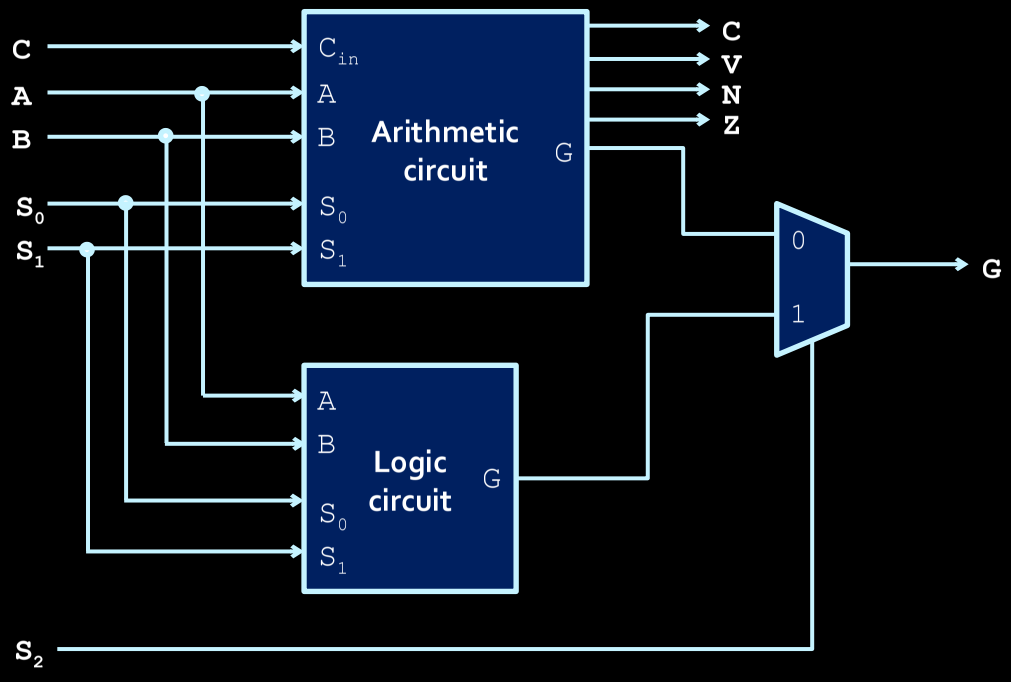
Week 6 Review

ALU Diagram



Question 1

- What should we set S_2, S_1, S_0 and C_{in} to subtract B from A (G = A – B)?
  - OK, really, you need the table….

Question 1

- What should we set S_2, S_1, S_0 and C_{in} to subtract B from A (G = A – B)?

Select		Input	Operation	
S_1	S_0	Y	C_{in}=0	C_{in}=1
0	0	All 0s	G = A (transfer)	G = A+1 (increment)
0	1	B	G = A+B (add)	G = A+B+1
1	0	$\overline{B}$	G = A+$\overline{B}$	G = A+$\overline{B}$+1 (subtract)
1	1	All 1s	G = A-1 (decrement)	G = A (transfer)

→ **S_2 = 0 S_1 = 1 S_0 = 0 C_{in} = 1**

→ we could call this : **func = 0101**

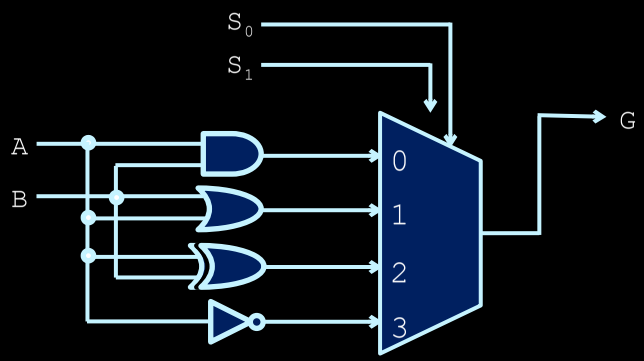
Arithmetic Side

Select		Input	Operation	
S_1	S_0	Y	C_{in}=0	C_{in}=1
0	0	All 0s	G = A (transfer)	G = A+1 (increment)
0	1	B	G = A+B (add)	G = A+B+1
1	0	$\overline{B}$	G = A+$\overline{B}$	G = A+$\overline{B}$+1 (subtract)
1	1	All 1s	G = A-1 (decrement)	G = A (transfer)

- Based on the values on the select bits and the carry bit, we can perform any number of basic arithmetic operations by manipulating what value is added to A.

