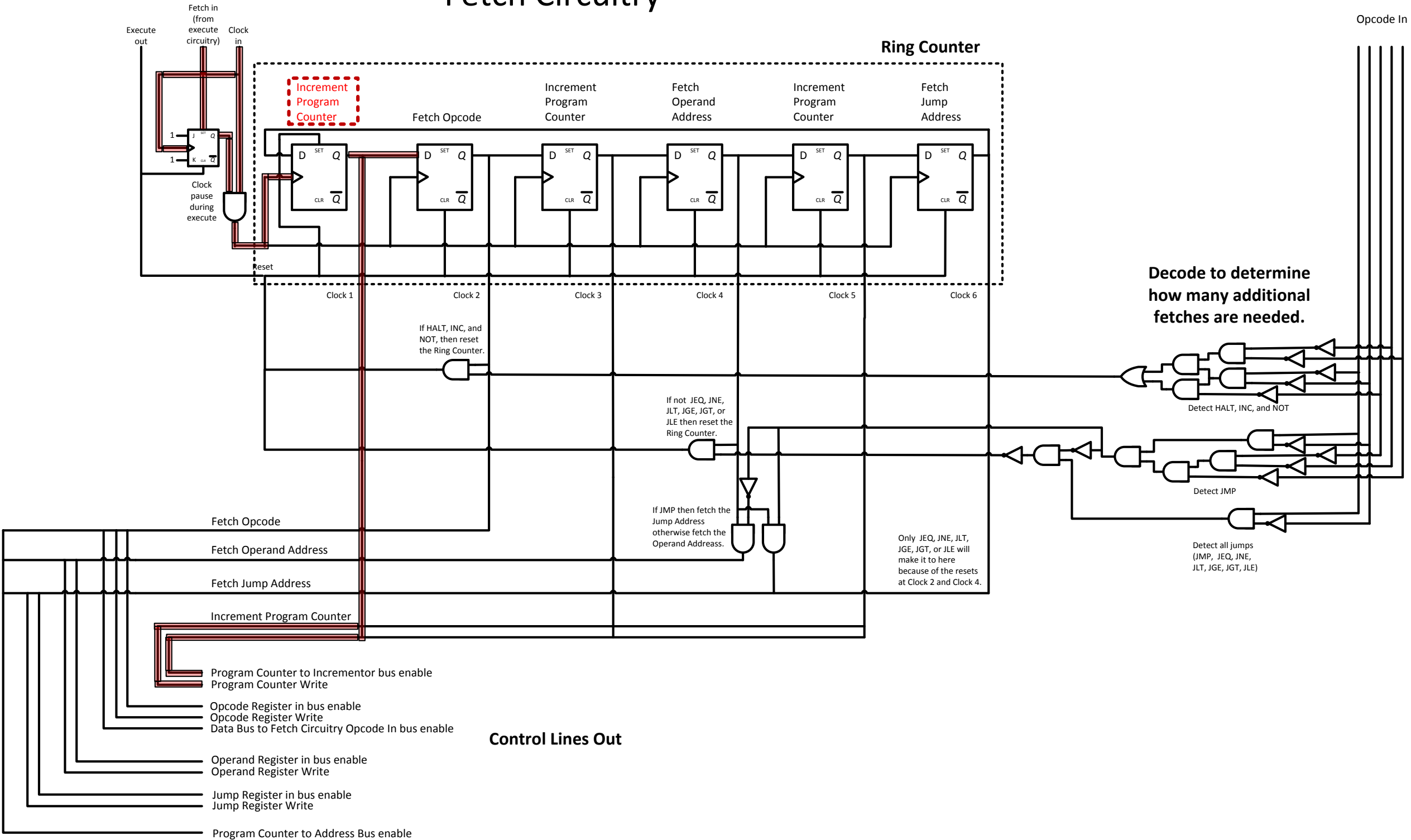
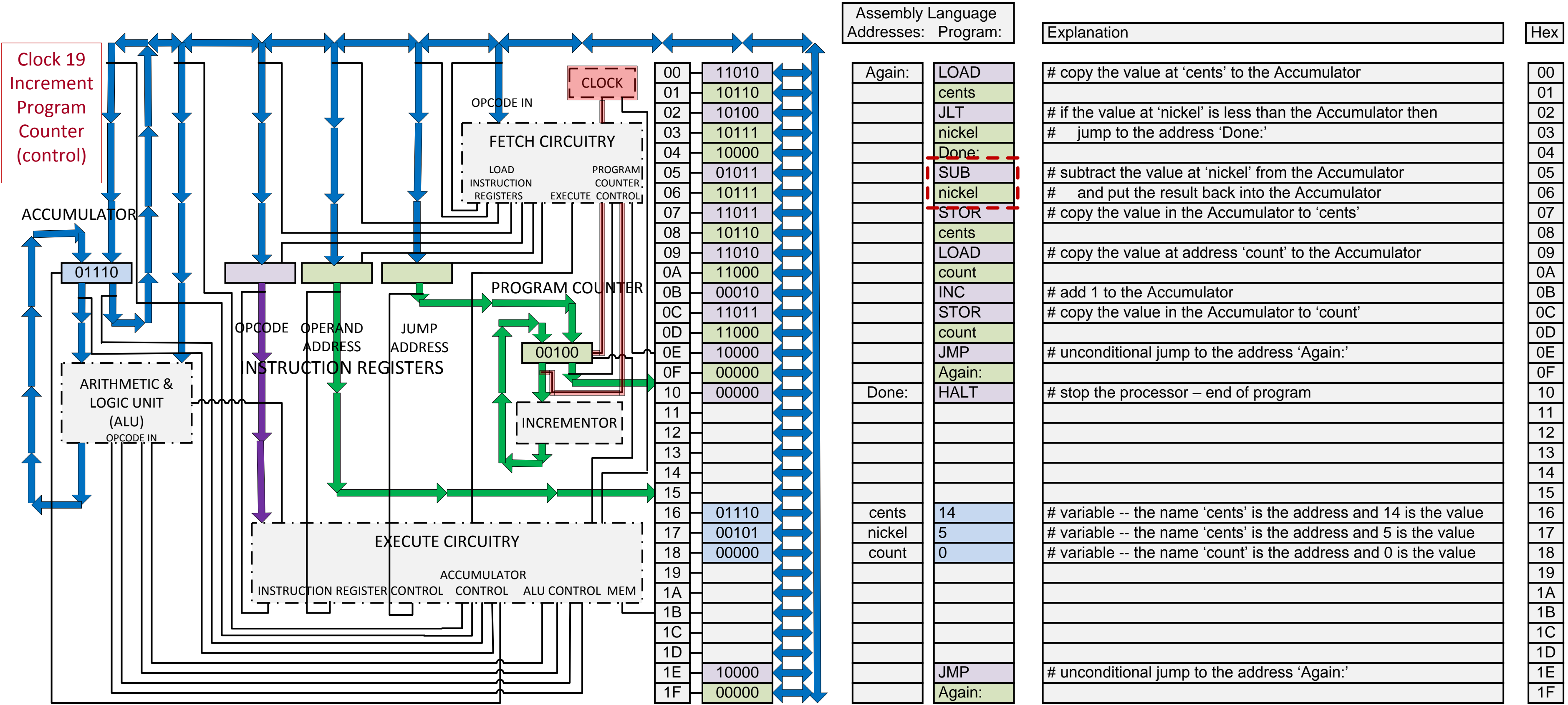




