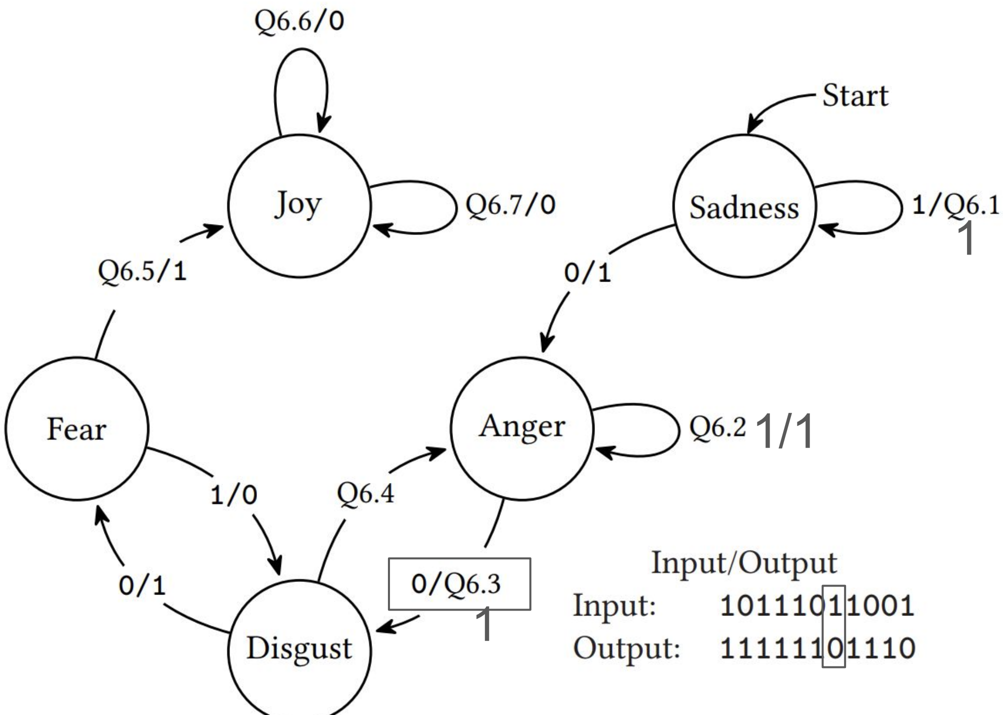
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.3: 1



Input/Output

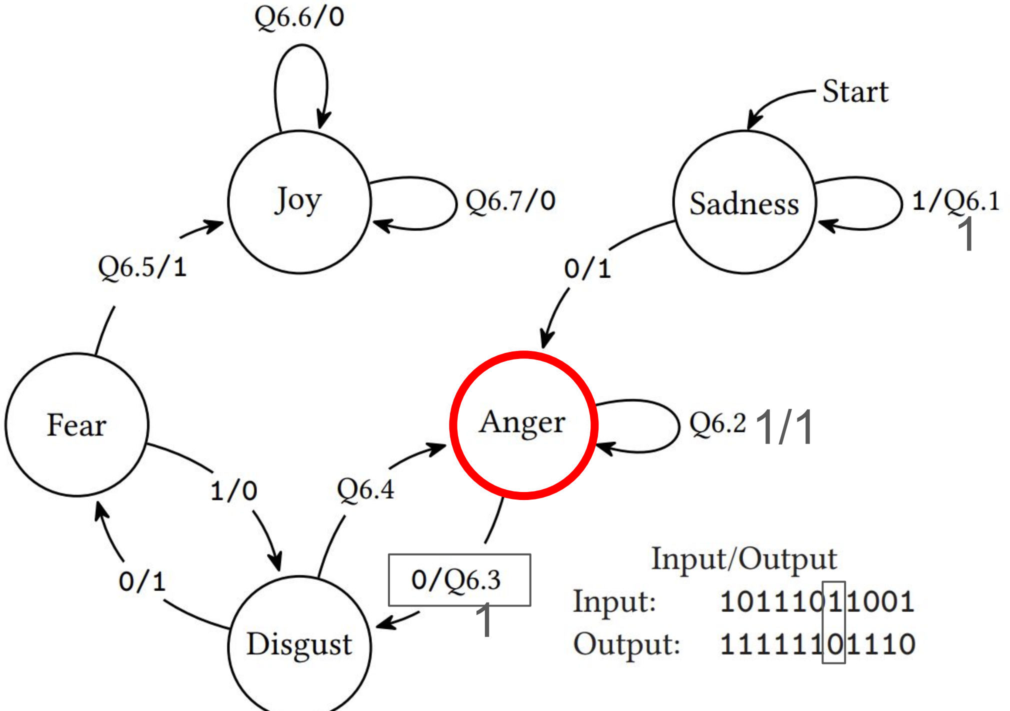
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.4: 1/0