Logical Side

- We also want a circuit that can perform logical operations, in addition to arithmetic ones.

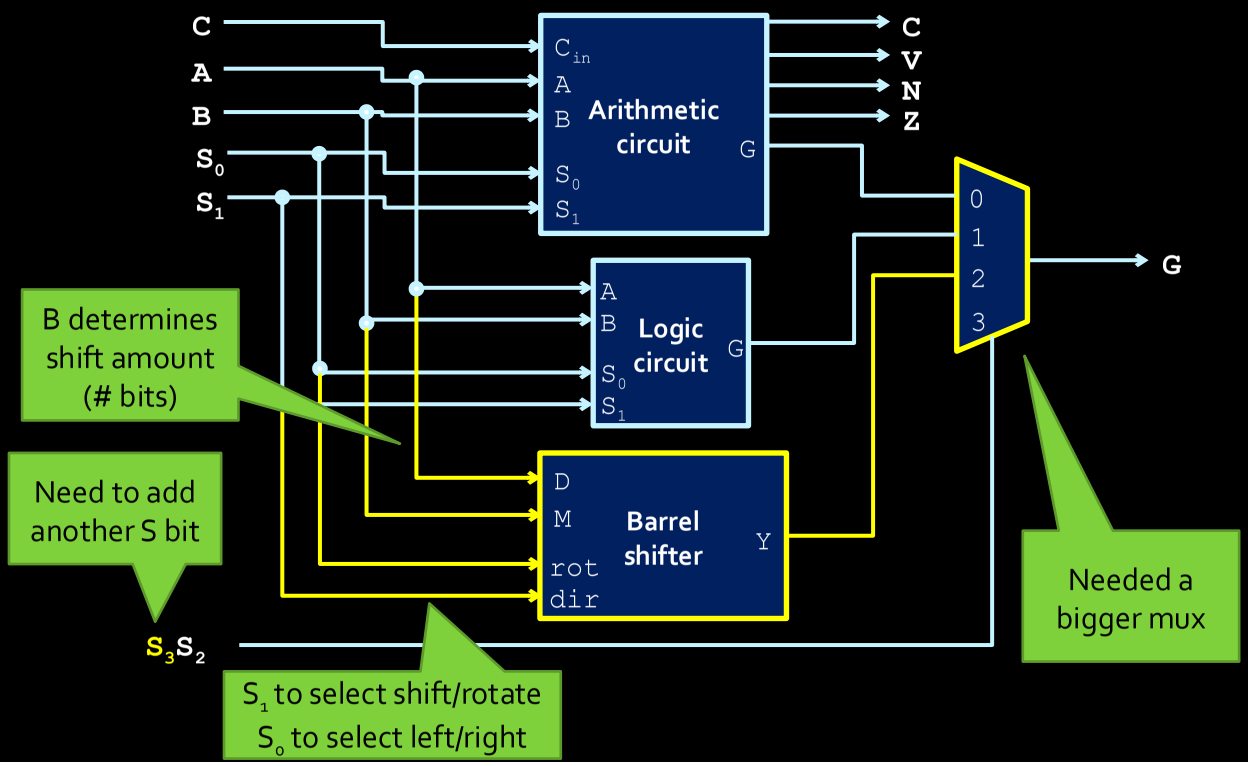


Select bits		Result
S_1	S_0	
0	0	G = A AND B
0	1	G = A OR B
1	0	G = A XOR B
1	1	G = $\overline{A}$

Question #2

- Start with a 16-bit ALU using our simple ALU design.
- Want to add shift and rotate operations to the ALU
  - Using the barrel shifter on the right
- New ALU will support:
  - Arithmetic ops (as before!)
  - Logical ops (as before)
  - Shifting (new!):
    - Shift right/left
    - Rotate right/left
- Task: sketch a new design for the ALU
  - Think which components to add and how to connect.
  - Think which inputs and outputs we'd have to add.

