CMSC 313
COMPUTER ORGANIZATION
&
ASSEMBLY LANGUAGE PROGRAMMING

LECTURE 25, SPRING 2013

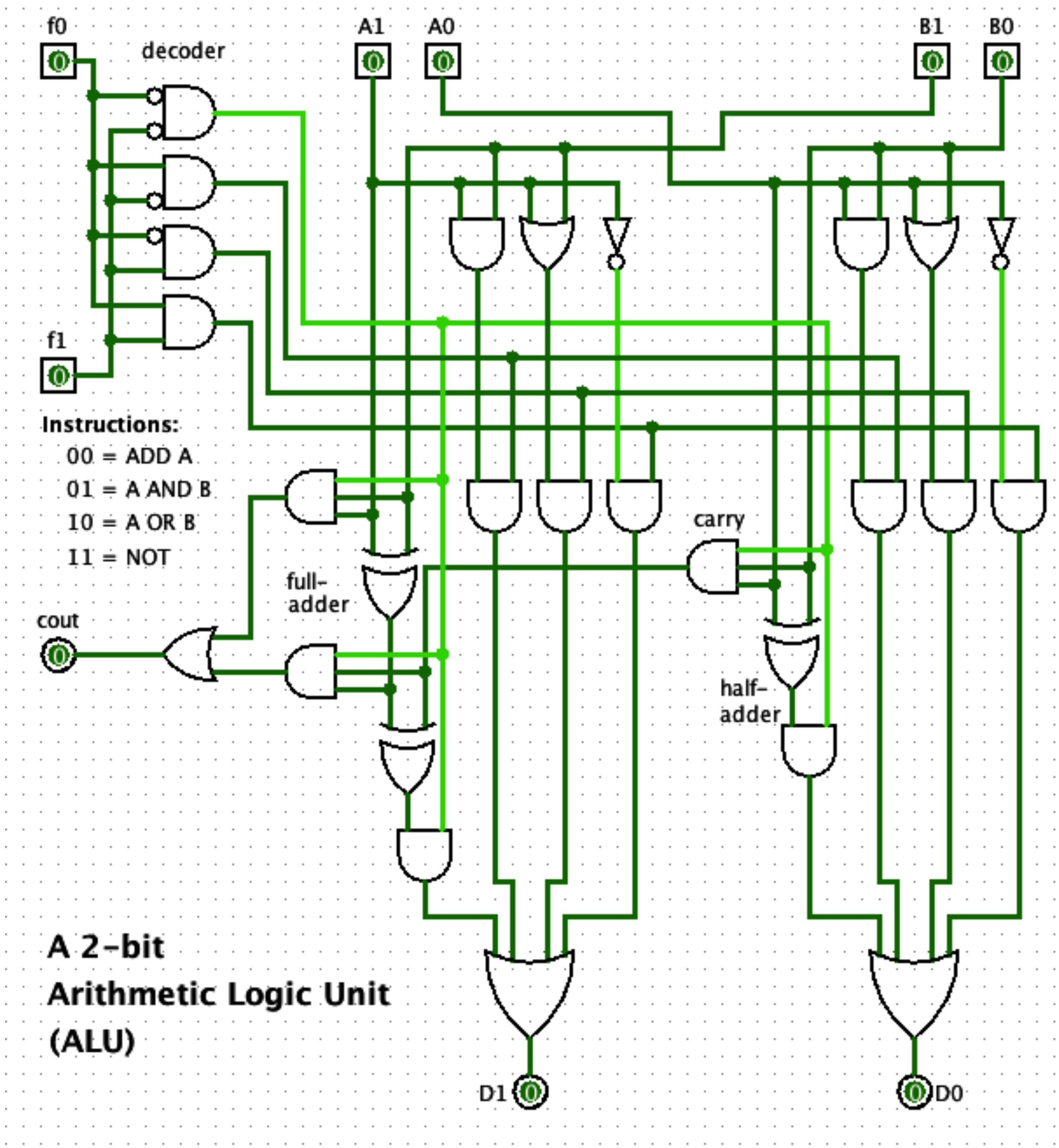
TOPICS TODAY

- A 2-bit "CPU"