Input/Output

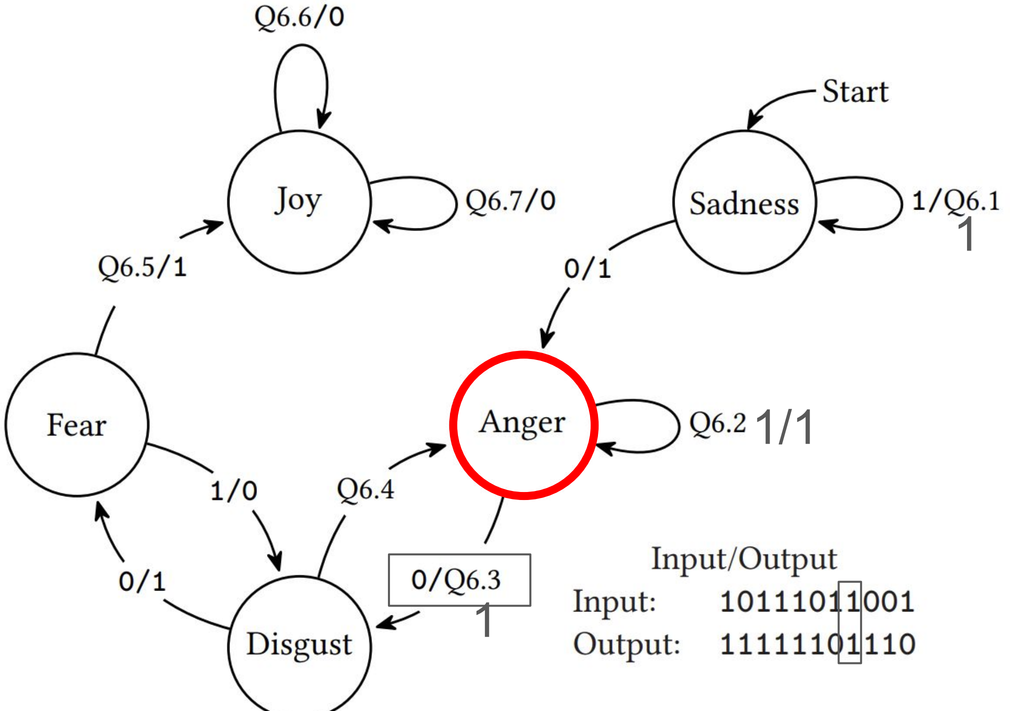
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.4: 1/0