Question #2



Question

- Design a datapath that computes $Res = X^Y$
  - X and Y, Res are unsigned 8-bit numbers.
  - Can assume no overflow.
  - Res output must come from a register.
- We are allowed the following:
  - one 8-bit multiplier
  - one 8-bit subtractor
  - one 8-bit comparator
  - two 8-bit registers
  - Up to five 8-bit muxes (4-to-1 or 2-to-1)
  - As many wires and constants as we want.

Question

- $Res = X^Y$
- Guidance:
  1. Relax
  2. Think of algorithm (high-level pseudocode) to do this using only
     - Operators: assignment, multiply, subtract, compare
     - Two variables A and B?
     - Constants
  3. Build datapath to support your algorithm
  4. Build FSM to implement the algorithm and run the datapath

Question : Res = X^Y

- Algorithm:

Question : Res = X^Y

- Algorithm:
  1. A ← 1
  2. B ← Y
  3. While B > 0:
     - A ← A * X
     - B ← B – 1
  4. Res is in A

- (how will we do "while"? Using the FSM!)

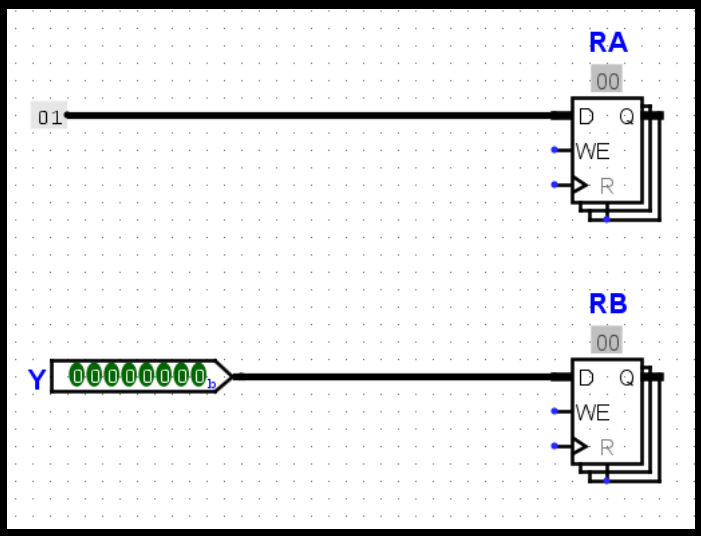
Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A

- Build Datapath to support it
  - Start with inputs, registers, and operations
  - Connnect with muxes and wires
  - Remember control signals: register load, select

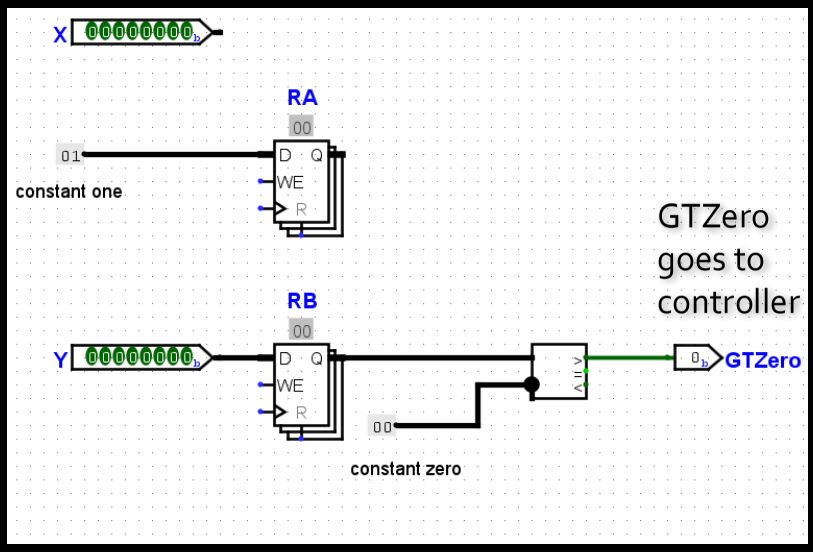
Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A