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

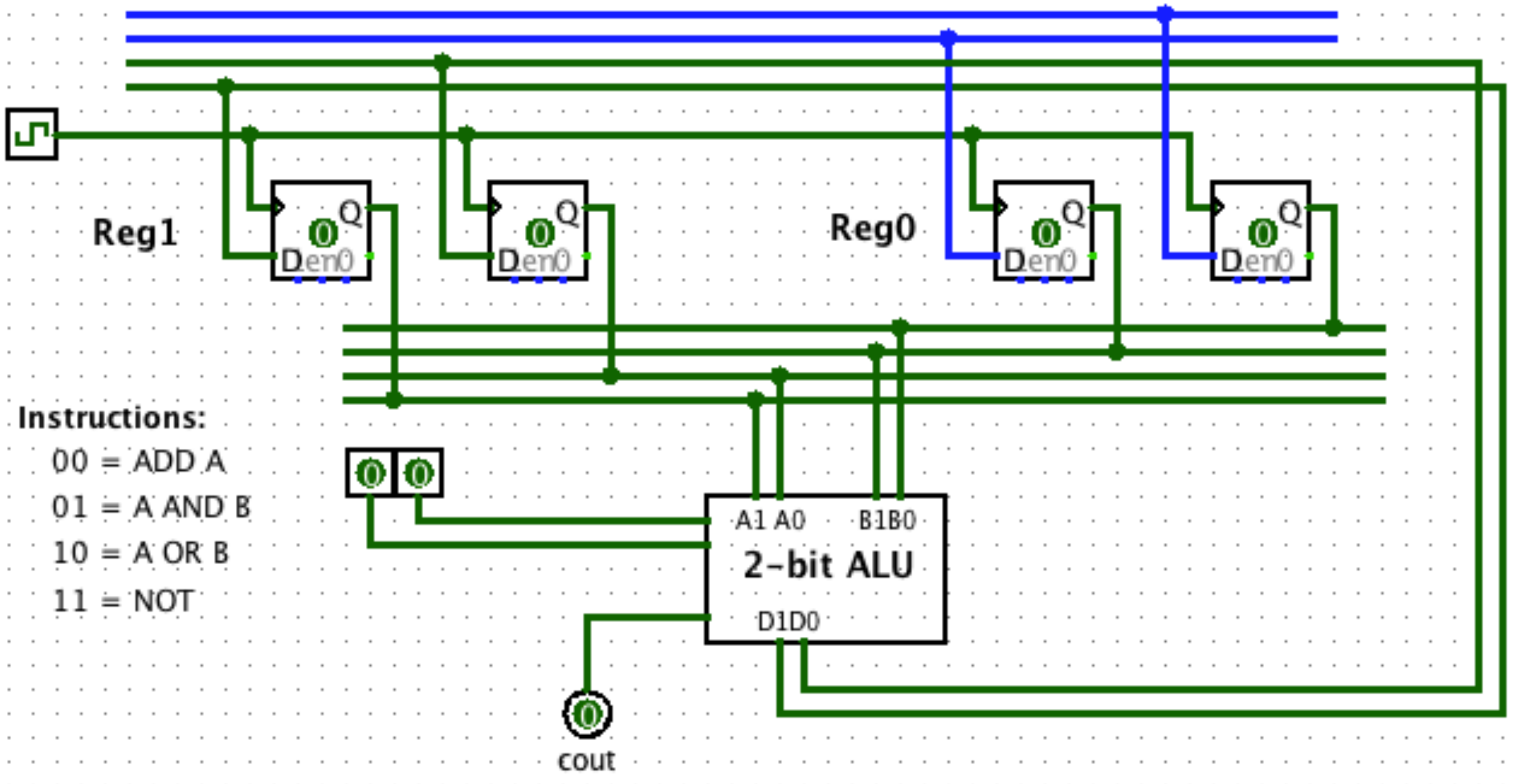
01 = A AND B

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

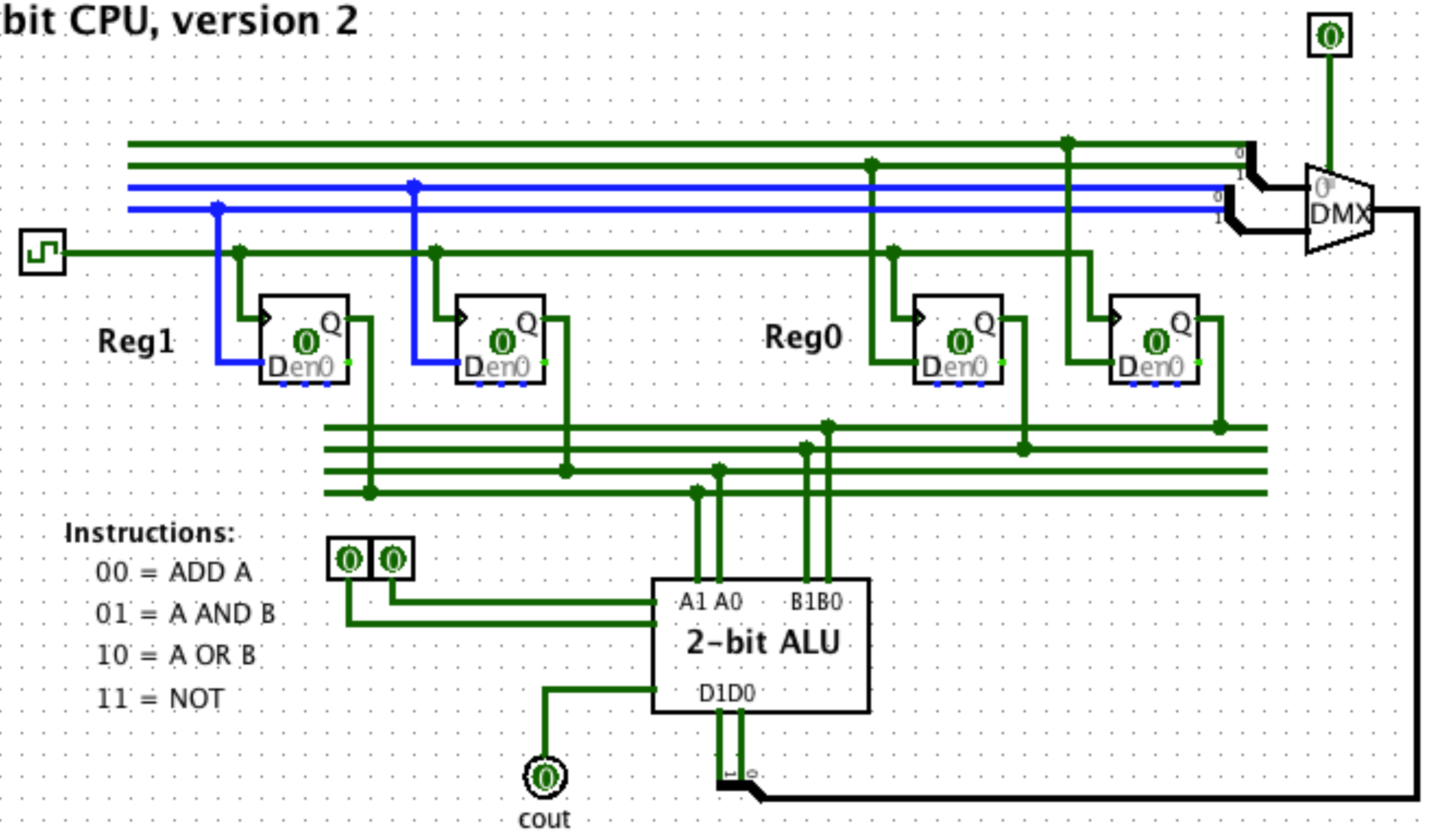
10 = A OR B

11 = NOT

2-BIT CPU: VERSION 2

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

2-bit CPU, version 2



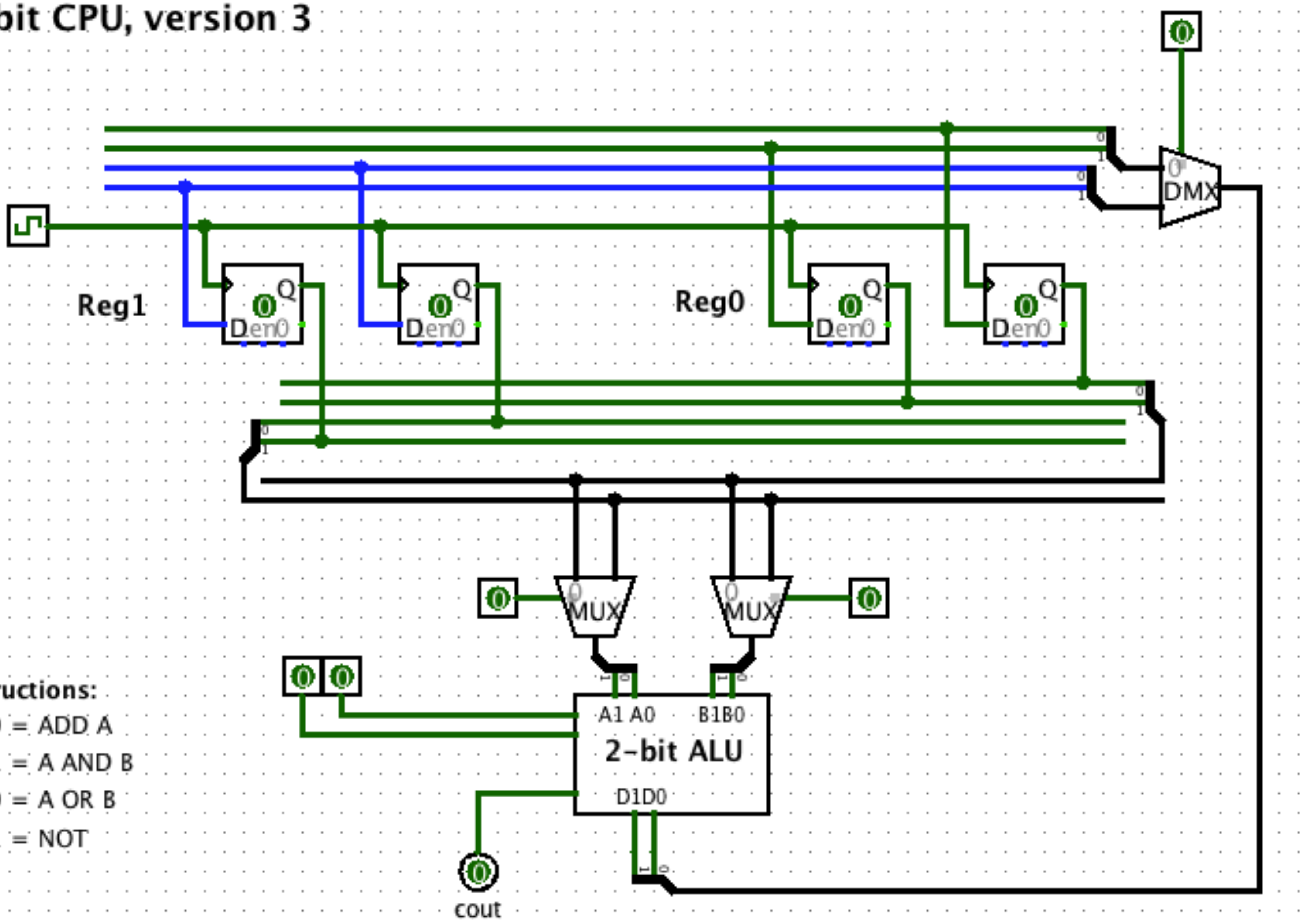
Instructions:

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

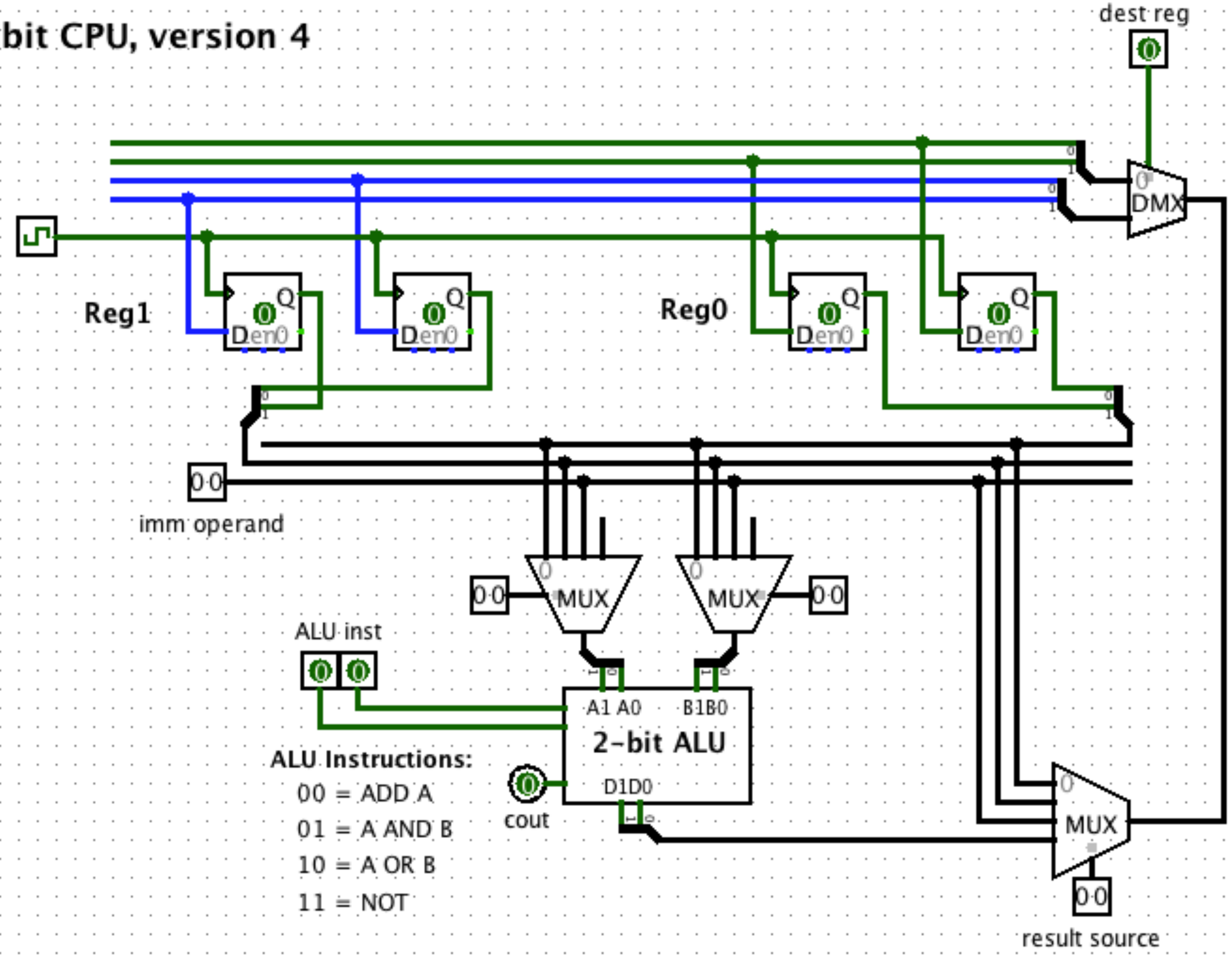
10 = A OR B

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

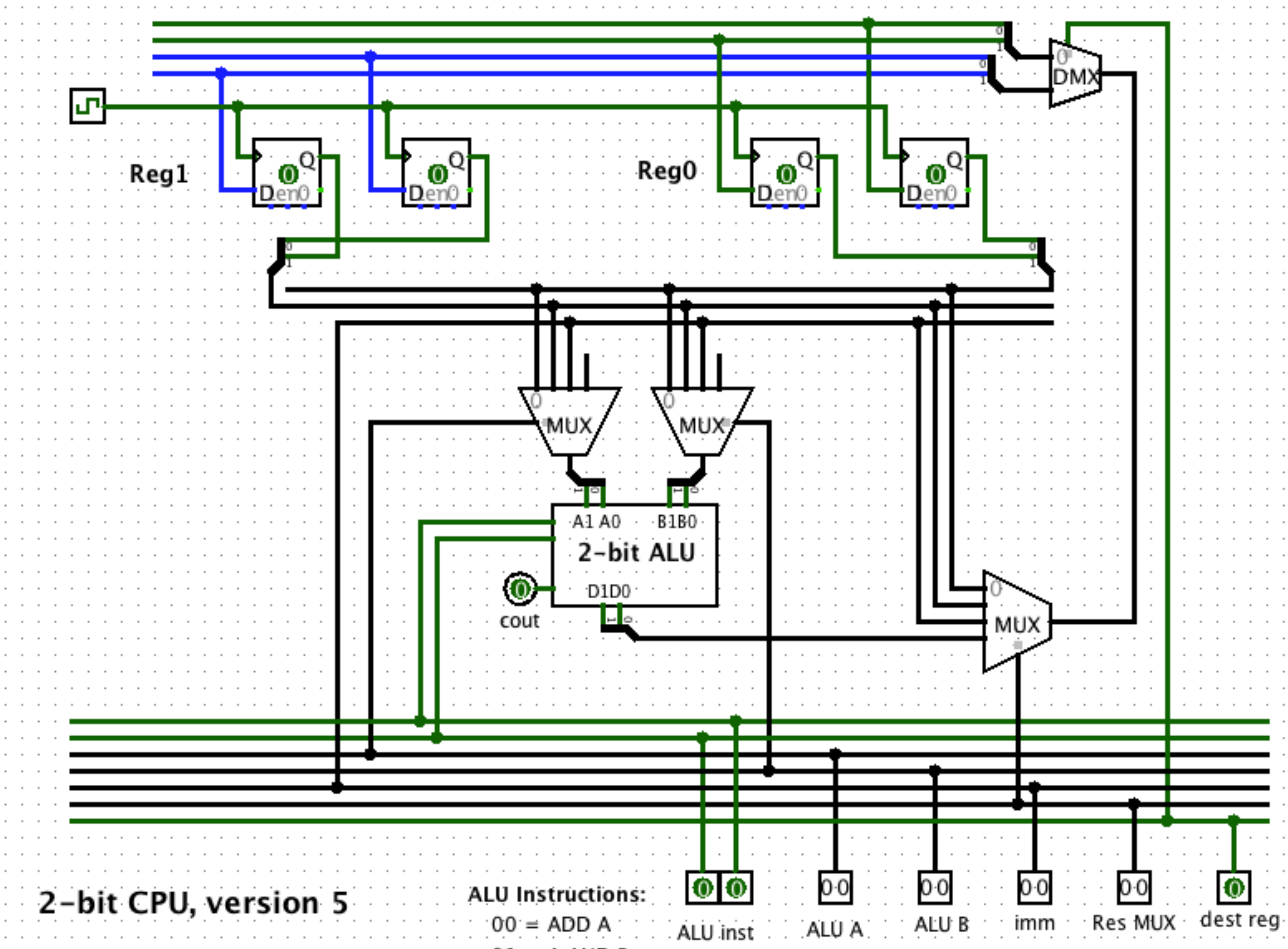


ALU Instructions:

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	