Input/Output

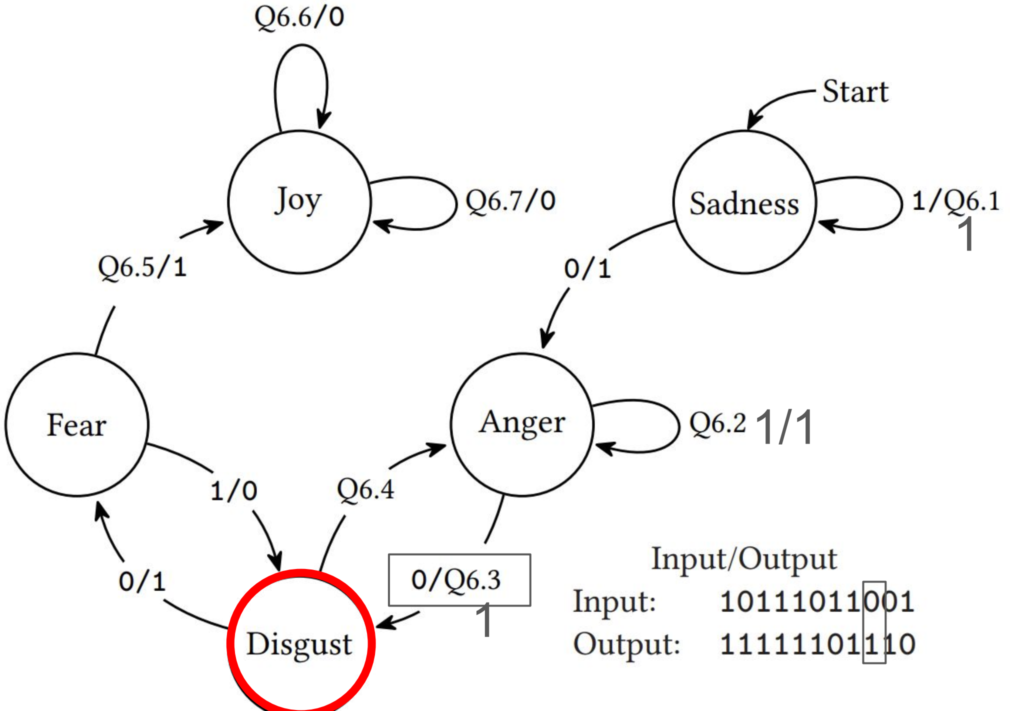
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.4: 1/0



Input/Output

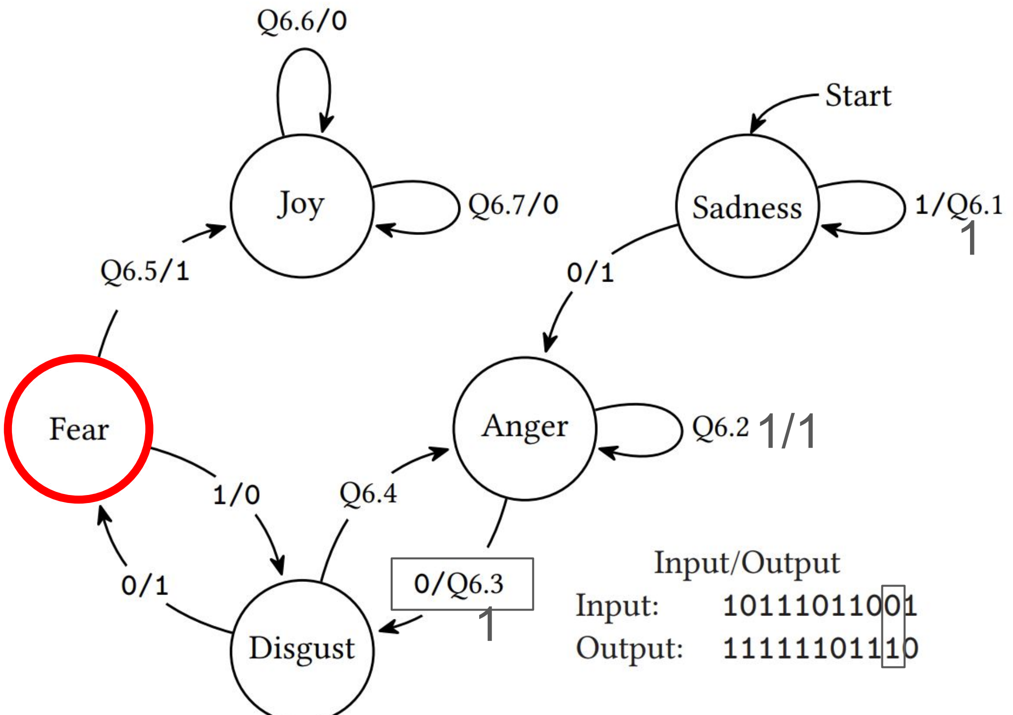
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.4: 1/0