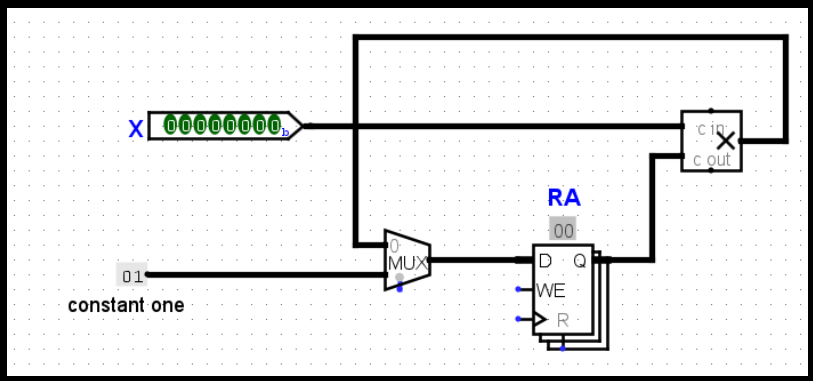
Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A



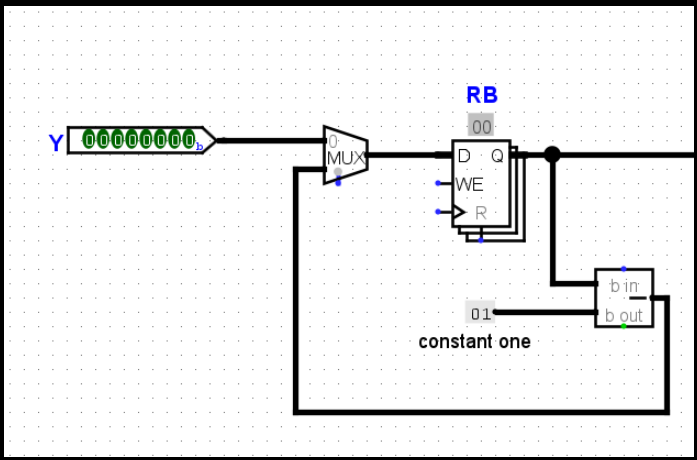
Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A

