CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 25, FALL 2012

TOPICS TODAY

- **Finite State Machine Simplification**
- **A 2-bit "CPU"**

FINITE STATE MACHINE SIMPLIFICATION STEPS

- Minimize combinational logic circuit (hard)
- Reduce number of states
- Apply state assignment heuristics
- Consider choice of flip flops (e.g., J-K vs D)

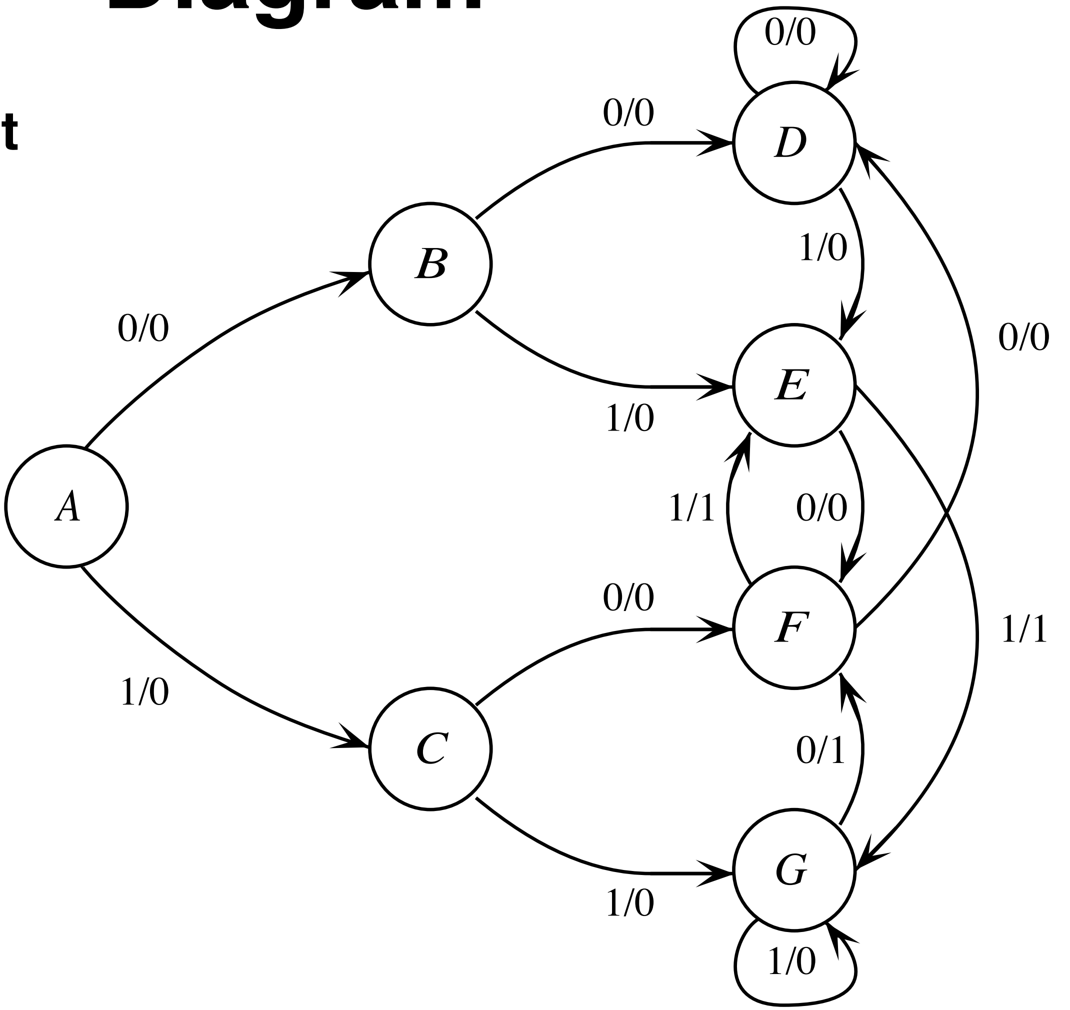
EXAMPLE:
SEQUENCE DETECTOR

Example: A Sequence Detector

- Example: Design a machine that outputs a 1 when exactly two of the last three inputs are 1.
- *e.g.* input sequence of 011011100 produces an output sequence of 001111010.
- Assume input is a 1-bit serial line.
- Use D flip-flops and 8-to-1 Multiplexers.
- Start by constructing a state transition diagram (next slide).

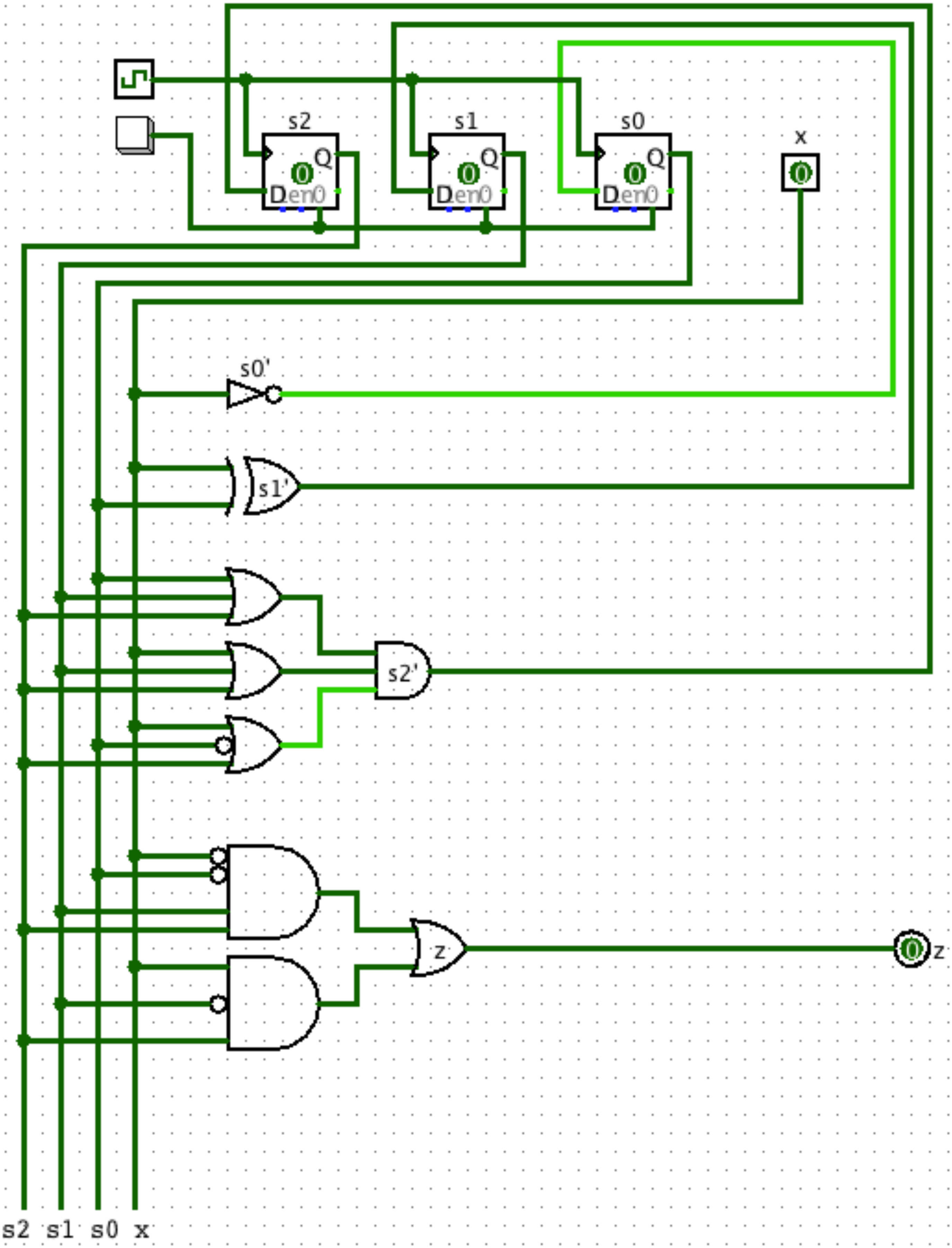
Sequence Detector State Transition Diagram

- **Design a machine that outputs a 1 when exactly two of the last three inputs are 1.**



Sequence Dectector

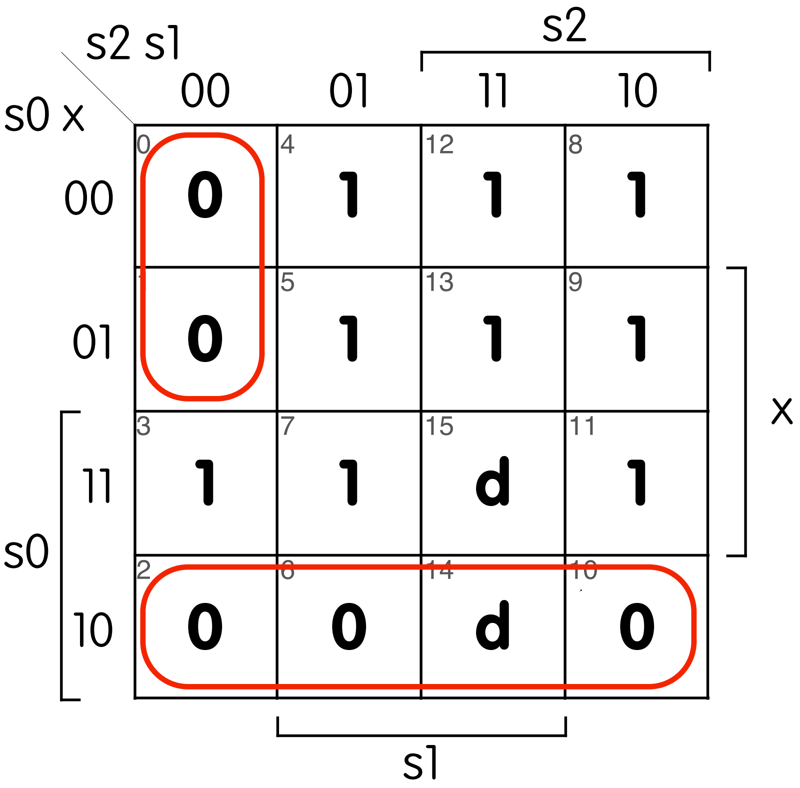
Output 1 when EXACTLY two of last three bits are 1



COMBINATIONAL LOGIC CIRCUIT MINIMIZATION

Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

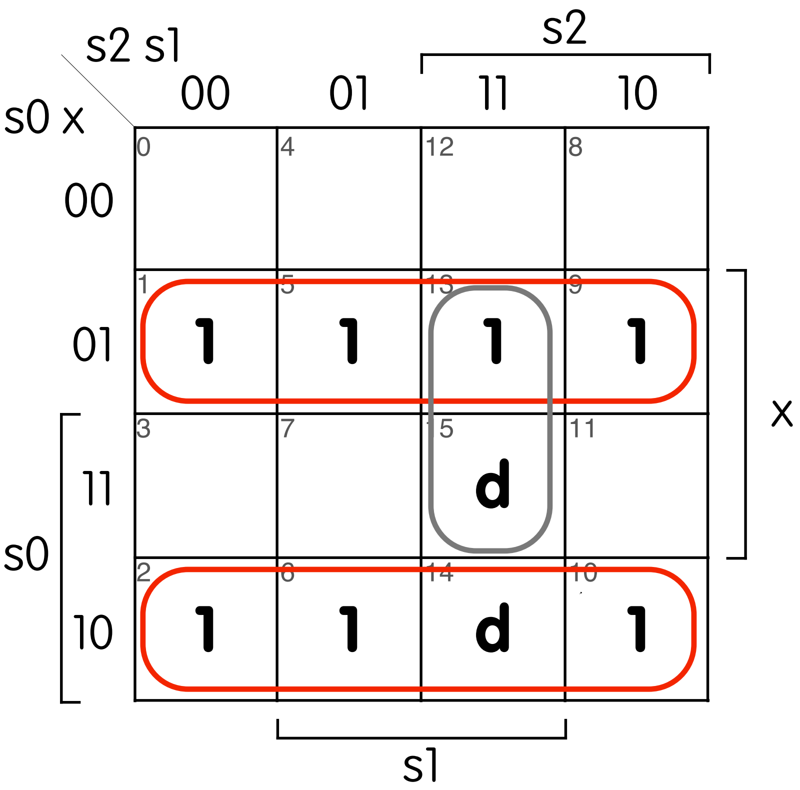


s0 x \ s2 s1	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	d	1
10	0	0	d	0