Input/Output

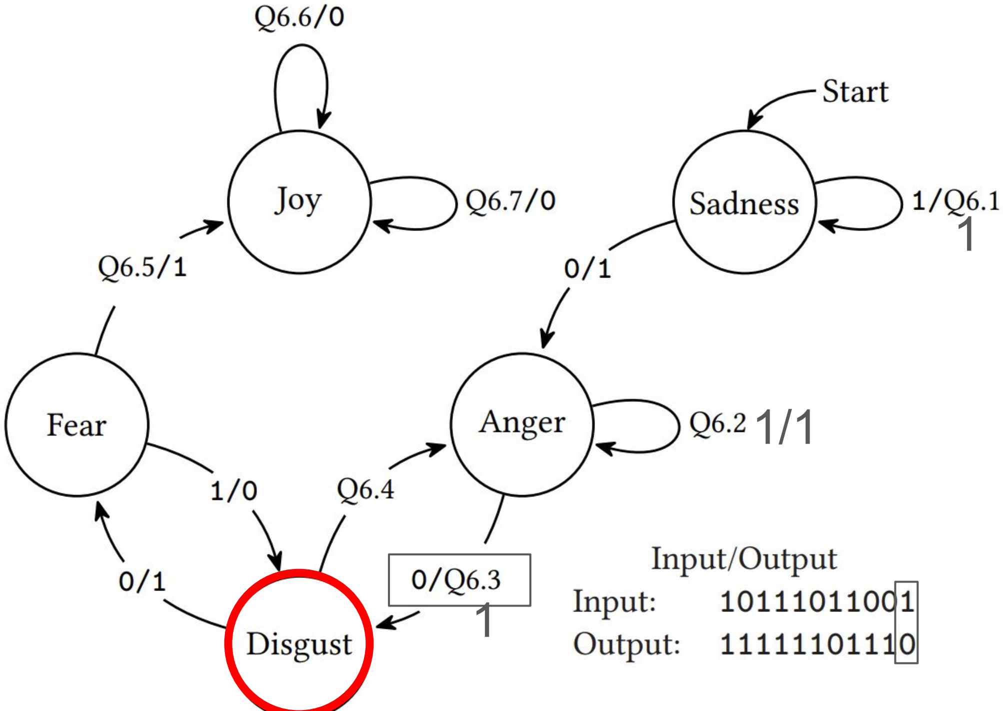
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.4: 1/0



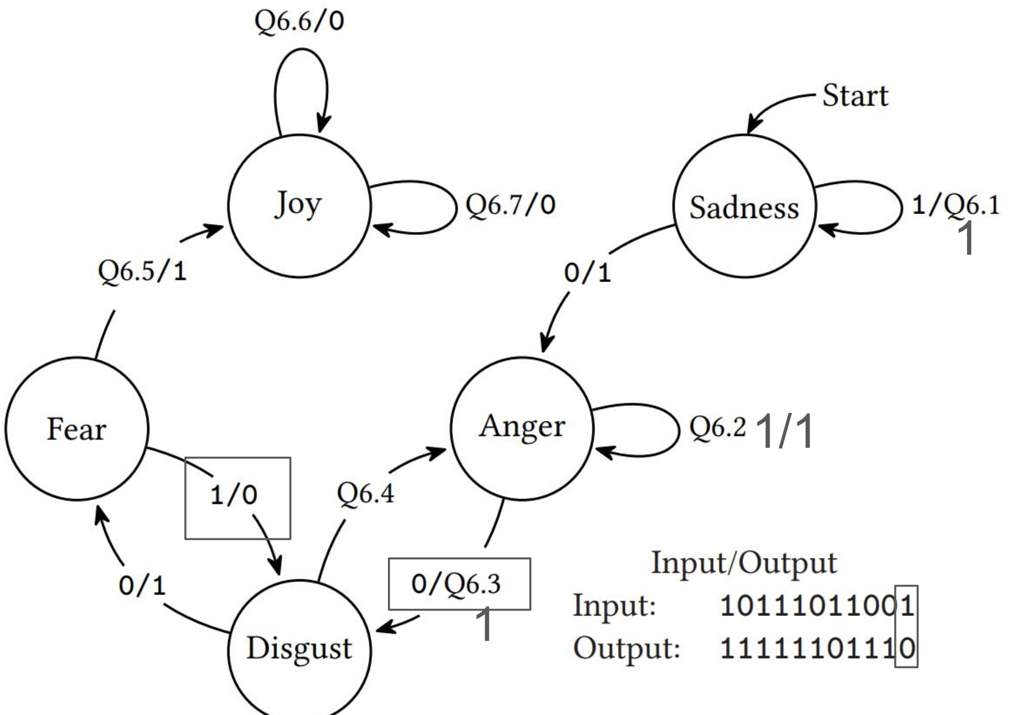
Input/Output

Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.