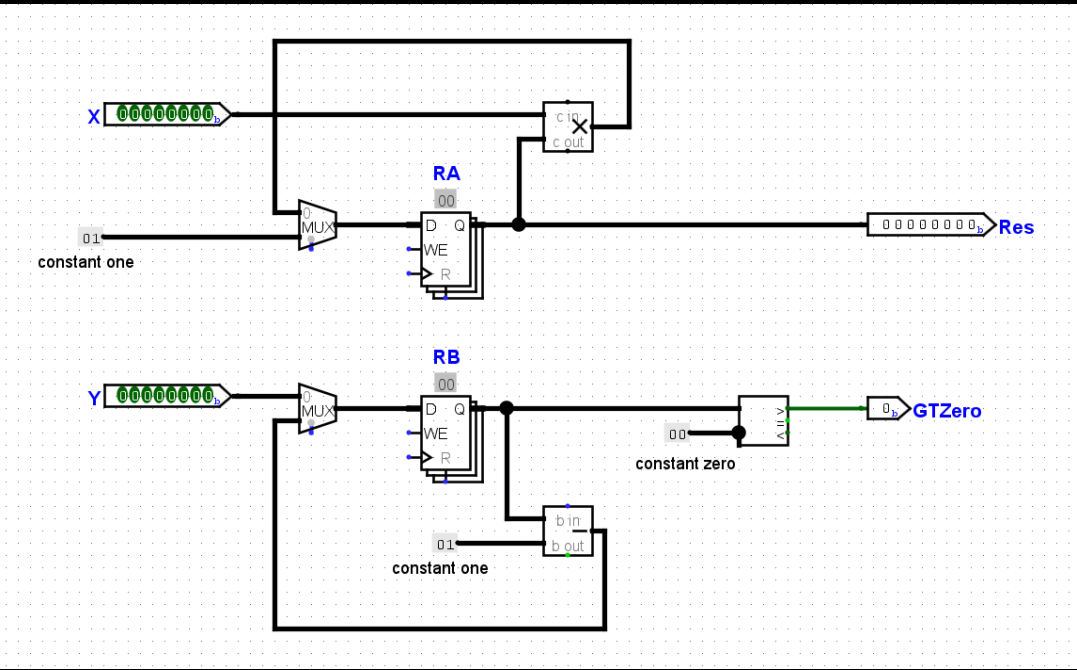


Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A

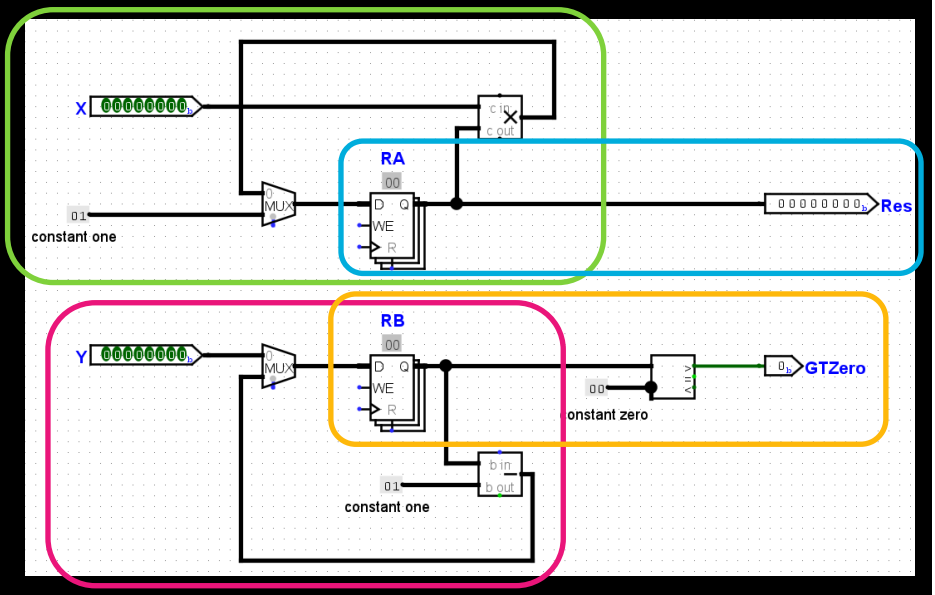


Question : Res = X^Y

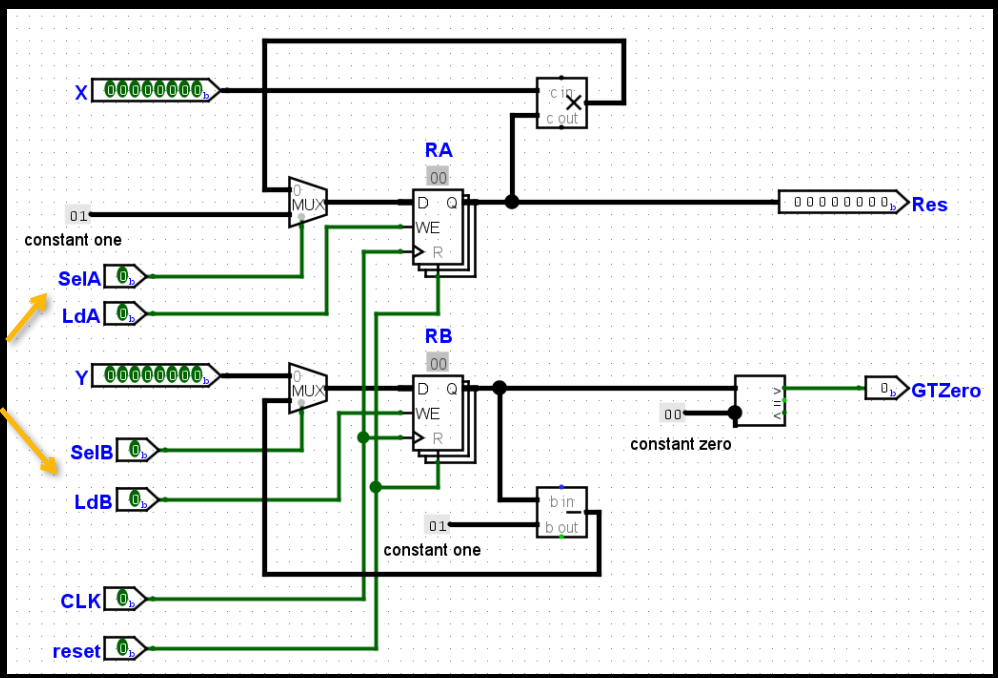


Question : Res = X^Y

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A



Question : Res = X^Y



control signals