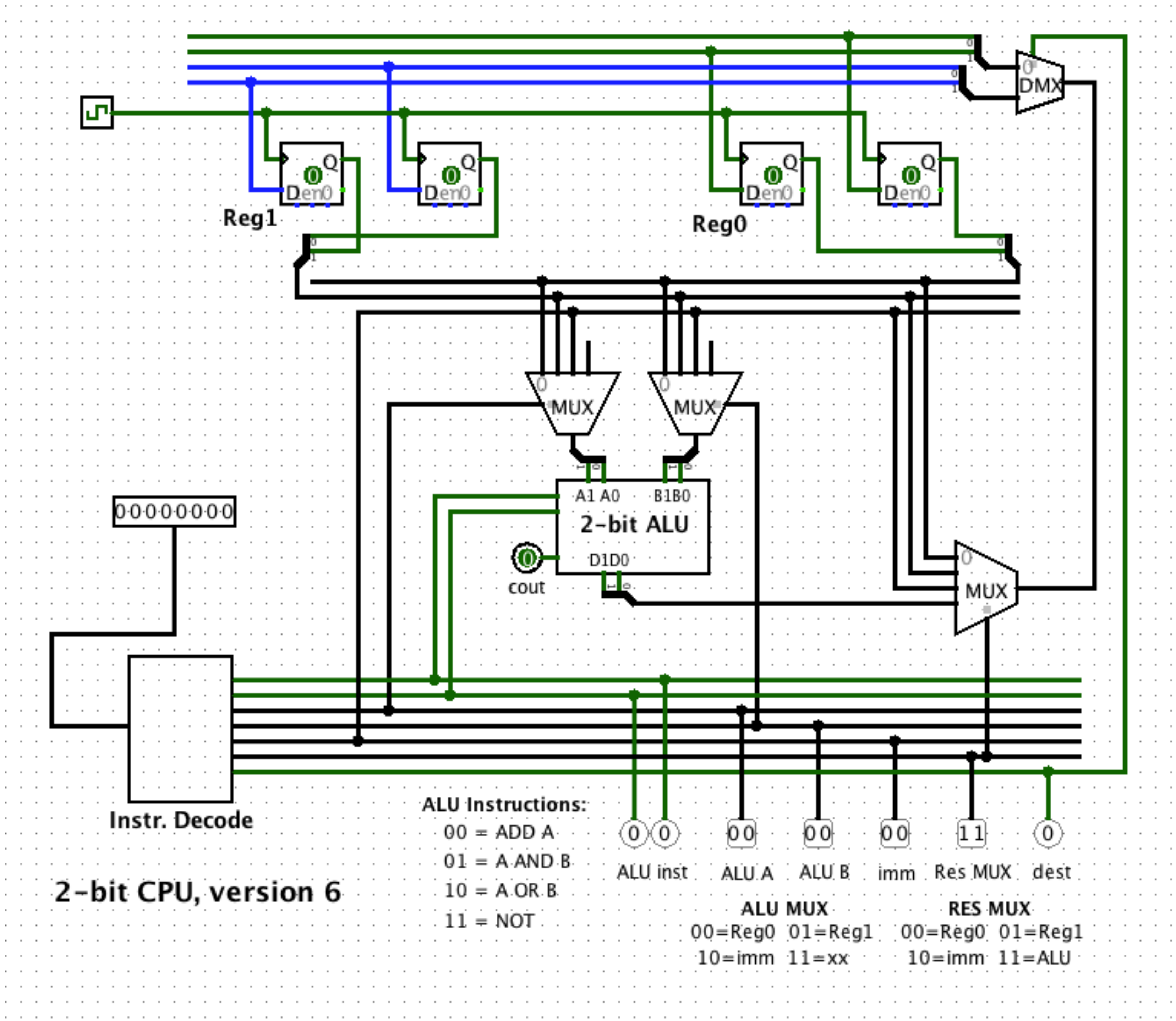
INSTRUCTION DECODER

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	
0	0	0	1	1	1	
0	0	1	0	1	1	
0	0	1	1	1	1	ALU
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	
1	1	0	1	1	0	Imm
1	1	1	0	1	0	
1	1	1	1	1	0	