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

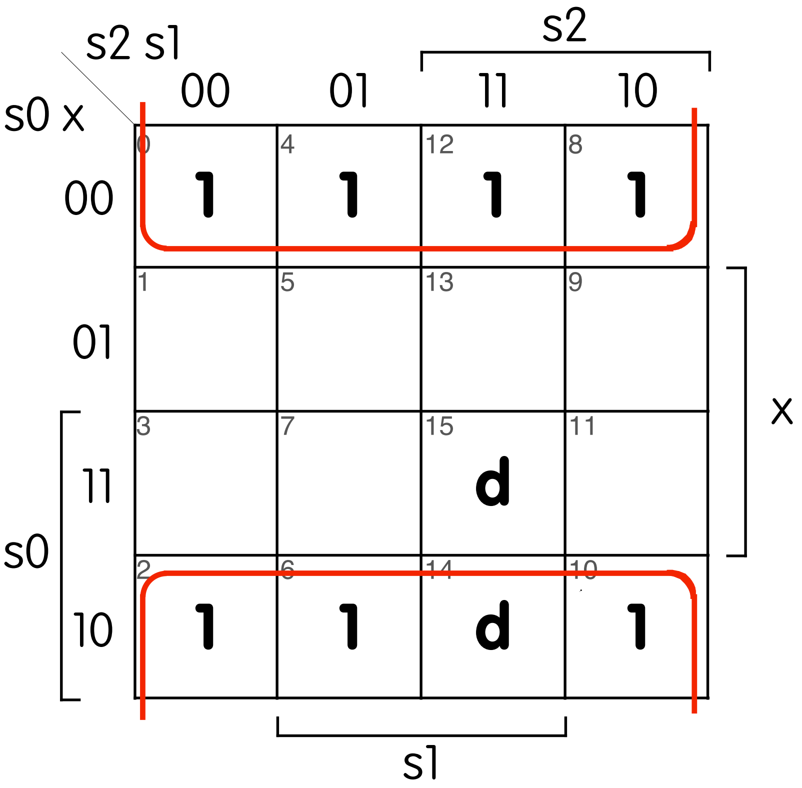


s0 x \ s2 s1	00	01	11	10
00	0	4	12	8
01	1 1	5 1	13 1	9 1
11	3	7	15 d	11
10	2 1	6 1	14 d	10 1

$$s1' = \overline{s0}\ x + s0\ \overline{x} = s0\ \mathrm{xor}\ x$$

Sequence Detector

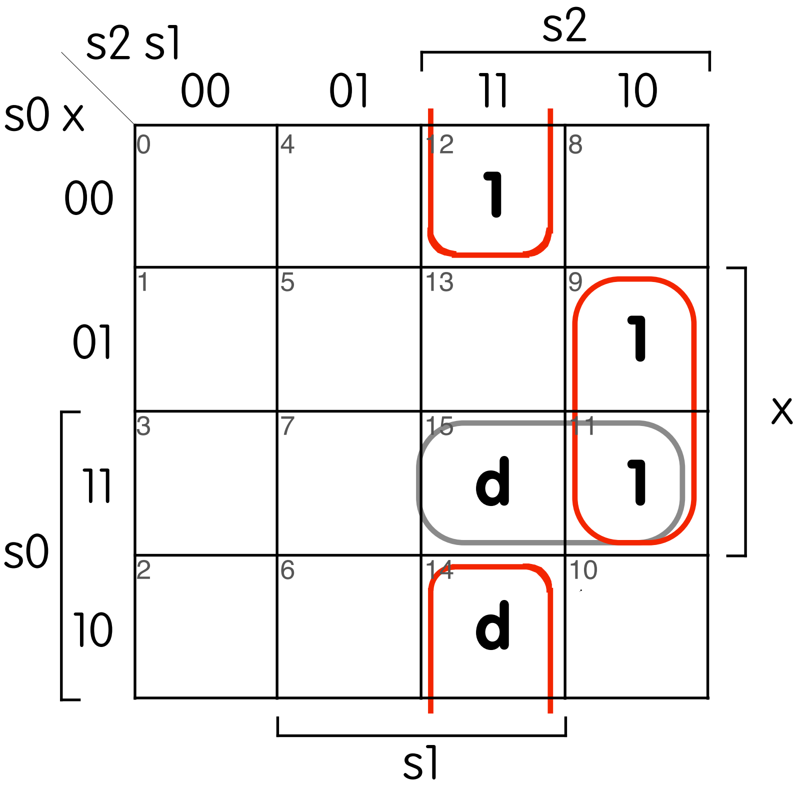
	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d



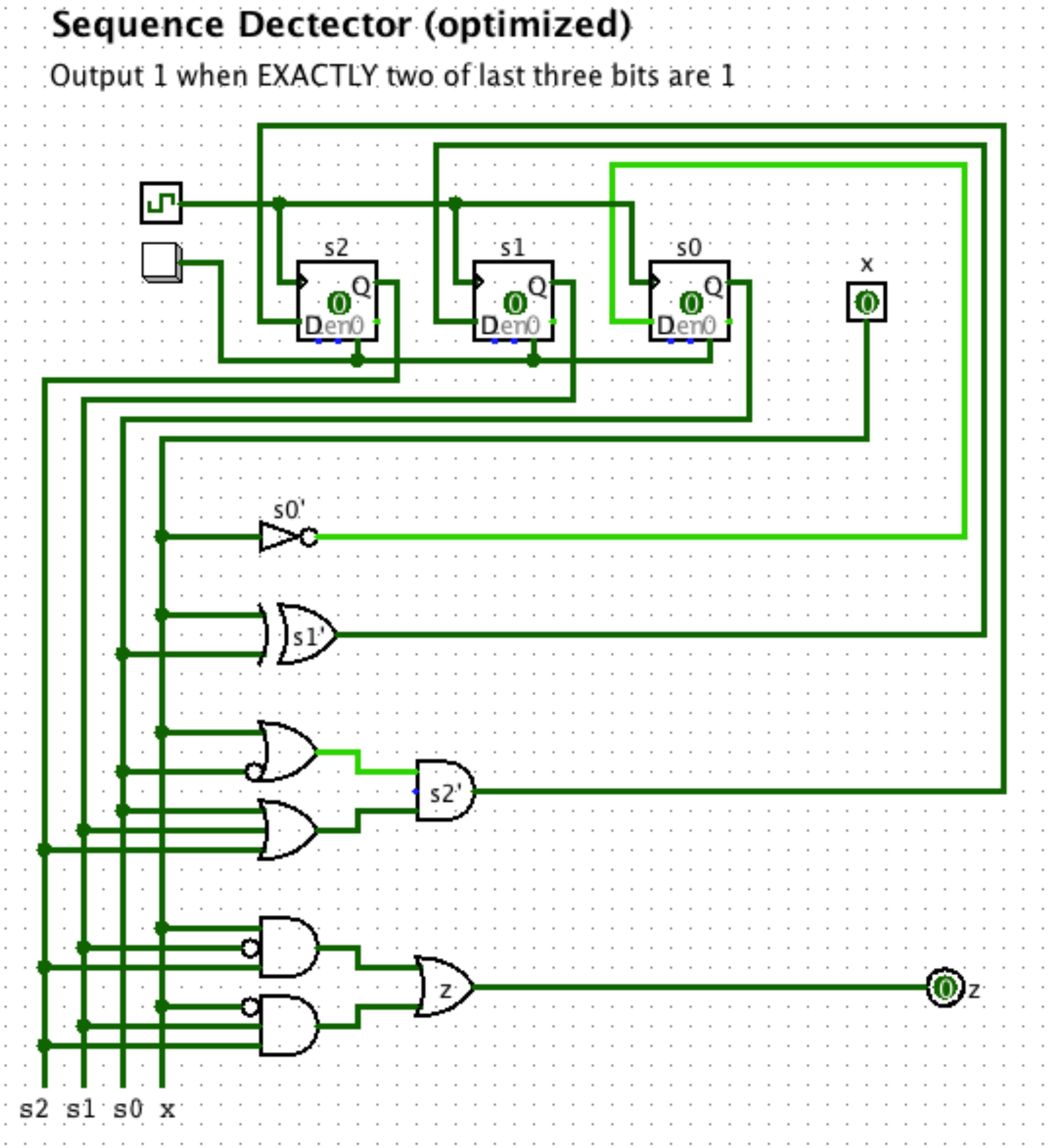
$$s0' = \overline{x}$$

Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d



$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$



Sequence Dectector (optimized)

Output 1 when EXACTLY two of last three bits are 1

Circuit Minimization is Hard

- Unix systems store passwords in encrypted form.
  - User types in x, system computes f(x) and looks for f(x) in a file.
- Suppose we us 64-bit passwords and I want to find the password x, such that f(x) = y. Let

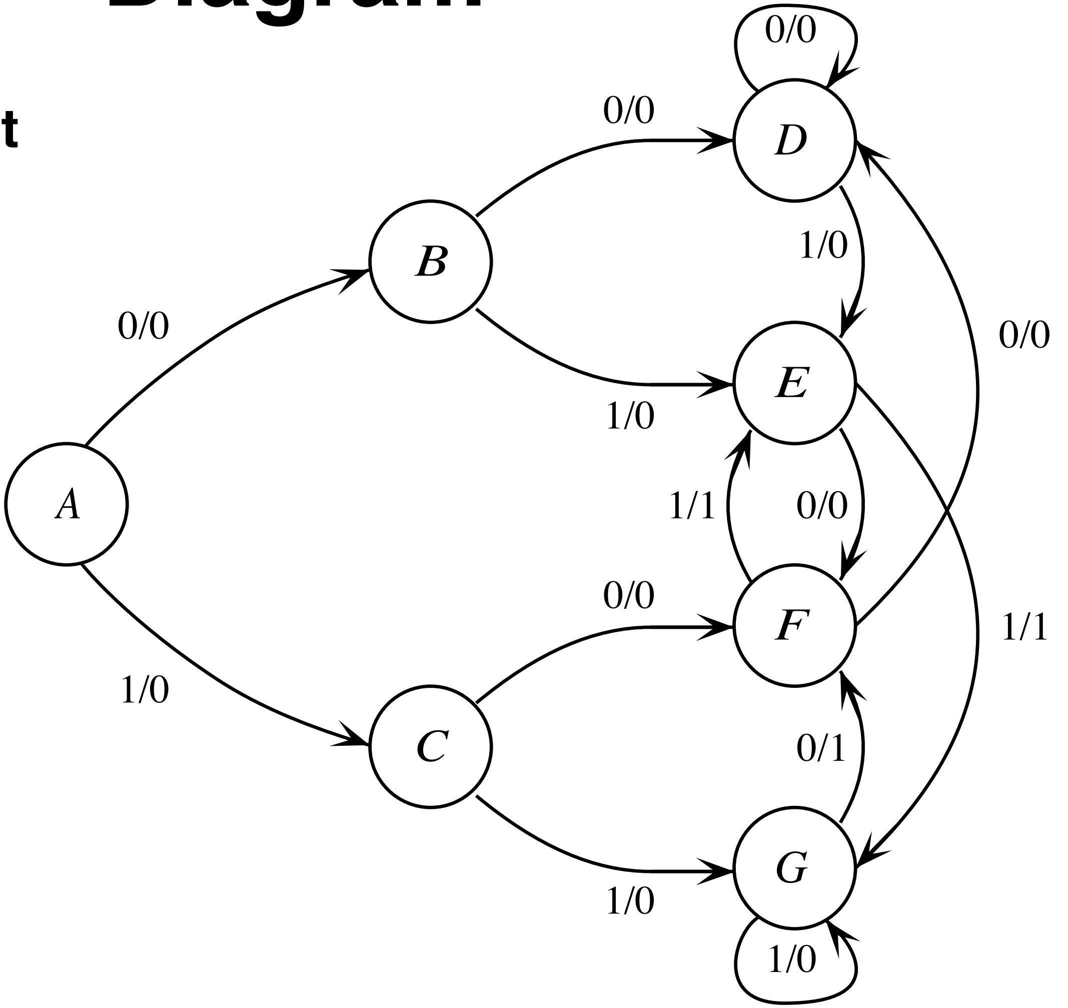
  $g_i(x)$ = 0 if f(x) = y and the ith bit of x is 0

  1 otherwise.
- If the ith bit of x is 1, then $g_i(x)$ outputs 1 for every x and has a very, very simple circuit.
- If you can simplify every circuit quickly, then you can crack passwords quickly.