Input/Output

Input: 10111011001

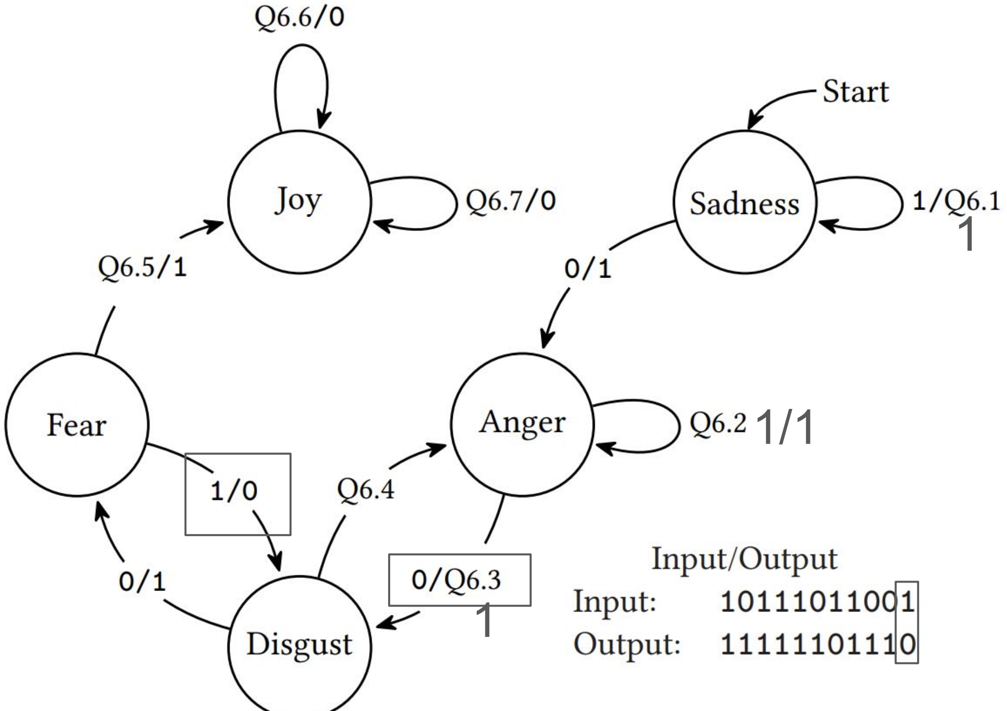
Output: 11111101110

Q6.5: hmmmm What can it be?