2-bit CPU, version 6

ALU Instructions:

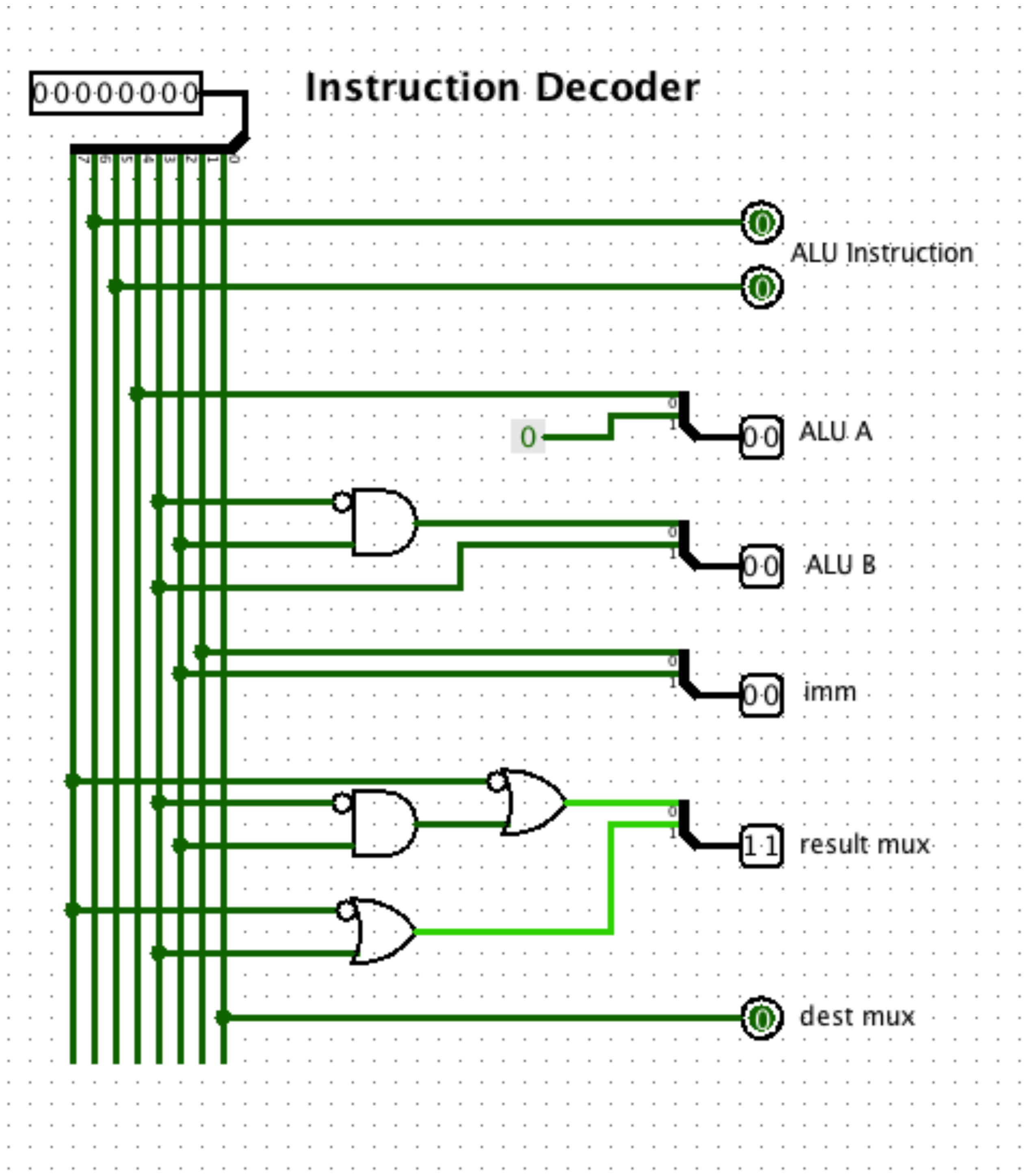
00	= ADD A
01	= A AND B
10	= A OR B
11	= NOT