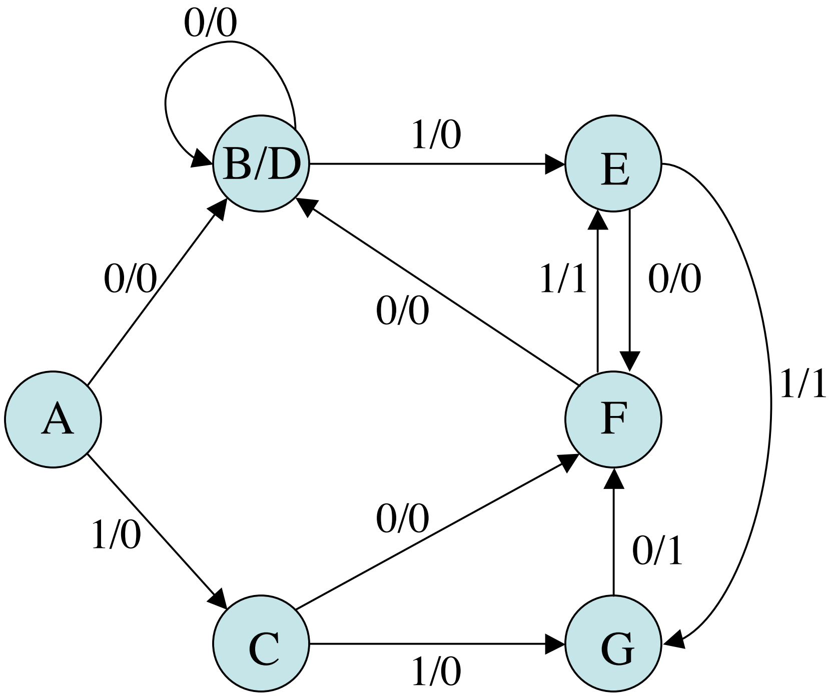
STATE REDUCTION

Sequence Detector State Transition Diagram

- Design a machine that outputs a 1 when exactly two of the last three inputs are 1.



6-State Sequence Detector



State Reduction Algorithm

1. Use a 2-dimensional table — an entry for each pair of states.
2. Two states are "distinguished" if:

   a. States X and Y of a finite state machine M are distinguished if there exists an input r such that the output of M in state X reading input r is different from the output of M in state Y reading input r.

   b. States X and Y of a finite state machine are distinguished if there exists an input r such that M in state X reading input r goes to state X', M in state Y reading input r goes to state Y' and we already know that X' and Y' are distinguished states.

3. For each pair (X,Y), check if X and Y are distinguished using the definition above.
4. At the end of the algorithm, states that are not found to be distinguished are in fact equivalent.

Sequence Detector State Reduction Table

	A	B	C	D	E	F	G
A	╳	x	x	x	x	x	x
B	╳	╳	x		x	x	x
C	╳	╳	╳	x	x	x	x
D	╳	╳	╳	╳	x	x	x
E	╳	╳	╳	╳	╳	x	x
F	╳	╳	╳	╳	╳	╳	x
G	╳	╳	╳	╳	╳	╳	╳

State Reduction Algorithm Performance

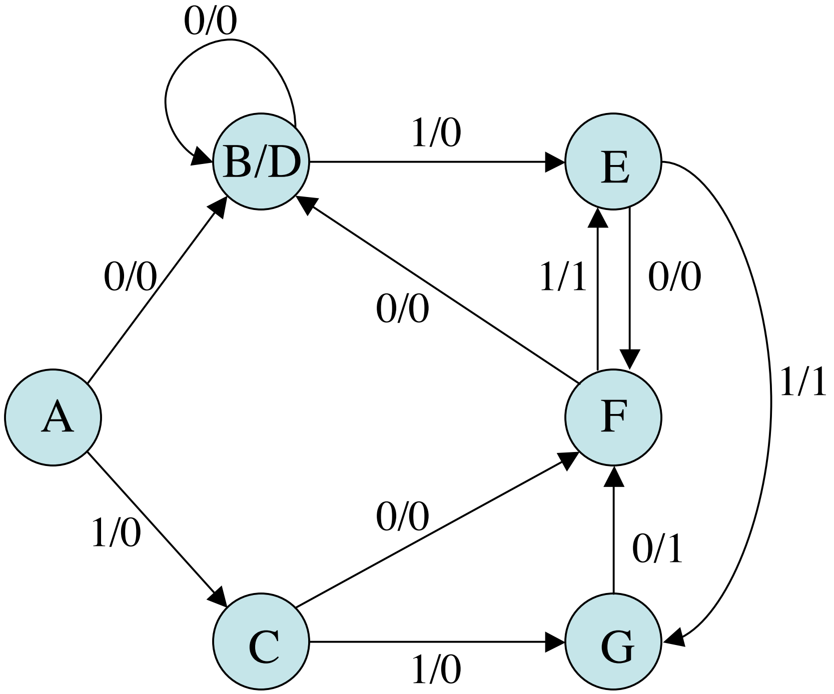
- As stated, the algorithm takes $O(n^4)$ time for a FSM with n states, because each pass takes $O(n^2)$ time and we make at most $O(n^2)$ passes.
- A more clever implementation takes $O(n^2)$ time.
- The algorithm produces a FSM with the fewest number states possible.
- Performance and correctness can be proven.

STATE ASSIGNMENT

State Assignment Heuristics

- No known efficient alg. for best state assignment
- Some heuristics (rules of thumb):
  - ◇ The initial state should be simple to reset — all zeroes or all ones.
  - ◇ Minimize the number of state variables that change on each transition.
  - ◇ Maximize the number of state variables that don't change on each transition.
  - ◇ Exploit symmetries in the state diagram.
  - ◇ If there are unused states (when the number of states s is not a power of 2), choose the unused state variable combinations carefully. (Don't just use the first s combination of state variables.)
  - ◇ Decompose the set of state variables into bits or fields that have well-defined meaning with respect to the input or output behavior.
  - ◇ Consider using more than the minimum number of states to achieve the objectives above.

6-State Sequence Detector



Sequence Detector State Assignment

Input Present state	X	
	0	1
$S_2S_1S_0$	$S_2S_1S_0Z$	$S_2S_1S_0Z$
A': 000	001/0	010/0
B': 001	001/0	011/0
C': 010	100/0	101/0
D': 011	100/0	101/1
E': 100	001/0	011/1
F': 101	100/1	101/0

Improved Sequence Detector?

- Formulas from the 7-state FSM:

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

$$s1' = \overline{s0}\ x + s0\ \overline{x} = s0 \text{ xor } x$$

$$s0' = \overline{x}$$

$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

- Formulas from the 6-state FSM:

$$s2' = s2\ s0 + s1$$

$$s1' = \overline{s2}\ \overline{s1}\ x + s2\ \overline{s0}\ x$$

$$s0' = \overline{s2}\ \overline{s1}\ \overline{x} + s0\ x + s2\ \overline{s0} + s1\ x$$

$$z = s2\ \overline{s0}\ x + s1\ s0\ x + s2\ s0\ \overline{x}$$

Sequence Detector State Assignment

7-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	0	1	1	0
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B = 001
C = 010
D = 011

E = 100
F = 101
G = 110

new 6-state

	s2	s1	s0	x	s2'	s1'	s0'	z
0	0	0	0	0	0	0	1	0
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	0	1	0
3	0	0	1	1	1	0	0	0
4	0	1	0	0	1	0	1	0
5	0	1	0	1	1	1	0	0
6	0	1	1	0	d	d	d	d
7	0	1	1	1	d	d	d	d
8	1	0	0	0	1	0	1	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	0	0	1	0
11	1	0	1	1	1	0	0	1
12	1	1	0	0	1	0	1	1
13	1	1	0	1	1	1	0	0
14	1	1	1	0	d	d	d	d
15	1	1	1	1	d	d	d	d

A = 000
B/D = 001
C = 010
~~D = 011~~

E = 100
F = 101
G = 110

Improved Sequence Detector

- Textbook formulas for the 6-state FSM:

$$s2' = s2\ s0 + s1$$

$$s1' = \overline{s2}\ \overline{s1}\ x + s2\ \overline{s0}\ x$$

$$s0' = \overline{s2}\ \overline{s1}\ \overline{x} + s0\ x + s2\ \overline{s0} + s1\ x$$

$$z = s2\ \overline{s0}\ x + s1\ s0\ x + s2\ s0\ \overline{x}$$

- New formulas for the 6-state FSM:

$$s2' = (\overline{s0} + x)(s2 + s1 + s0)$$

$$s1' = \overline{s0}\ x$$

$$s0' = \overline{x}$$

$$z = s2\ \overline{s1}\ x + s2\ s1\ \overline{x}$$

CHOICE OF FLIP FLOP

Excitation Tables

- **Each table shows the settings that must be applied at the inputs at time *t* in order to change the outputs at time *t*+1.**

S-R flip-flop

Q_t	Q_{t+1}	S	R
0	0	0	0
0	1	1	0
1	0	0	1
1	1	0	0

D flip-flop

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

J-K flip-flop

Q_t	Q_{t+1}	J	K
0	0	0	*d*
0	1	1	*d*
1	0	*d*	1
1	1	*d*	0

T flip-flop

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

6-State Sequence Detector

	s2	s1	s0	x	s2'	s1'	s0'	z	j2	k2	j1	k1	j0	k0
0	0	0	0	0	0	0	1	0	0	*d*	0	*d*	1	*d*
1	0	0	0	1	0	1	0	0	0	*d*	1	*d*	0	*d*
2	0	0	1	0	0	0	1	0	0	*d*	0	*d*	*d*	0
3	0	0	1	1	1	0	0	0	1	*d*	0	*d*	*d*	1
4	0	1	0	0	1	0	1	0	1	*d*	*d*	1	1	*d*
5	0	1	0	1	1	1	0	0	1	*d*	*d*	0	0	*d*
6	0	1	1	0	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
7	0	1	1	1	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
8	1	0	0	0	1	0	1	0	*d*	0	0	*d*	1	*d*
9	1	0	0	1	1	1	0	1	*d*	0	1	*d*	0	*d*
10	1	0	1	0	0	0	1	0	*d*	1	0	*d*	*d*	0
11	1	0	1	1	1	0	0	1	*d*	0	0	*d*	*d*	1
12	1	1	0	0	1	0	1	1	*d*	0	*d*	1	1	*d*
13	1	1	0	1	1	1	0	0	*d*	0	*d*	0	0	*d*
14	1	1	1	0	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*
15	1	1	1	1	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*	*d*

Q	Q'	J	K
0	0	0	*d*
0	1	1	*d*
1	0	*d*	1
1	1	*d*	0

Improved Sequence Detector

- Formulas for the 6-state FSM with D Flip-flops:

$s2' = (\overline{s0} + x)(s2 + s1 + s0)$

$s1' = \overline{s0}\ x$

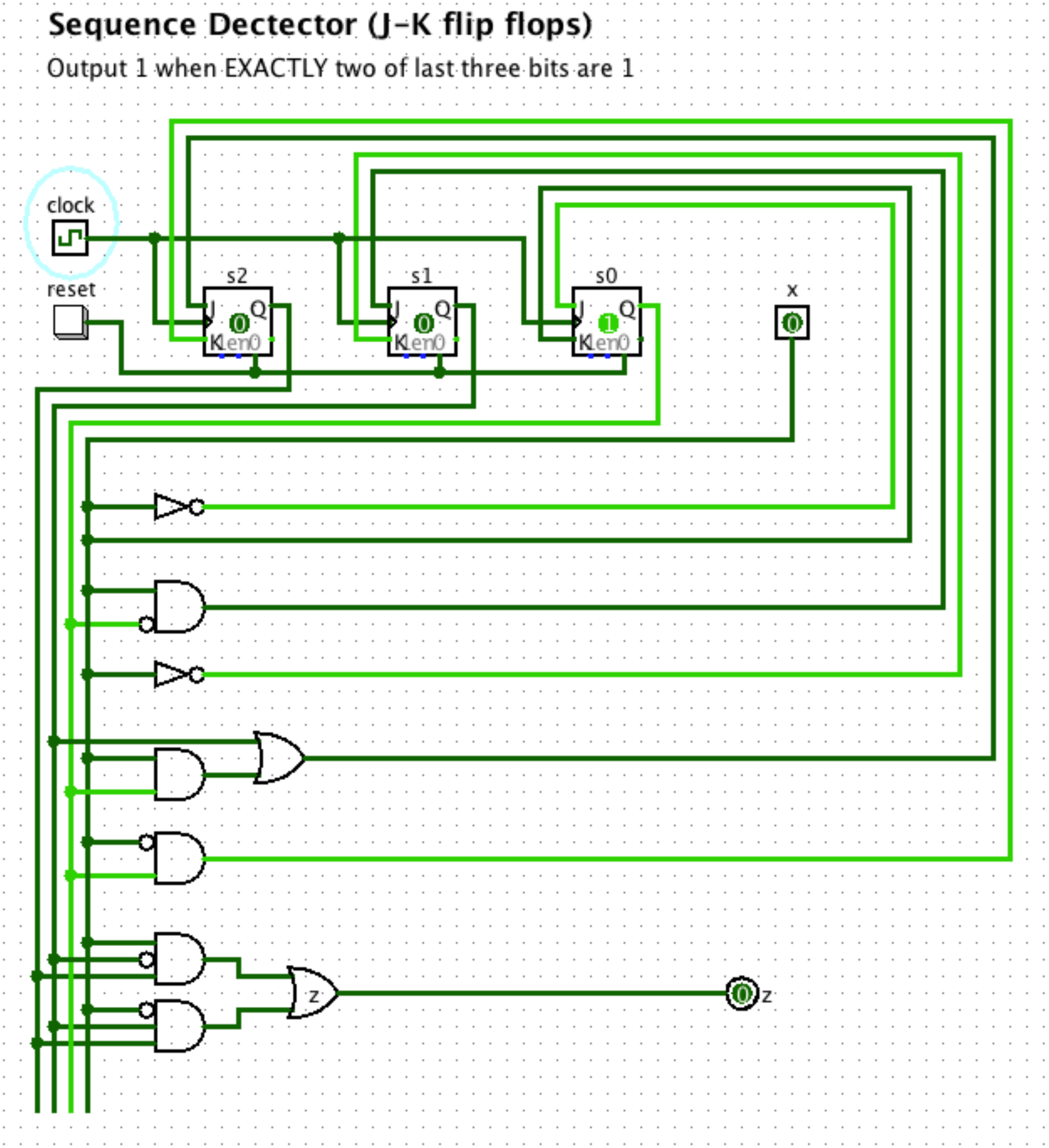
$s0' = \overline{x}$

- Formulas for the 6-state FSM with J-K Flip-flops:

$J2 = s1 + s0\ x$ $K2 = s0\ \overline{x}$

$J1 = \overline{s0}\ x$ $K1 = \overline{x}$

$J0 = \overline{x}$ $K0 = x$



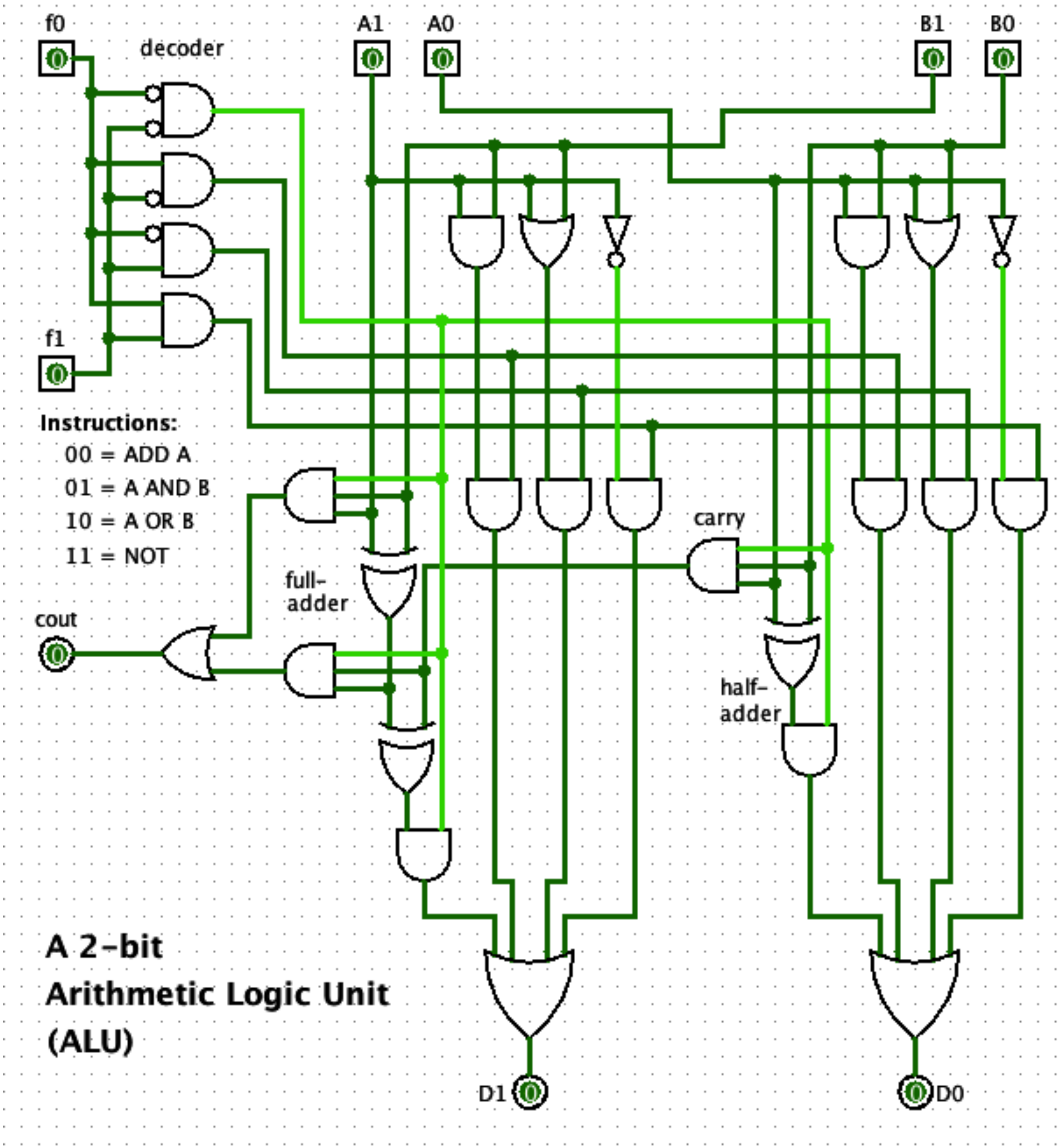
Sequence Dectector (J-K flip flops)

Output 1 when EXACTLY two of last three bits are 1

A 2-BIT "CPU"

2-BIT CPU: VERSION 1

- 2-bit ALU in sub-circuit
- Connect two 2-bit registers to 2-bit ALU
- Output of ALU stored in Register 1



Instructions:

00 = ADD A

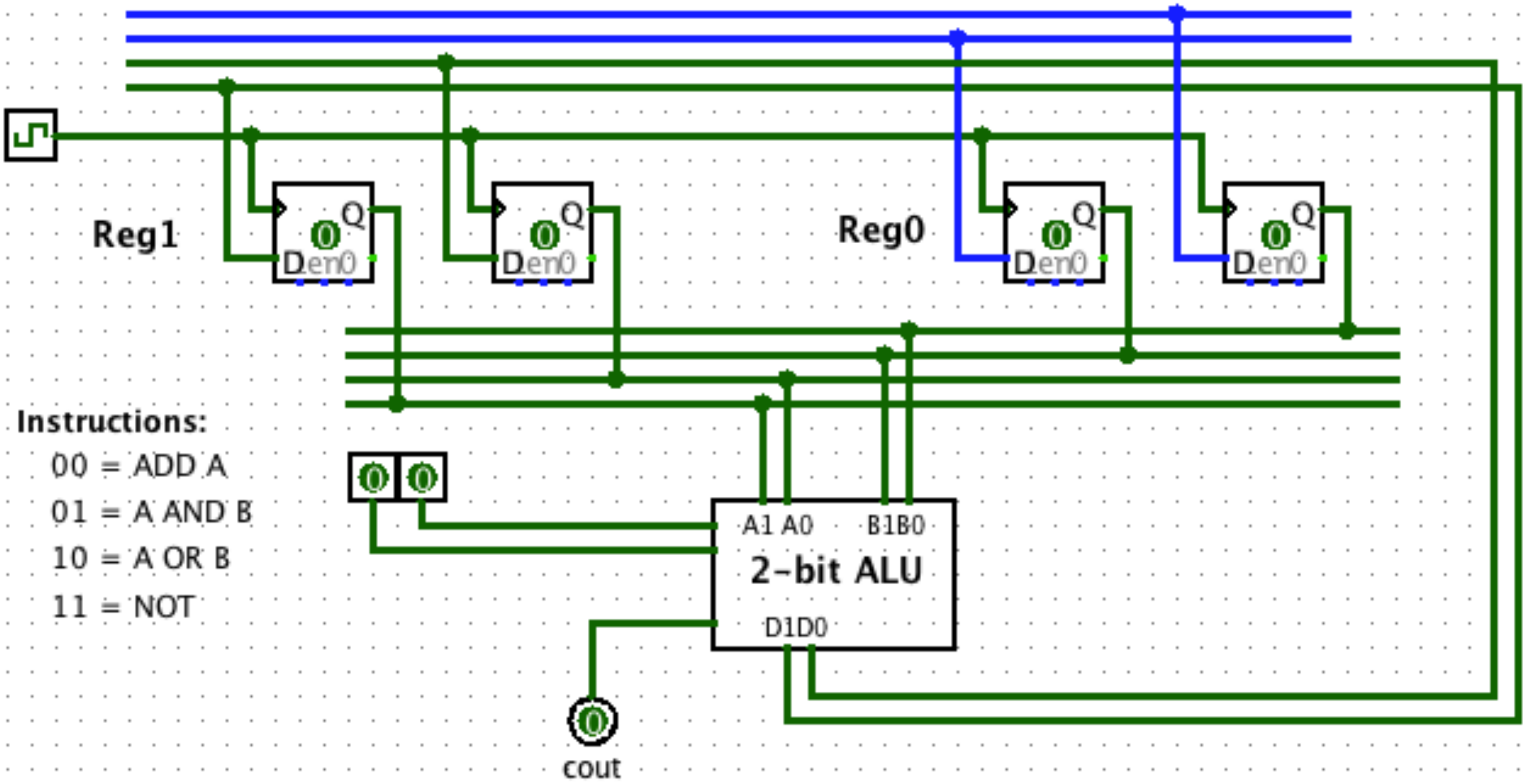
01 = A AND B

10 = A OR B

11 = NOT

A 2-bit Arithmetic Logic Unit (ALU)

2-bit CPU, version 1



Instructions:

00 = ADD A

01 = A AND B

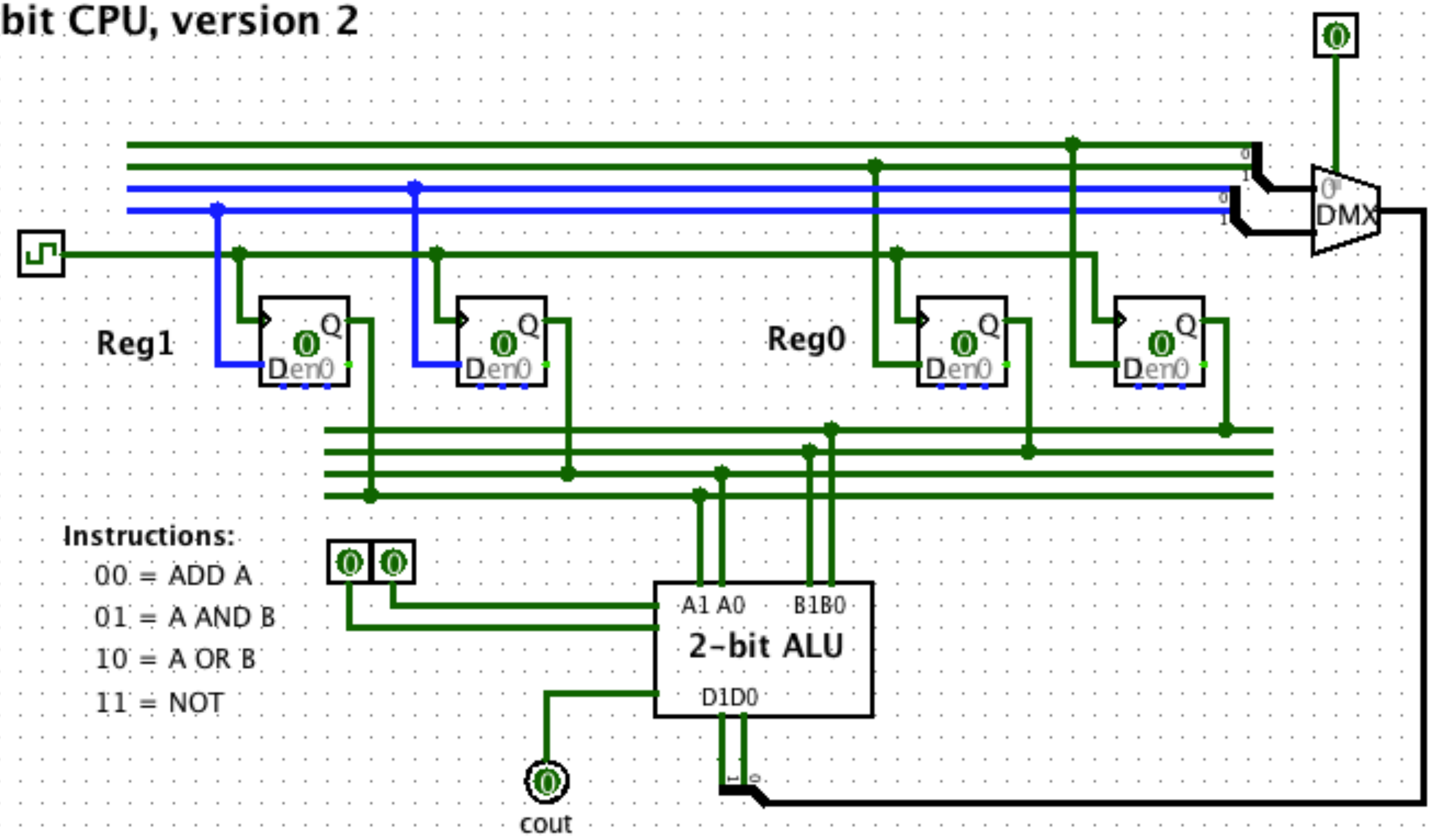
10 = A OR B

11 = NOT

2-BIT CPU: VERSION 2

- Use DEMUX to select destination register
- Use Logisim wire bundles

2-bit CPU, version 2



Instructions:

00 = ADD A

01 = A AND B

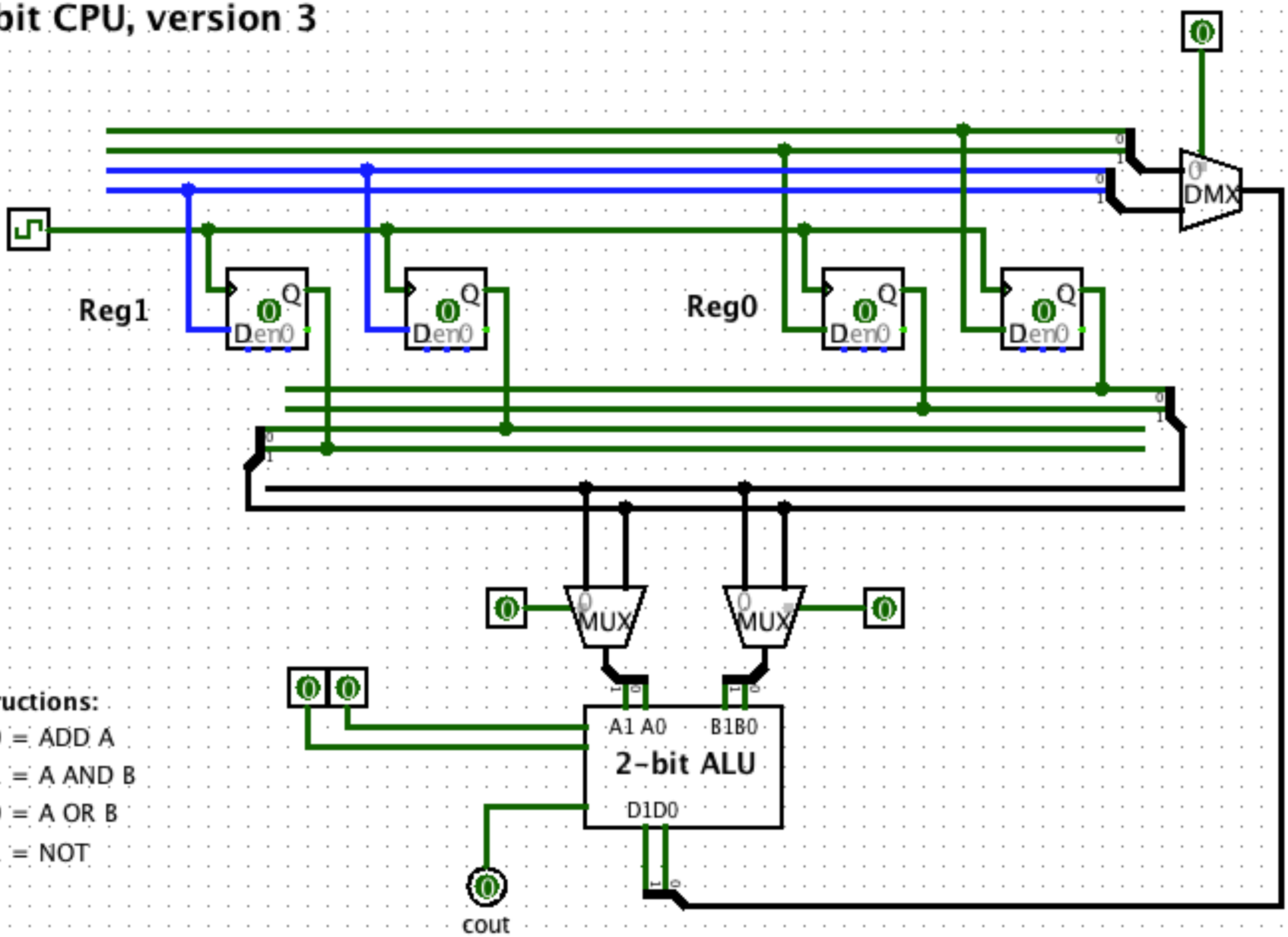
10 = A OR B

11 = NOT

2-BIT CPU: VERSION 3

Use MUX to select input to each ALU "port".

2-bit CPU, version 3



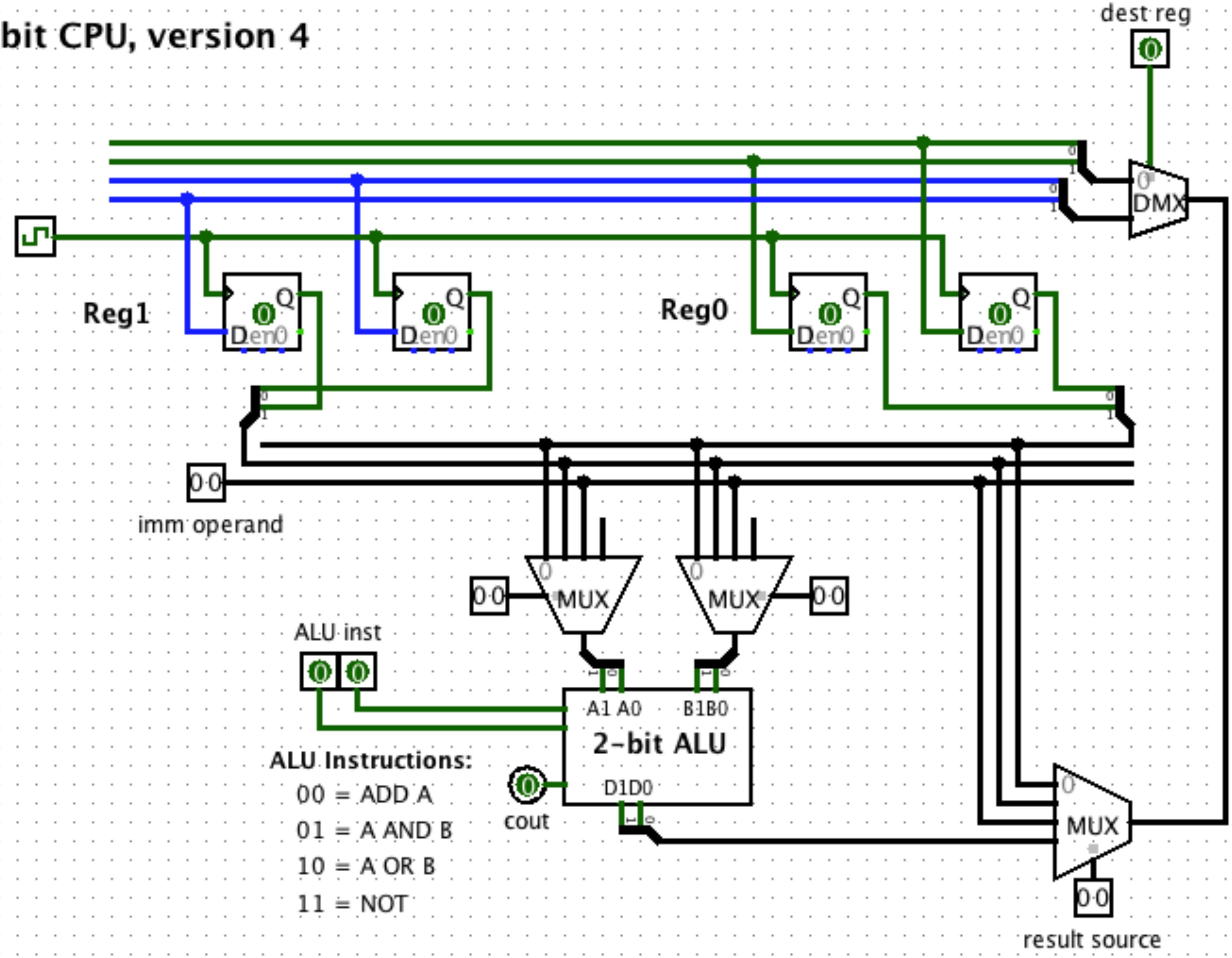
Instructions:

- 00 = ADD A
- 01 = A AND B
- 10 = A OR B
- 11 = NOT

2-BIT CPU: VERSION 4

- **Simplify "data bus" using wire bundles**
- **Add immediate operand to data bus**
- **Use result MUX to select input to DEMUX for destination register. Input may be:**
  - Register 0
  - Register 1
  - Immediate Operand
  - ALU output

2-bit CPU, version 4



ALU Instructions:

00 = ADD A

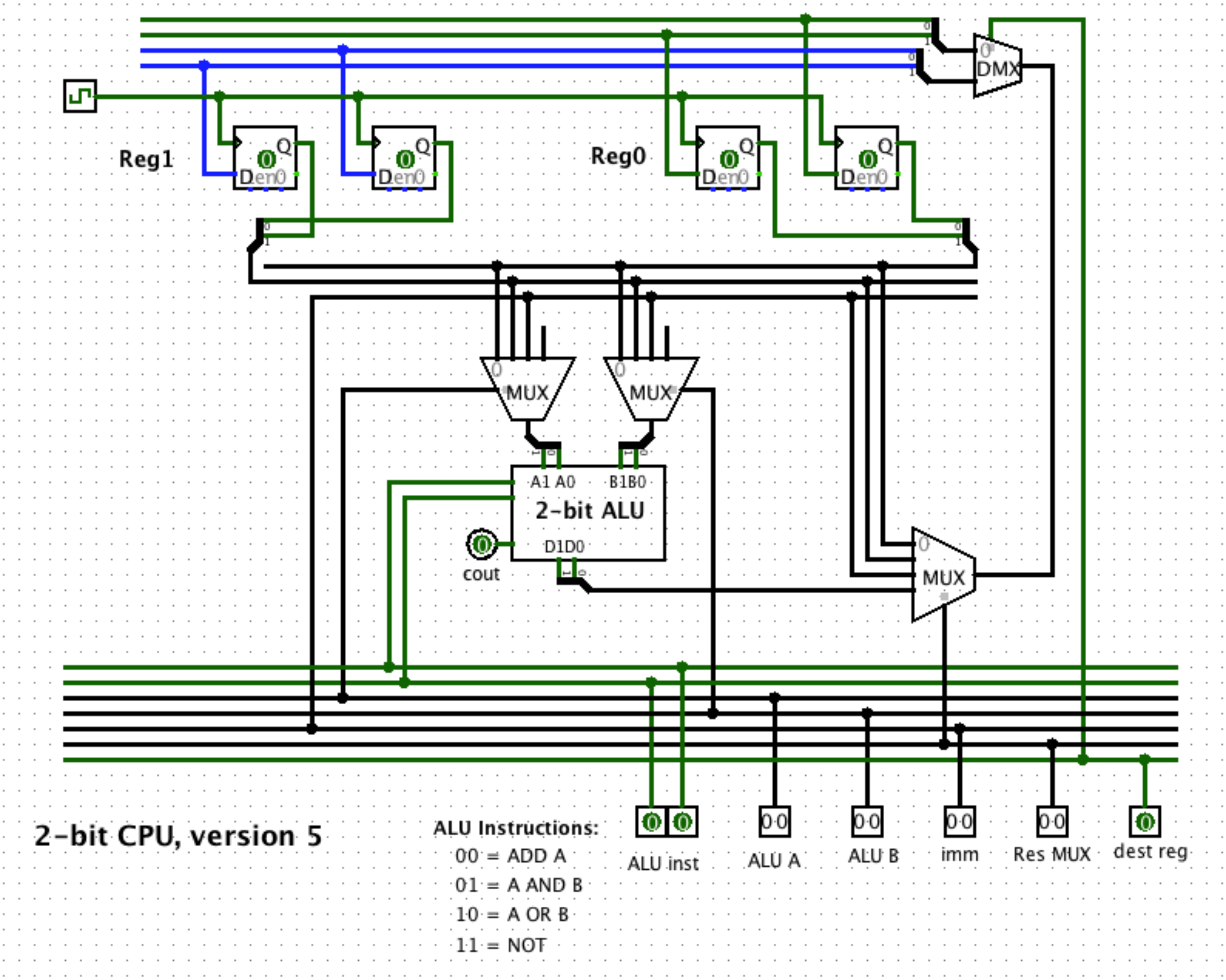
01 = A AND B

10 = A OR B

11 = NOT

2-BIT CPU: VERSION 5

Consolidate controls to a "control bus"