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.5: 0



Input/Output

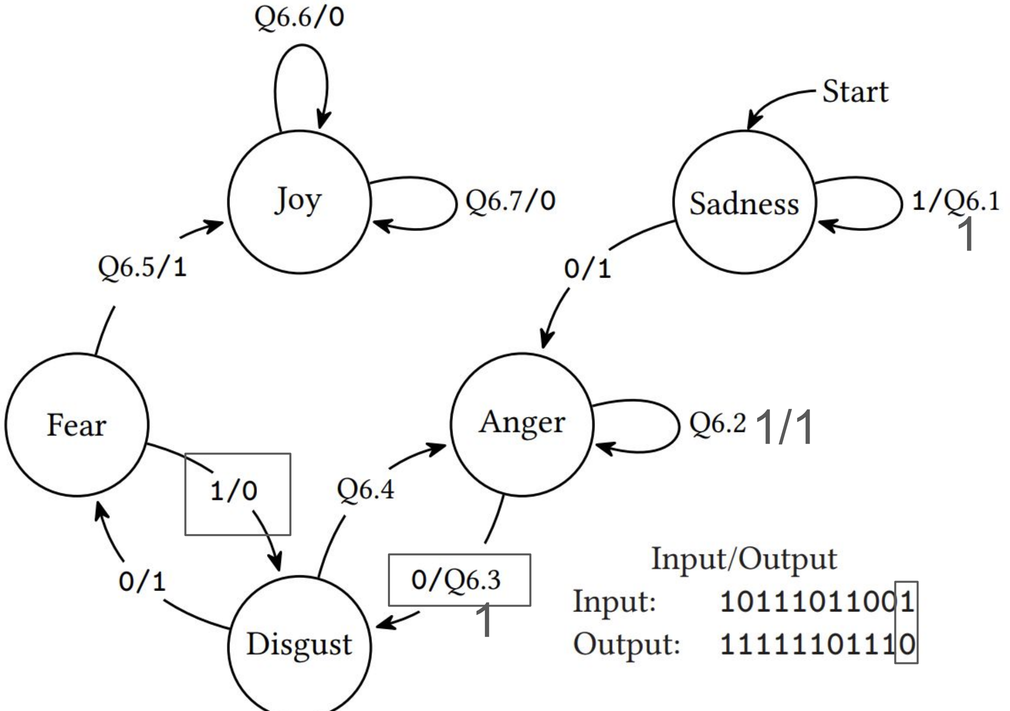
Input: 10111011001

Output: 11111101110

FSM (Su24) Tip: be patient :)))

Uh oh! Riley's emotions are scrambled, and she's forgotten how to go from one emotion (state) to the next. Select the state transitions such that the FSM matches the input/output provided.

Q6.6/7: 0/1



Input/Output

Input: 1011101100$\boxed{1}$

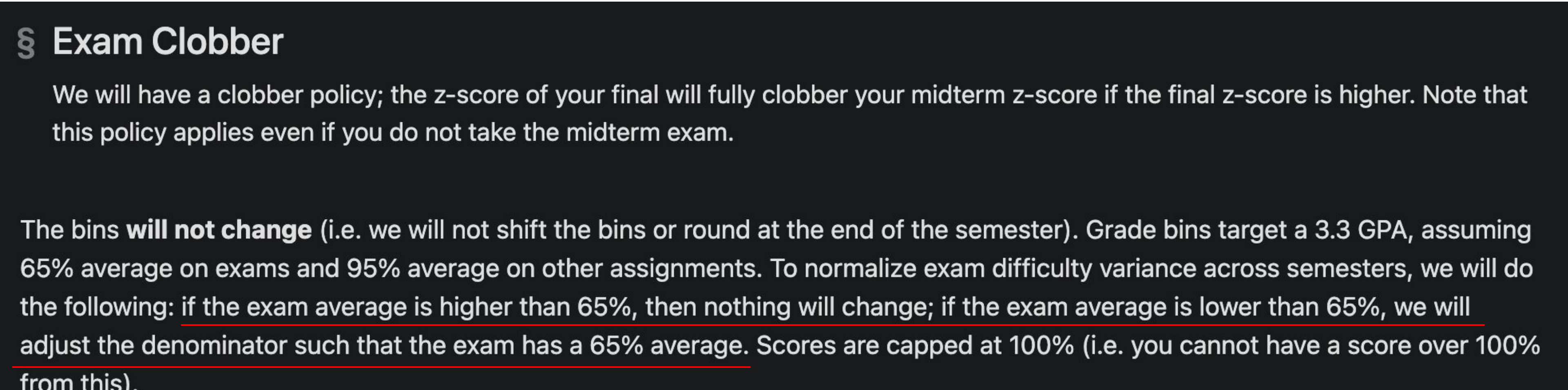
Output: 1111110111$\boxed{0}$

Joy forever!!!!
Hope you joy the midterm as well :)))))

Best of luck on your midterm!!!!!!!!!!

You all got this!

Remember, there’s a clobber policy :))))



Exam Clobber

We will have a clobber policy; the z-score of your final will fully clobber your midterm z-score if the final z-score is higher. Note that this policy applies even if you do not take the midterm exam.

The bins **will not change** (i.e. we will not shift the bins or round at the end of the semester). Grade bins target a 3.3 GPA, assuming 65% average on exams and 95% average on other assignments. To normalize exam difficulty variance across semesters, we will do the following: if the exam average is higher than 65%, then nothing will change; if the exam average is lower than 65%, we will adjust the denominator such that the exam has a 65% average. Scores are capped at 100% (i.e. you cannot have a score over 100% from this).