ALU MUX

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

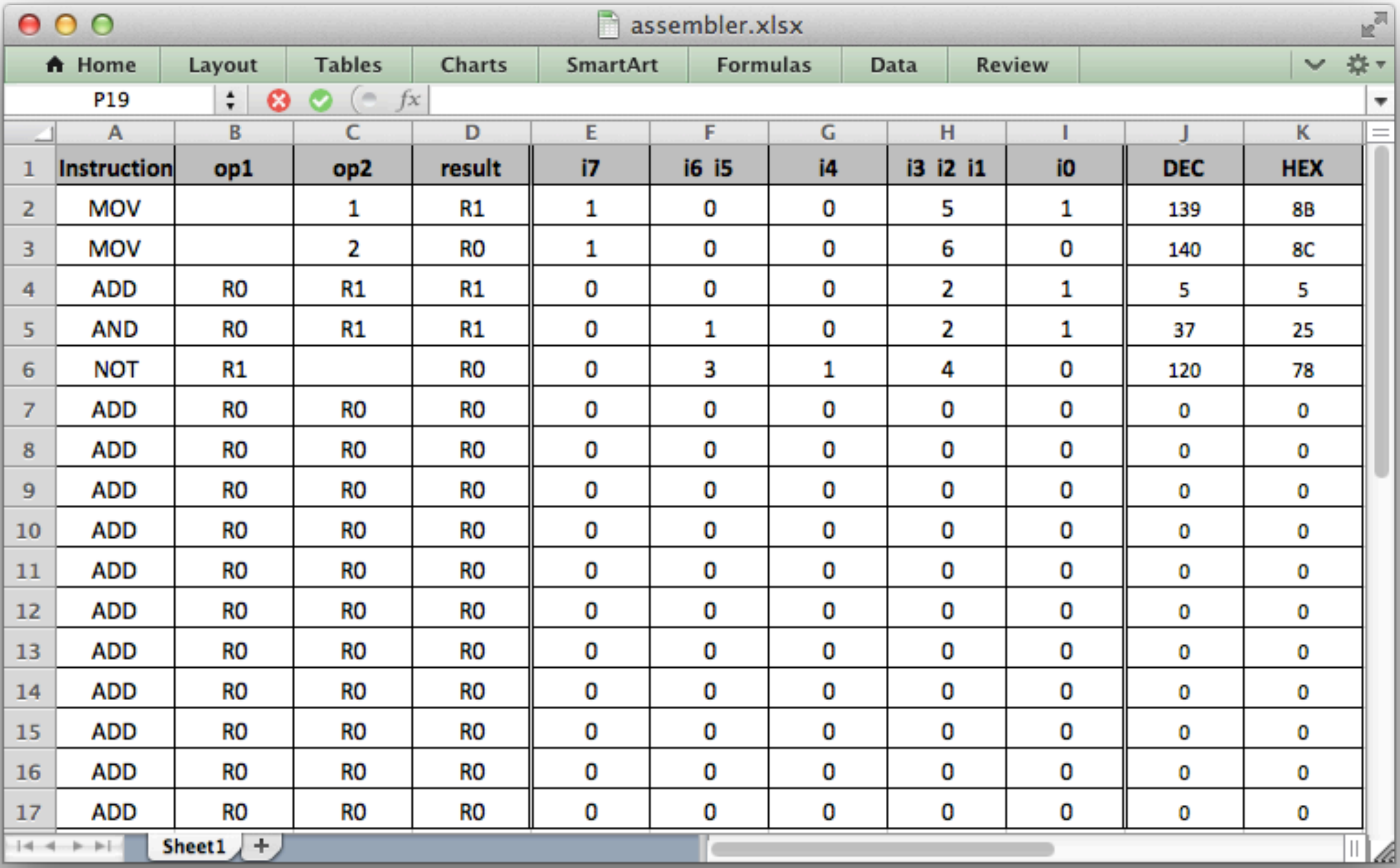
RES MUX

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

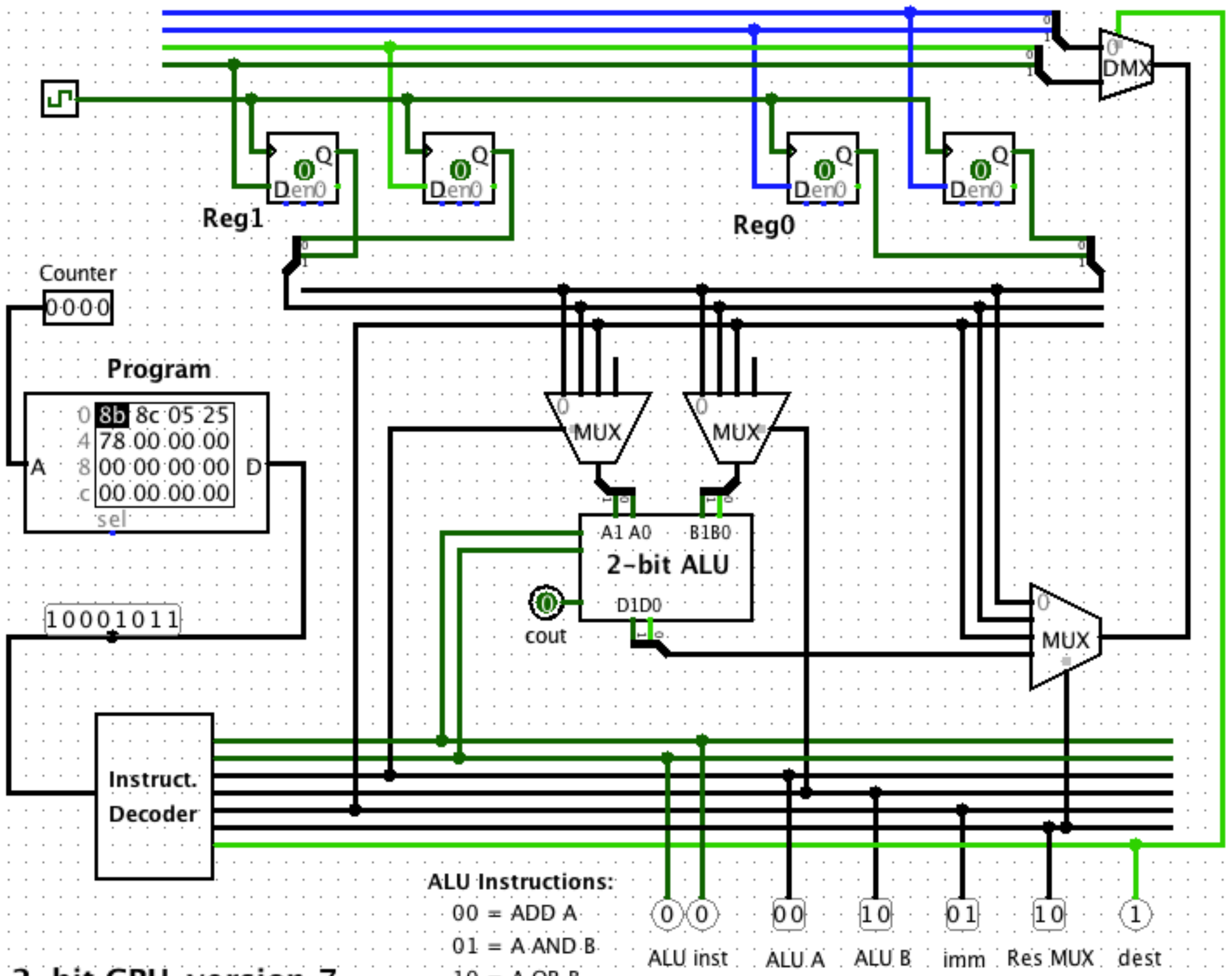

Instruction Decoder
0·0·0·0·0·0·0·0
7 6 5 4 3 2 1 0
0
0
ALU Instruction
0
0·0
ALU A
0·0
ALU B
0·0
imm
1·1
result mux
0
dest mux