2-bit CPU, version 5

ALU Instructions:

00 = ADD A
01 = A AND B
10 = A OR B
11 = NOT

2-BIT CPU: VERSION 6

Use 8-bit "instruction code"

i7	i6	i5	i4	i3	i2	i1	i0
0							

`i7:`	0 if ALU instruction, 1 otherwise
`i6 i5:`	ALU instruction
`i4:`	operand 1 register (Reg 0 or Reg 1)
`i3 i2 i1:`	0rx = operand 2 is Reg r 1xy = immediate operand xy
`i0:`	destination register

2-BIT CPU: VERSION 6

Use 8-bit "instruction code"

i7	i6	i5	i4	i3	i2	i1	i0
1	0	0	0				

`i7:` **0 if ALU instruction, 1 otherwise**

`i6 i5 i4:` **000 = move, others not implemented**

`i3 i2 i1:` **0rx = source operand is Reg r**
1xy = immediate operand xy

`i0:` **destination register**

INSTRUCTION DECODER

MUX for ALU port B

$$B1 = i3$$

$$B0 = \overline{i3}\ i2\ \overline{i1} + \overline{i3}\ i2\ i1$$

$$= \overline{i3}\ i2$$

i3	i2	i1	B1	B0	
0	0	0	0	0	Reg 0
0	0	1	0	0	
0	1	0	0	1	Reg 1
0	1	1	0	1	
1	0	0	1	0	Imm
1	0	1	1	0	
1	1	0	1	0	
1	1	1	1	0	

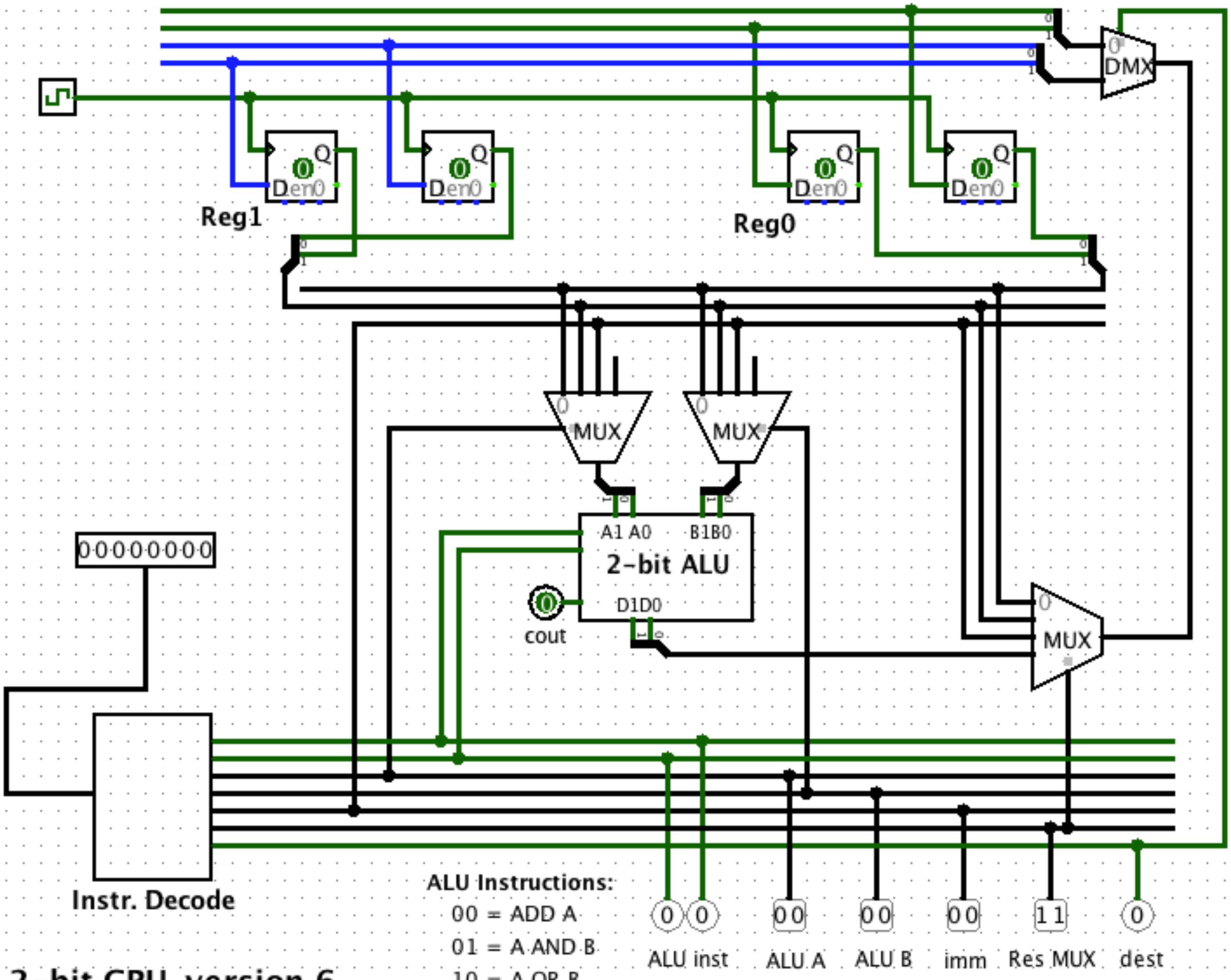
INSTRUCTION DECODER

Result MUX control

$$M1 = \overline{i7} + i3$$

$$M0 = \overline{i7} + \overline{i3}\ i2$$

i7	i3	i2	i1	M1	M0	
0	0	0	0	1	1	ALU
0	0	0	1	1	1	
0	0	1	0	1	1	
0	0	1	1	1	1	
0	1	0	0	1	1	
0	1	0	1	1	1	
0	1	1	0	1	1	
0	1	1	1	1	1	
1	0	0	0	0	0	Reg0
1	0	0	1	0	0	
1	0	1	0	0	1	Reg1
1	0	1	1	0	1	
1	1	0	0	1	0	Imm
1	1	0	1	1	0	
1	1	1	0	1	0	
1	1	1	1	1	0	



2-bit CPU, version 6

ALU Instructions:

00 = ADD A

01 = A AND B

10 = A OR B

11 = NOT

ALU MUX

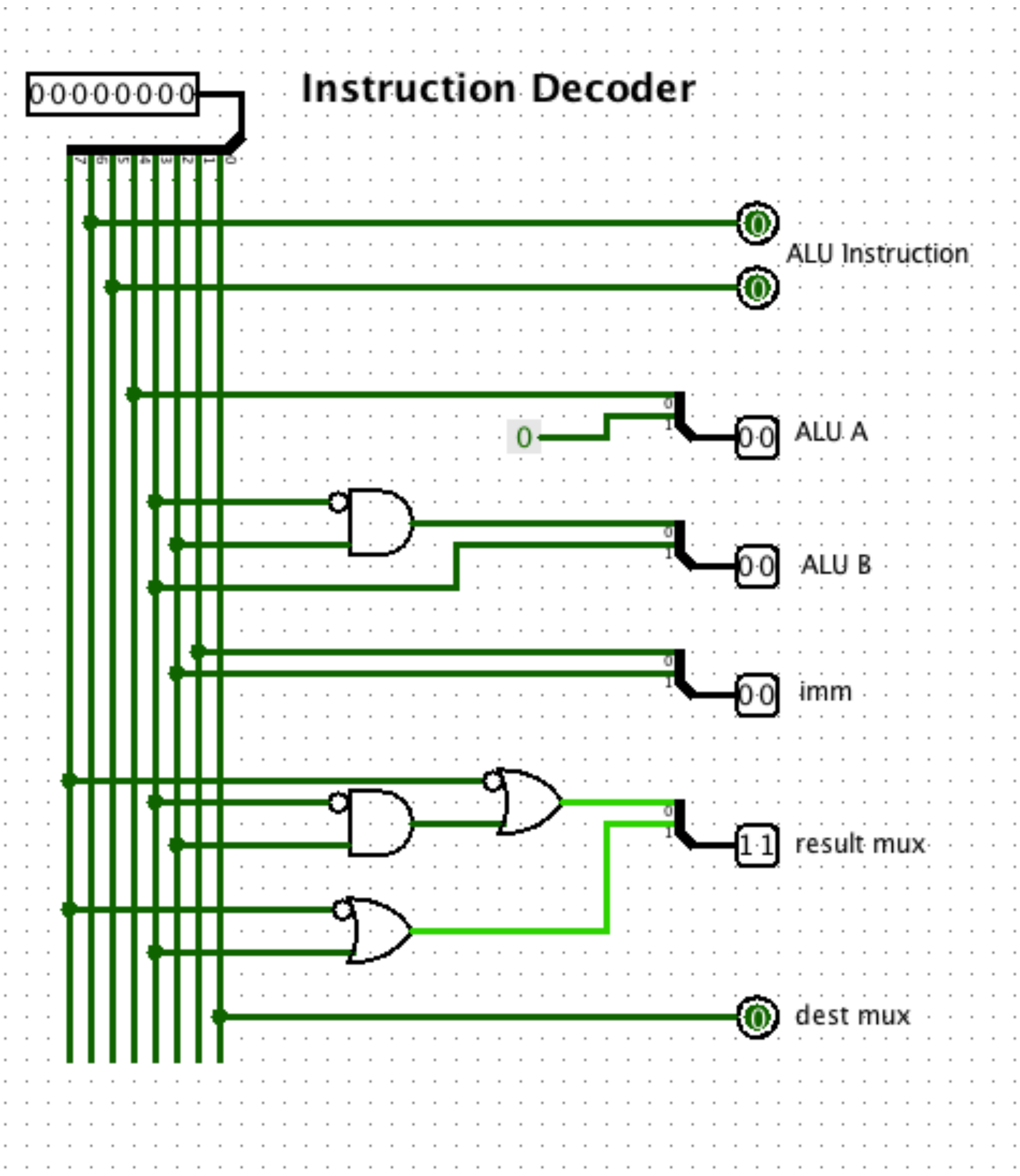
00=Reg0 01=Reg1

10=imm 11=xx

RES MUX

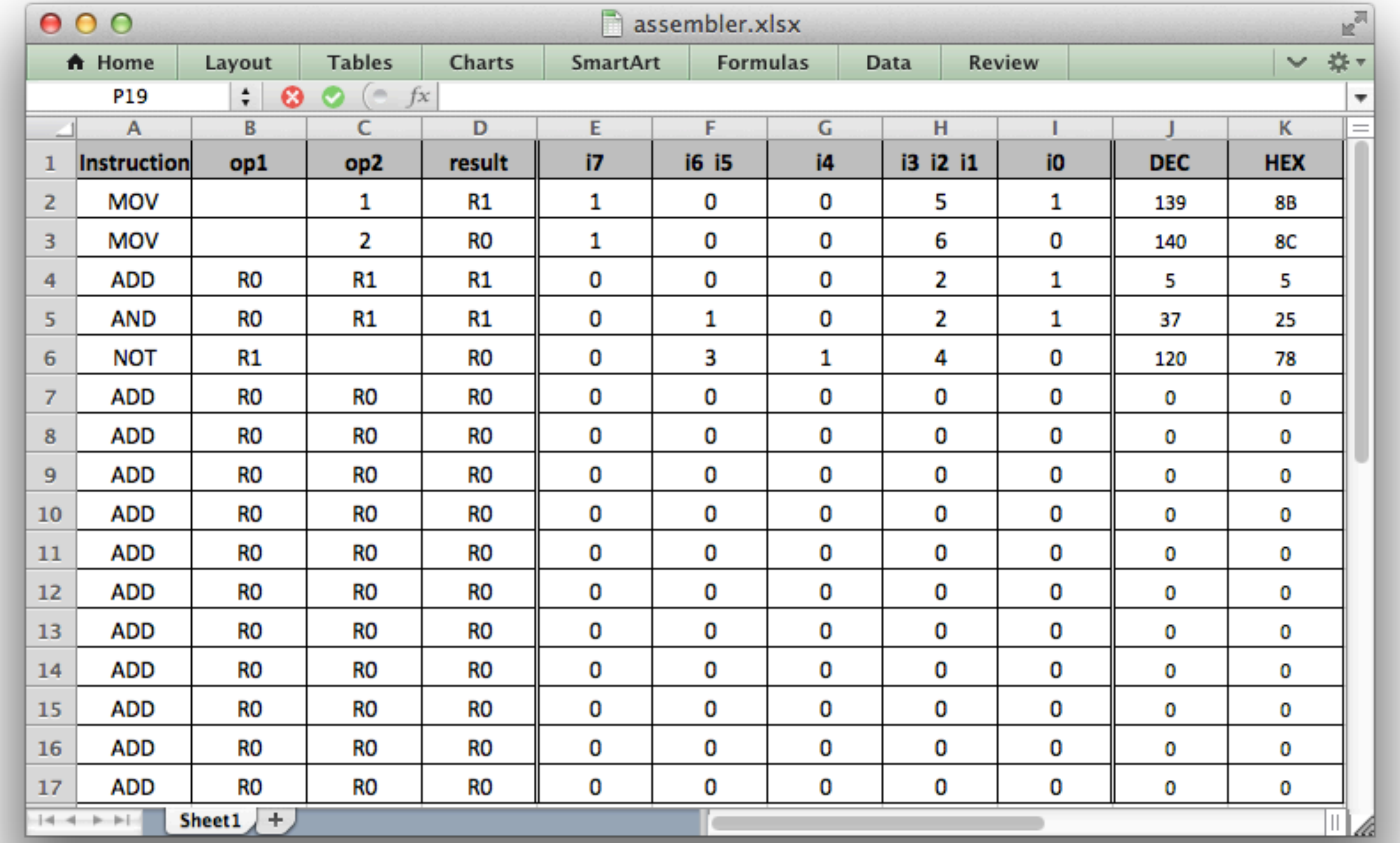
00=Reg0 01=Reg1

10=imm 11=ALU

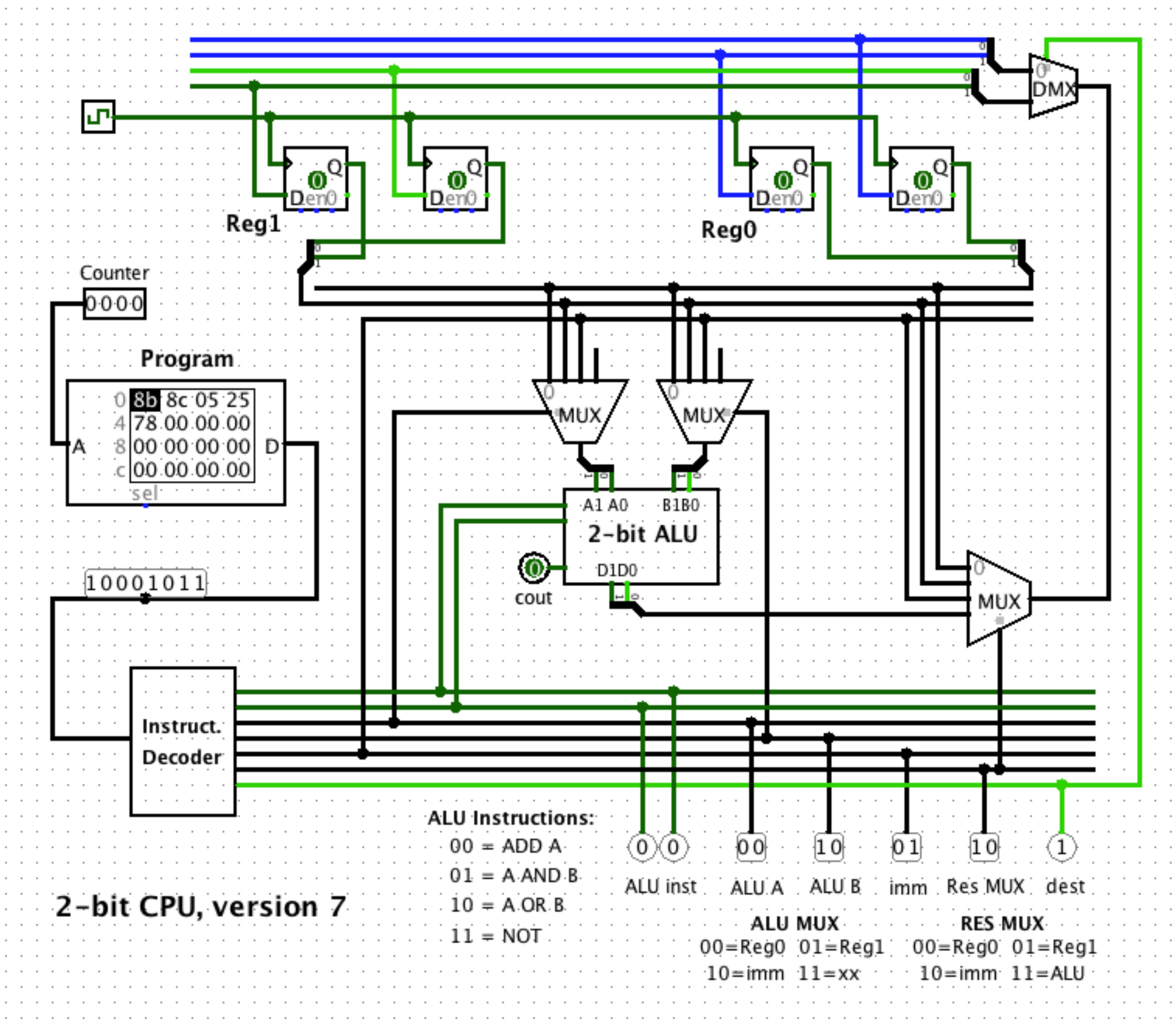

0·0·0·0·0·0·0·0
Instruction Decoder
7 6 5 4 3 2 1 0
0
0
ALU Instruction
0
1
0
0·0
ALU A
0
1
0·0
ALU B
0
1
0·0
imm
0
1
1·1
result mux
0
dest mux

2-BIT CPU: VERSION 7

Added Program ROM which can store up to 16 instructions.



	A	B	C	D	E	F	G	H	I	J	K
1	Instruction	op1	op2	result	i7	i6 i5	i4	i3 i2 i1	i0	DEC	HEX
2	MOV		1	R1	1	0	0	5	1	139	8B
3	MOV		2	R0	1	0	0	6	0	140	8C
4	ADD	R0	R1	R1	0	0	0	2	1	5	5
5	AND	R0	R1	R1	0	1	0	2	1	37	25
6	NOT	R1		R0	0	3	1	4	0	120	78
7	ADD	R0	R0	R0	0	0	0	0	0	0	0
8	ADD	R0	R0	R0	0	0	0	0	0	0	0
9	ADD	R0	R0	R0	0	0	0	0	0	0	0
10	ADD	R0	R0	R0	0	0	0	0	0	0	0
11	ADD	R0	R0	R0	0	0	0	0	0	0	0
12	ADD	R0	R0	R0	0	0	0	0	0	0	0
13	ADD	R0	R0	R0	0	0	0	0	0	0	0
14	ADD	R0	R0	R0	0	0	0	0	0	0	0
15	ADD	R0	R0	R0	0	0	0	0	0	0	0
16	ADD	R0	R0	R0	0	0	0	0	0	0	0
17	ADD	R0	R0	R0	0	0	0	0	0	0	0



```
0 8b 8c 05 25
4 78 00 00 00
8 00 00 00 00
c 00 00 00 00
```

2-bit CPU, version 7

ALU Instructions:

00 = ADD A
01 = A AND B
10 = A OR B
11 = NOT

ALU MUX

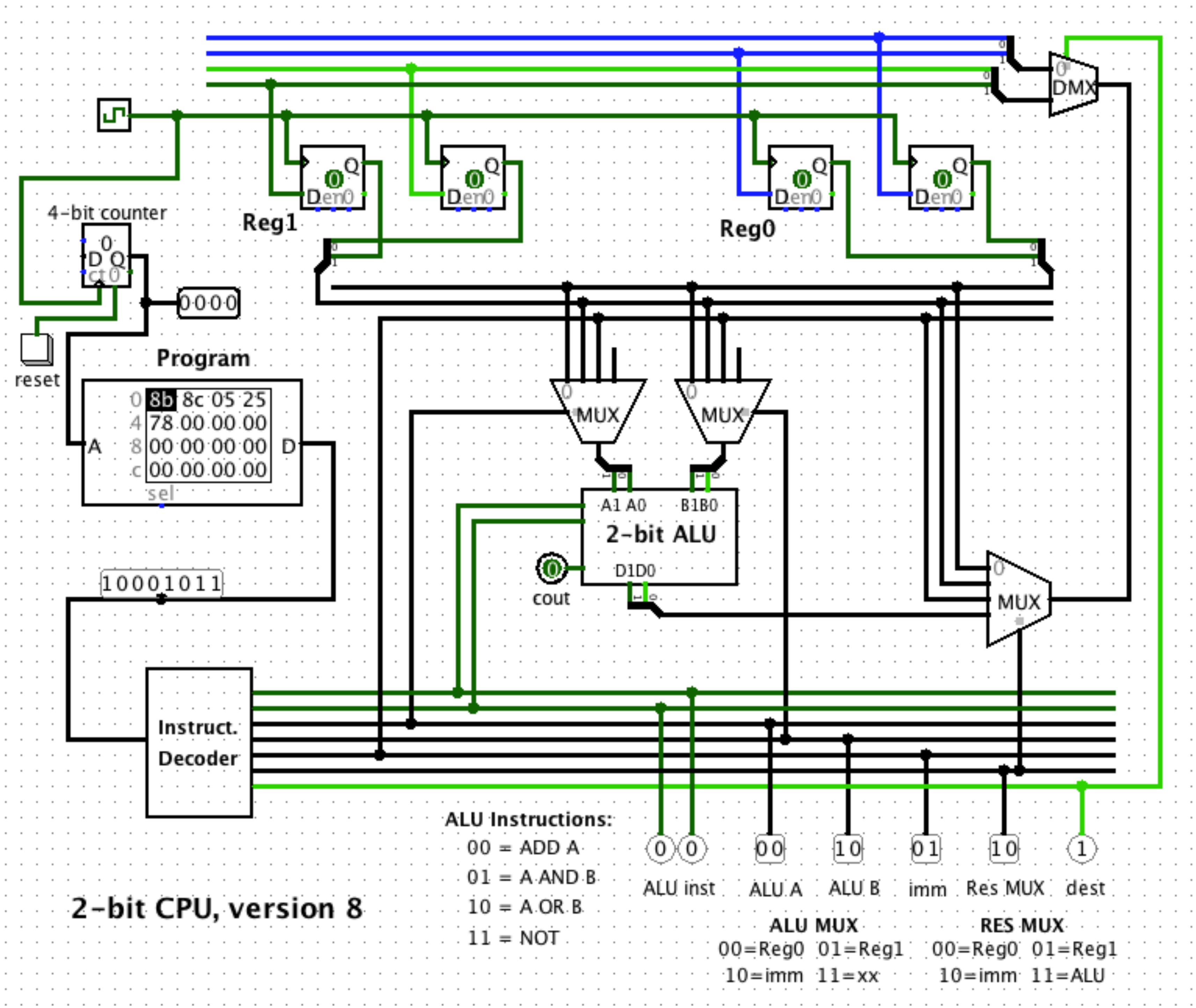
00=Reg0 01=Reg1
10=imm 11=xx

RES MUX

00=Reg0 01=Reg1
10=imm 11=ALU

2-BIT CPU: VERSION 8

Added 4-bit counter which automatically advances Program ROM to next instruction.