clock and reset

Question : Res = X^Y

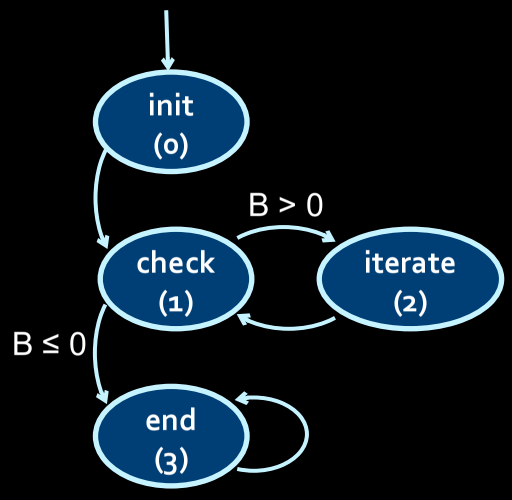
- Now design the FSM
- Guidance: Start with state diagram

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A

Question : Res = X^Y

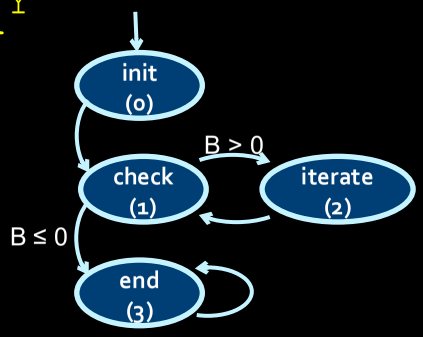
- Now design the FSM
- Guidance: Start with state diagram