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



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

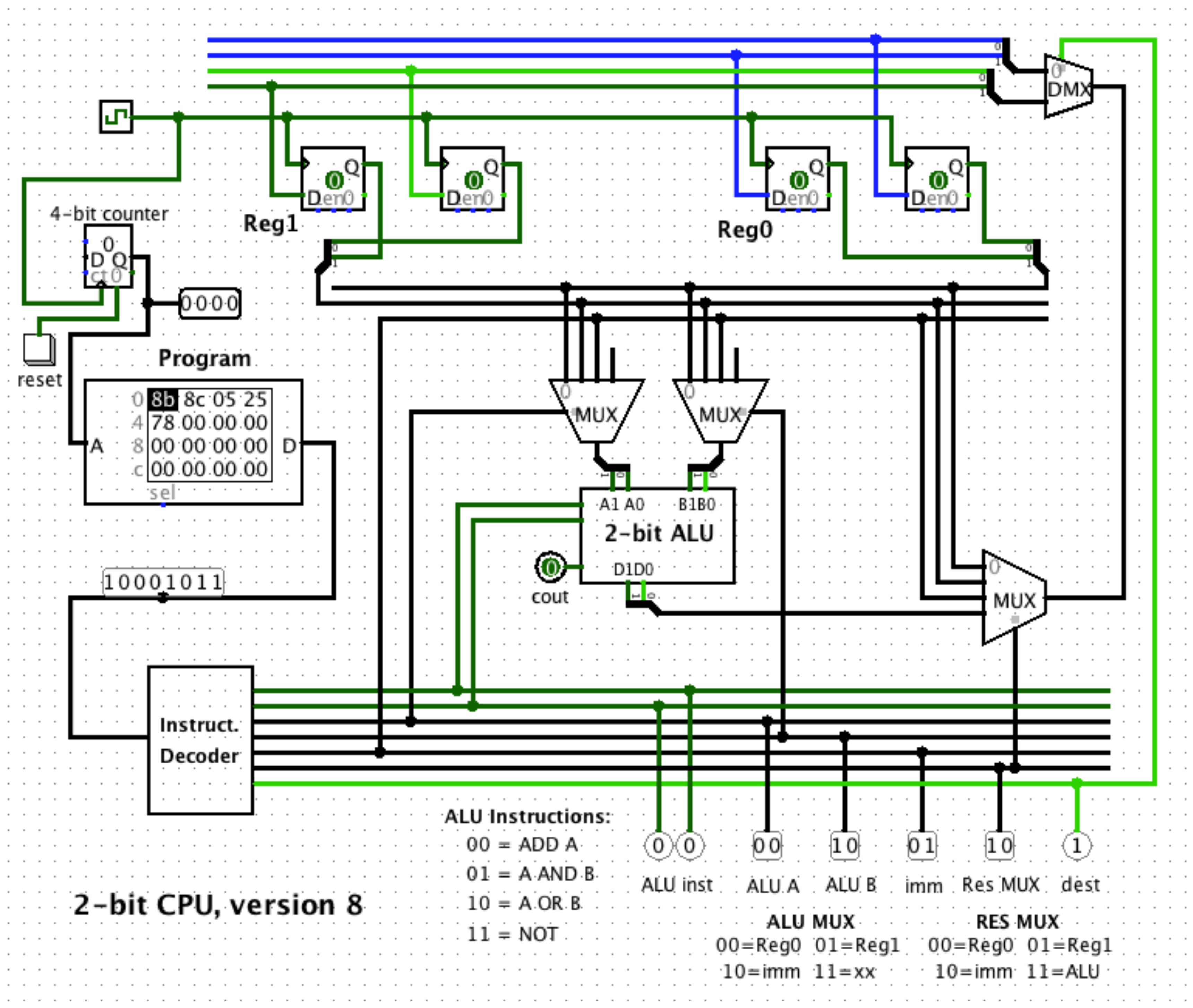
RES MUX

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.