2-bit CPU, version 8

ALU Instructions:

00 = ADD A
01 = A AND B
10 = A OR B
11 = NOT

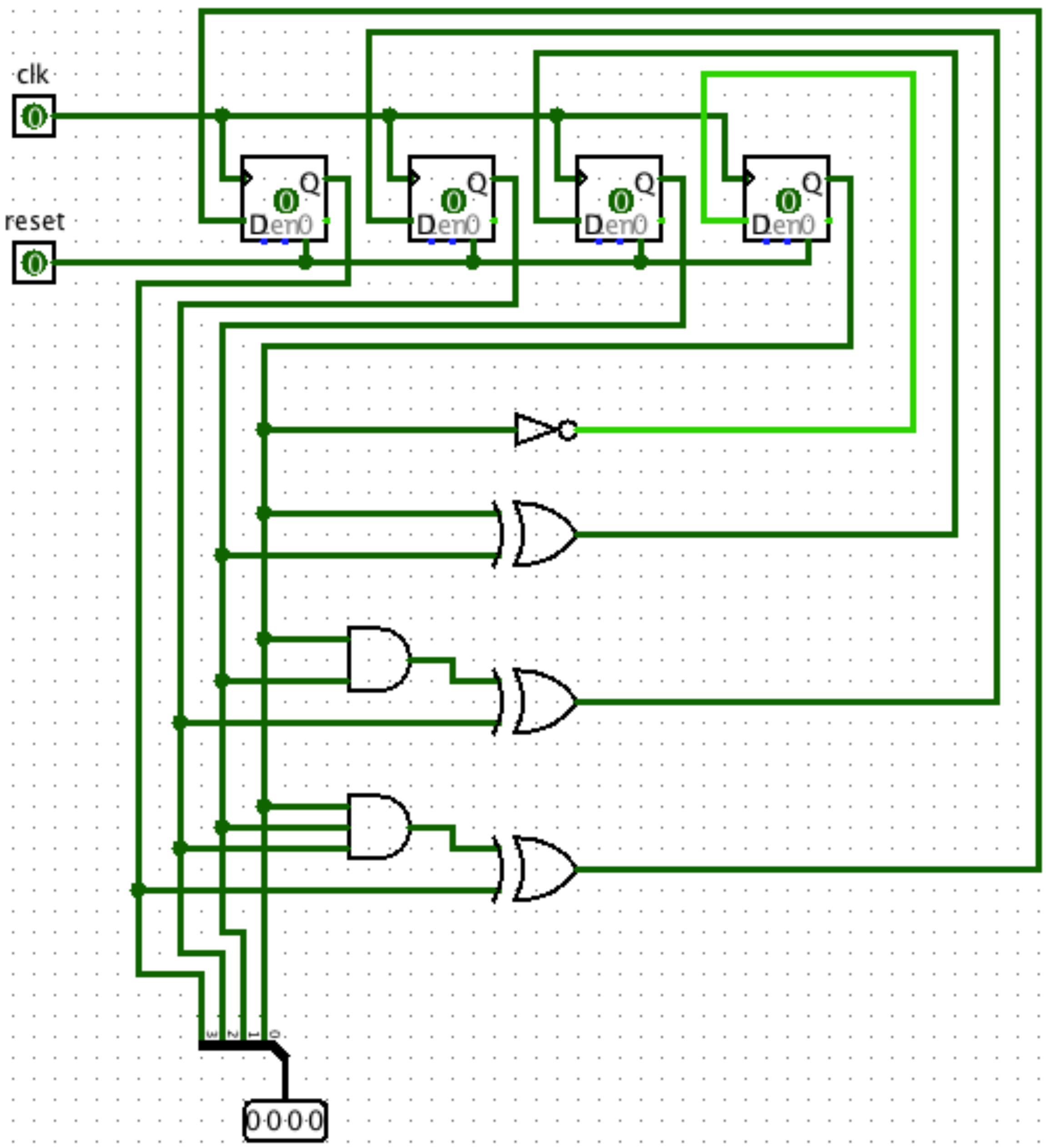
ALU MUX
00=Reg0 01=Reg1
10=imm 11=xx

RES MUX
00=Reg0 01=Reg1
10=imm 11=ALU

2-BIT CPU: VERSION 9

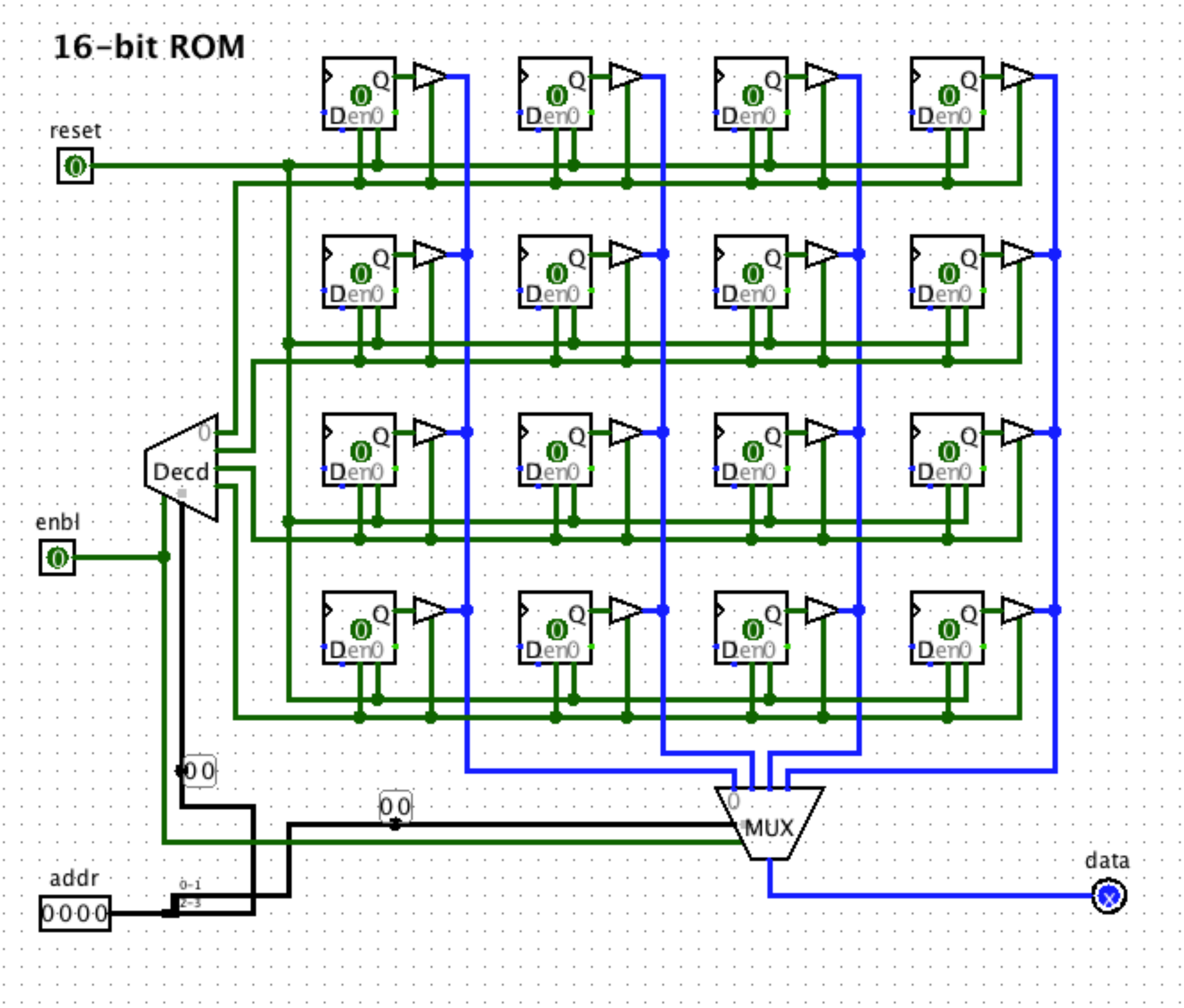
Implement 4-bit counter from scratch.

4-bit Counter

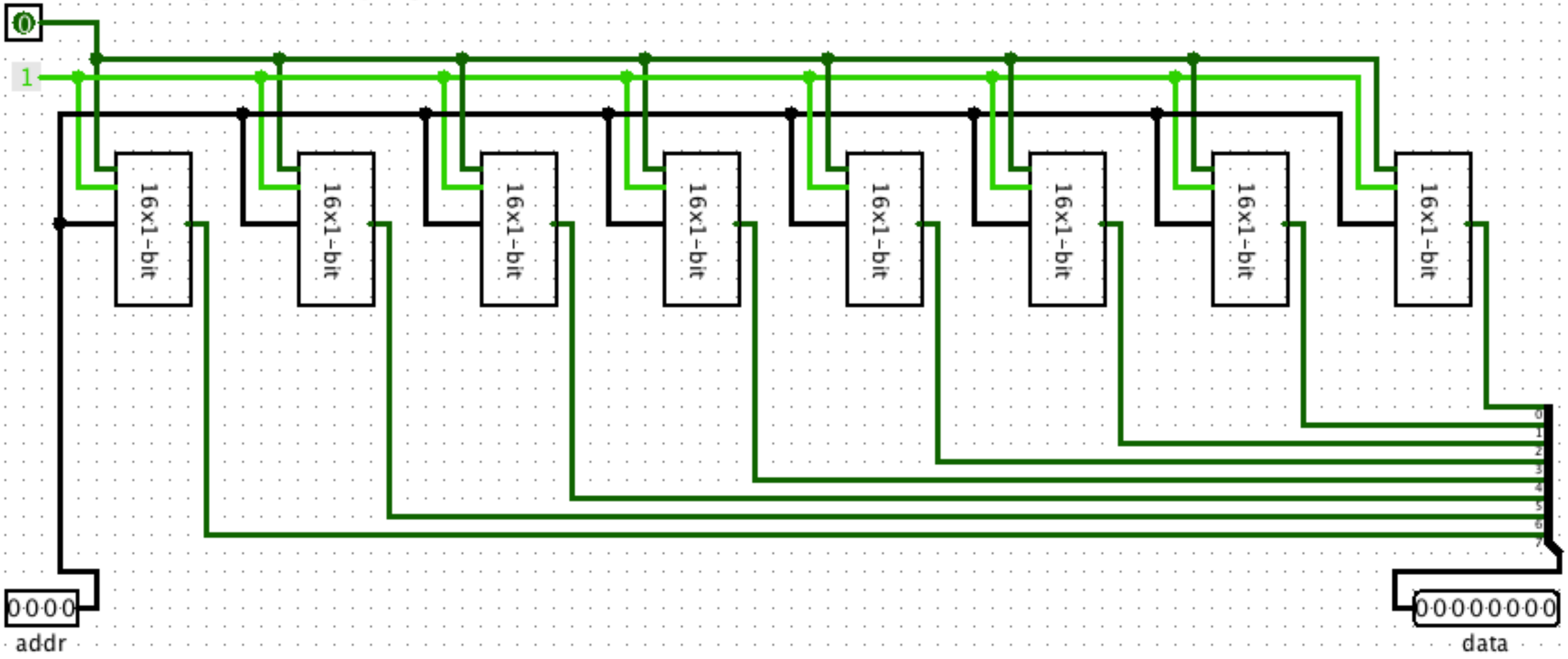


2-BIT CPU: VERSION 10

Implement Program ROM from scratch.


16-bit ROM
reset
0
enbl
0
Decd
0
Q
D en 0
0 0
0 0
MUX
0
addr
0 0 0 0
0-1
2-3
data

8 x 16-bit ROM


0
1
16x1-bit
16x1-bit
16x1-bit
16x1-bit
16x1-bit
16x1-bit
16x1-bit
16x1-bit
0
1
2
3
4
5
6
7
0·0·0·0
addr
0·0·0·0·0·0·0·0
data

NEXT TIME

- Memory Hierarchy
- Virtual Memory