1. A ← 1
2. B ← Y
3. While B > 0:
   - A ← A * X
   - B ← B – 1
4. Res is in A



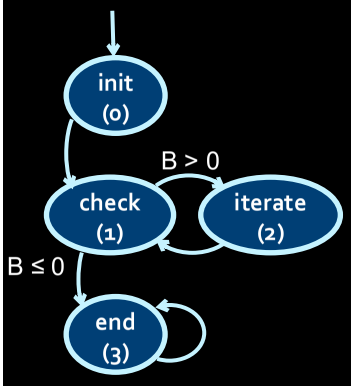
Question : Res = X^Y

- Now design the FSM
- Guidance: fill up this table

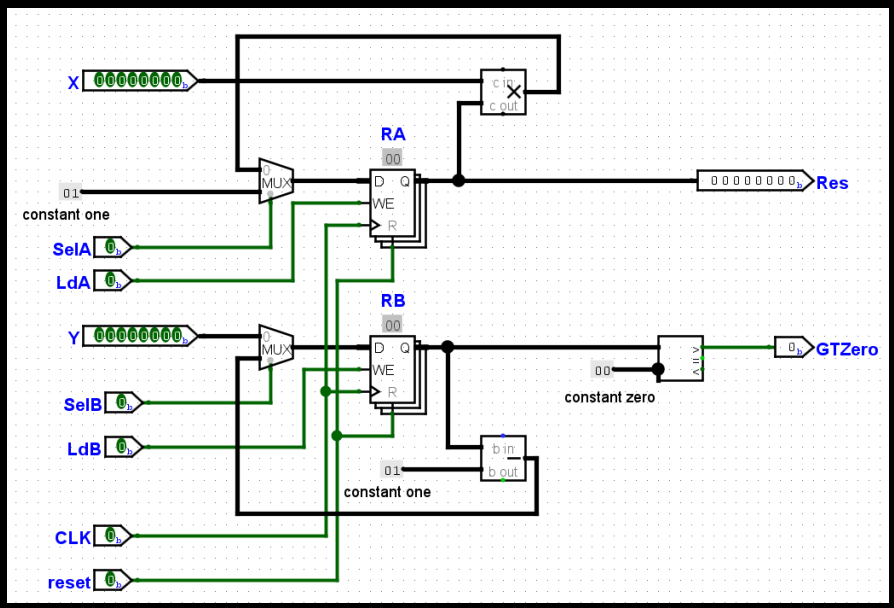


Step	Operation	GTZero	SelA	SelB	LdA	LdB	Done?	NextStep

Question : Res = X^Y

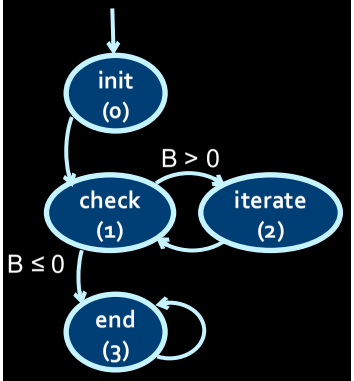


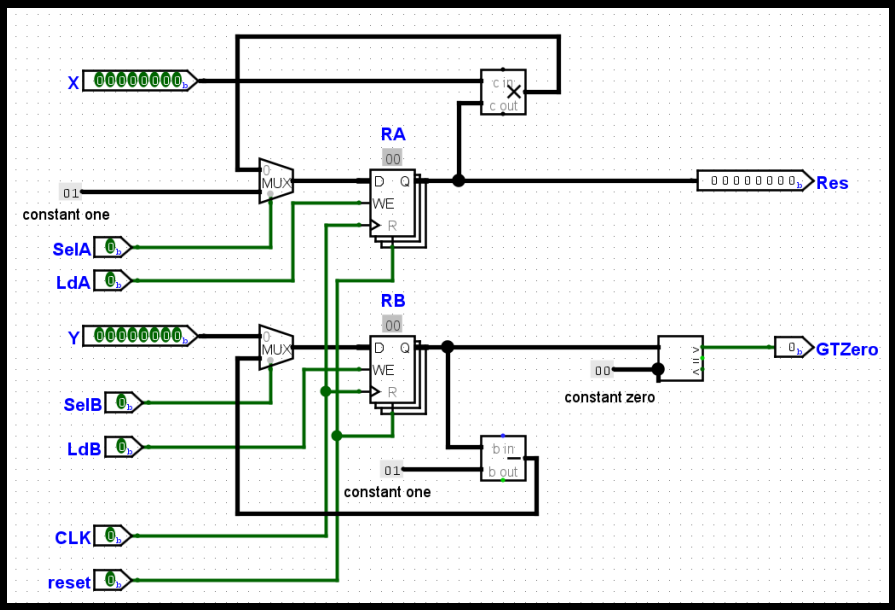
Fill the table



Step	Operation	GTZero	SelA	SelB	LdA	LdB	Done?	NextStep
0								
1								

Question : Res = X^Y

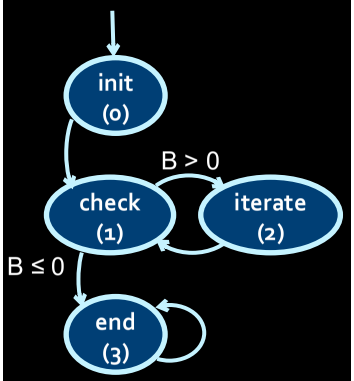


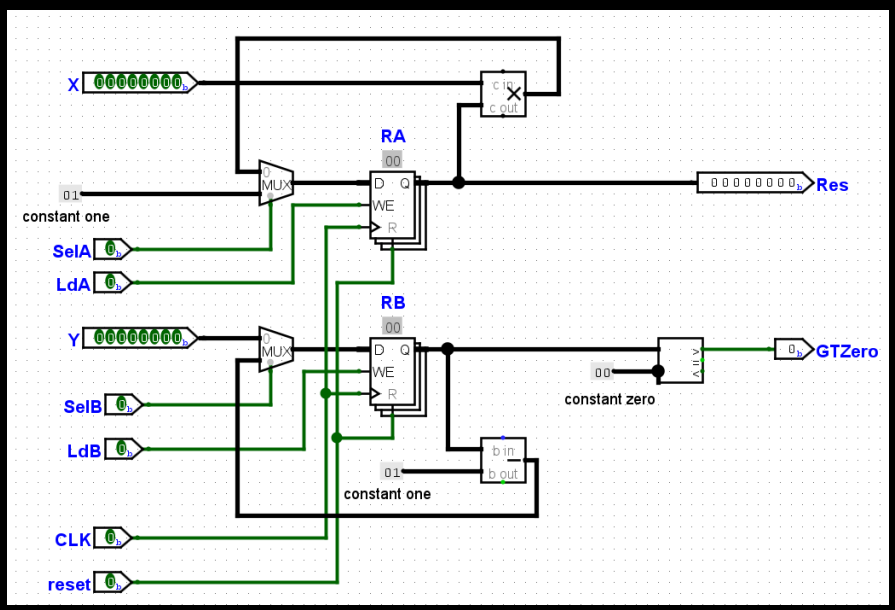


Fill the table

Step	Operation	GTZero	SelA	SelB	LdA	LdB	Done?	NextStep
0	A ← 1 B ← Y	X	1	0	1	1	0	1
1								

Question : Res = X^Y





Fill the table

Step	Operation	GTZero	SelA	SelB	LdA	LdB	Done?	NextStep