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

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

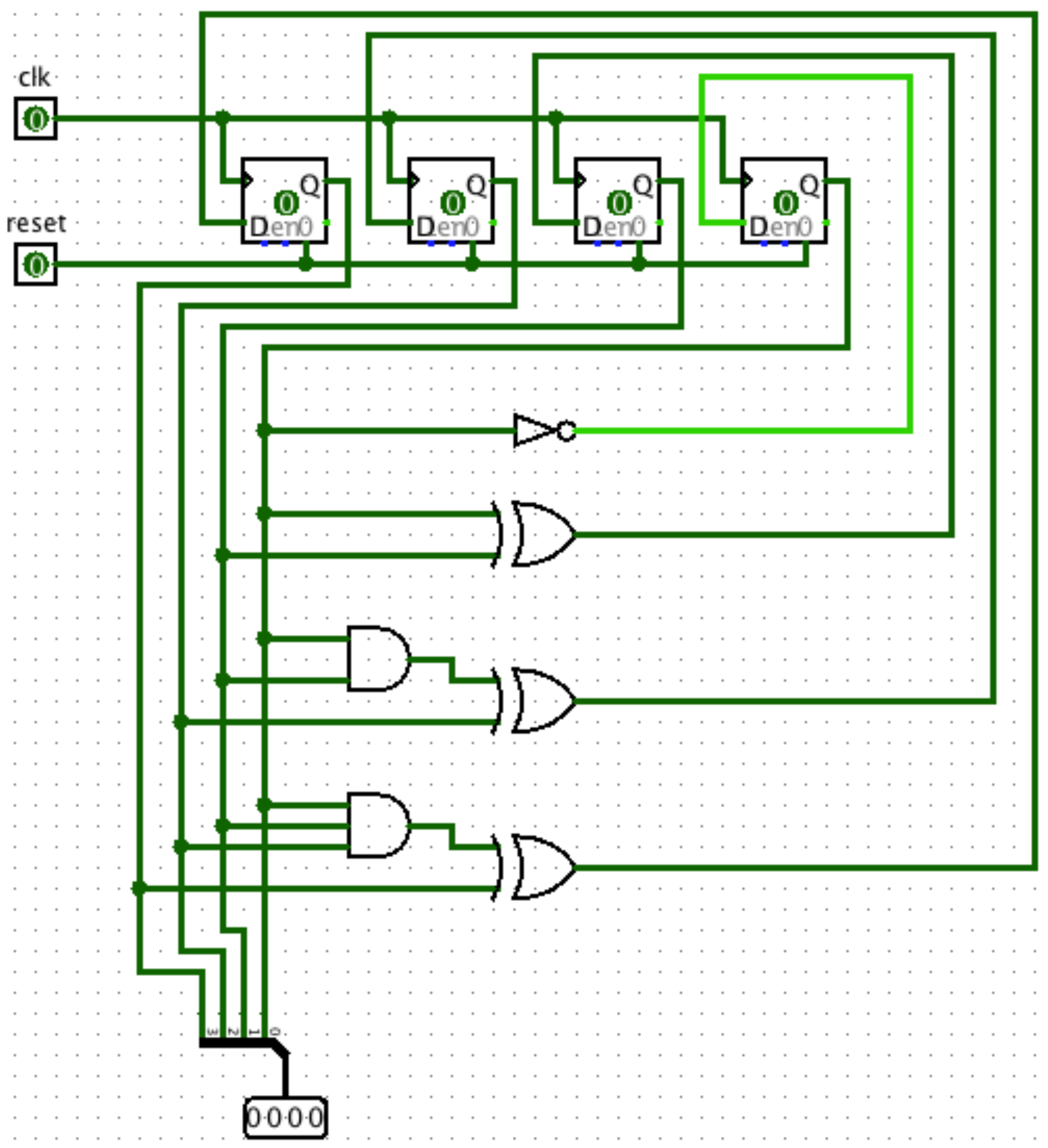
RES MUX

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

2-BIT CPU: VERSION 9

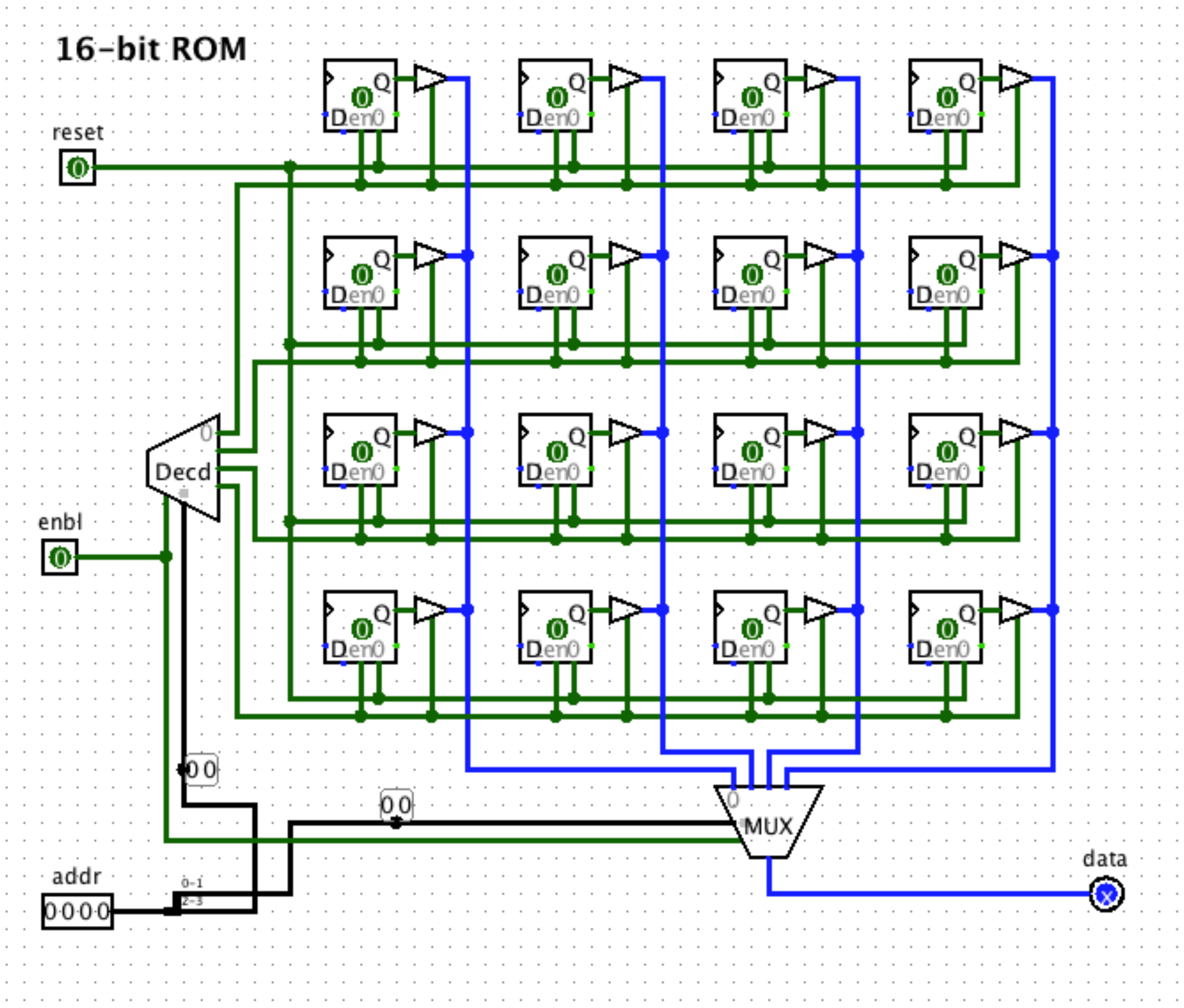
Implement 4-bit counter from scratch.

4-bit Counter

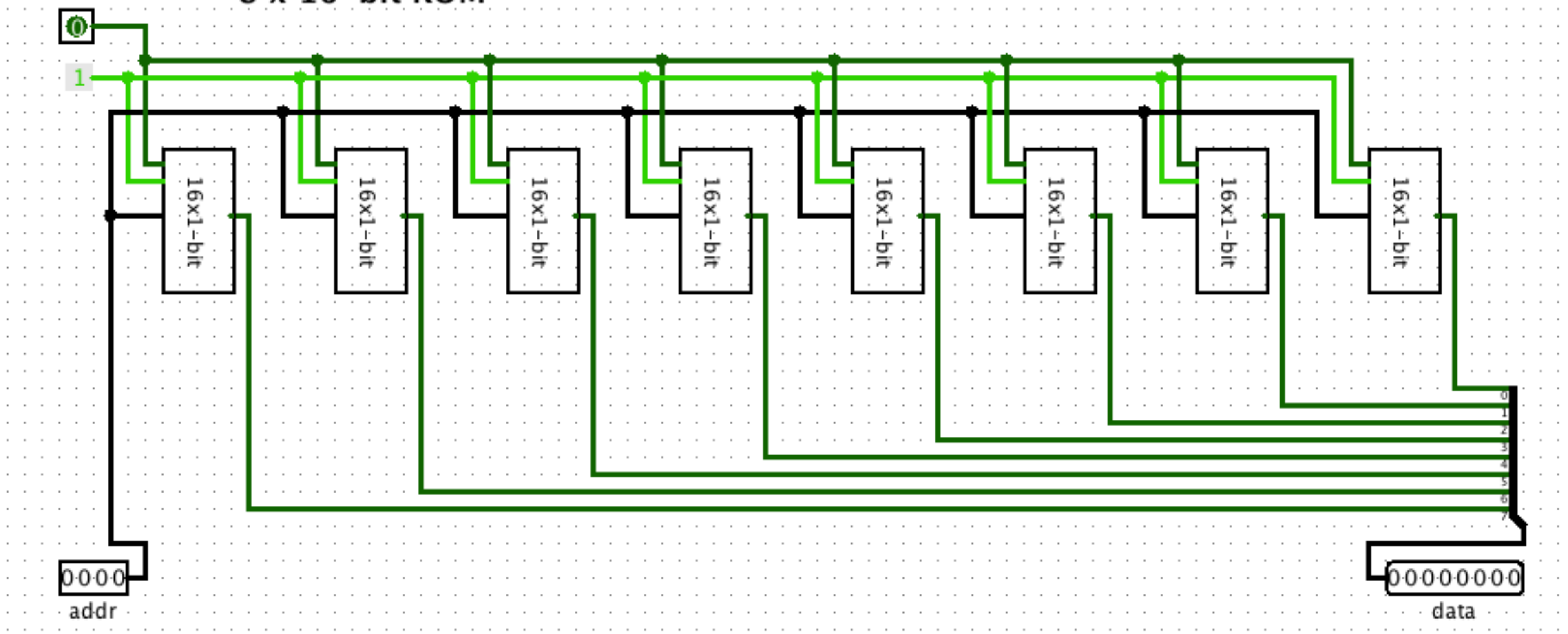


2-BIT CPU: VERSION 10

Implement Program ROM from scratch.


16-bit ROM
reset
enbl
addr
0000
Decd
MUX
data
0-1
2-3
00
00

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