0	A ← 1 B ← Y	X	1	0	1	1	0	1
1	If B > 0	1	x	x	0	0	0	2

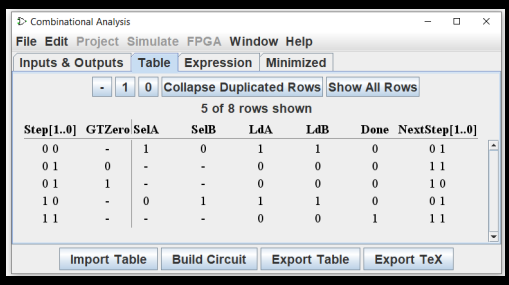
Question : Res = X^Y

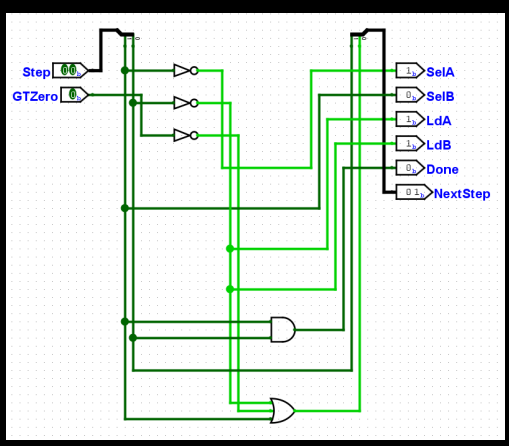
Step	Operation	GTZero	SelA	SelB	LdA	LdB	Done?	NextStep
0	A ← 1 B ← Y	x	1	0	1	1	0	1
1	If B > 0	1	x	x	0	0	0	2
1	If B <= 0	0	x	x	0	0	0	3
2	A ← A * X B ← B - 1	x	0	1	1	1	0	1
3	done	x	x	x	0	0	1	3 (stay here)

- If you ignore "operation" this is a truth table for FSM!
  - 4 steps → 4 states → need 2 flipflops
- Output never depends on input (only next state of step 1)
  - Moore machine
- Could remove step 1 by
  - Doing a check in step 0
  - Doing a check in step 2
  - Outputs would depend on GTZero → Mealy machine

Question : Res = X^Y

- Implement the FSM
- Just write the truth table in Circuit Analyzer
- Let it build the circuit for you





Question : Res = X^Y