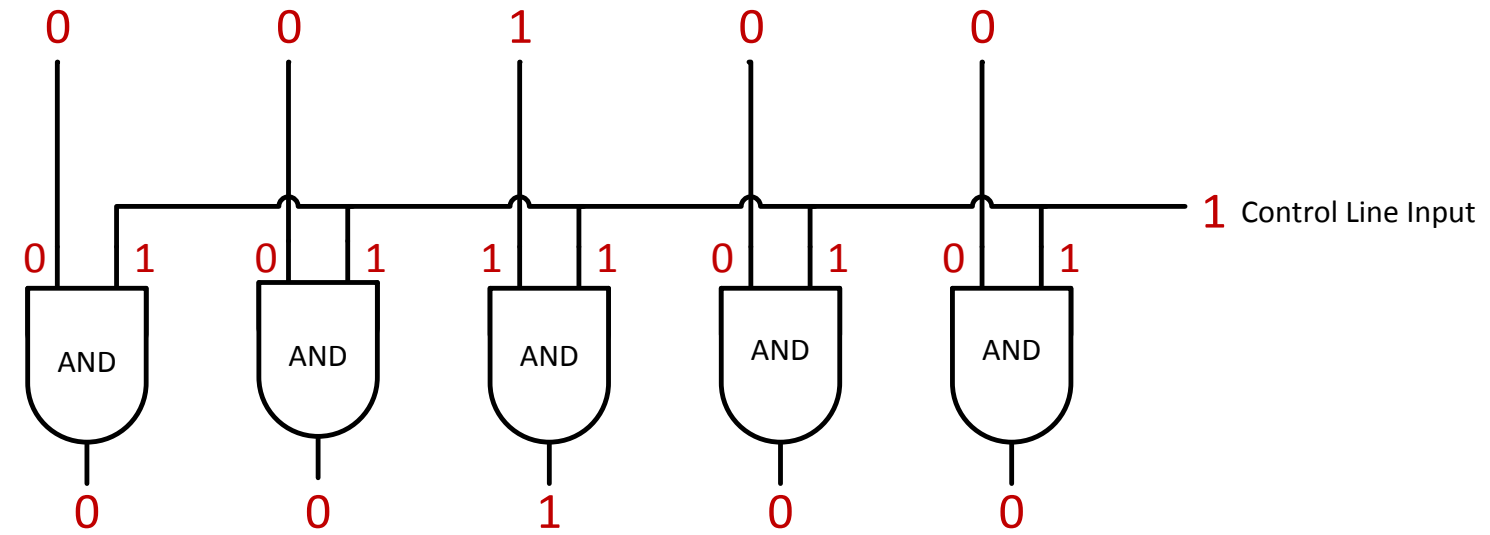
# Fetch Circuitry





# Bus Control

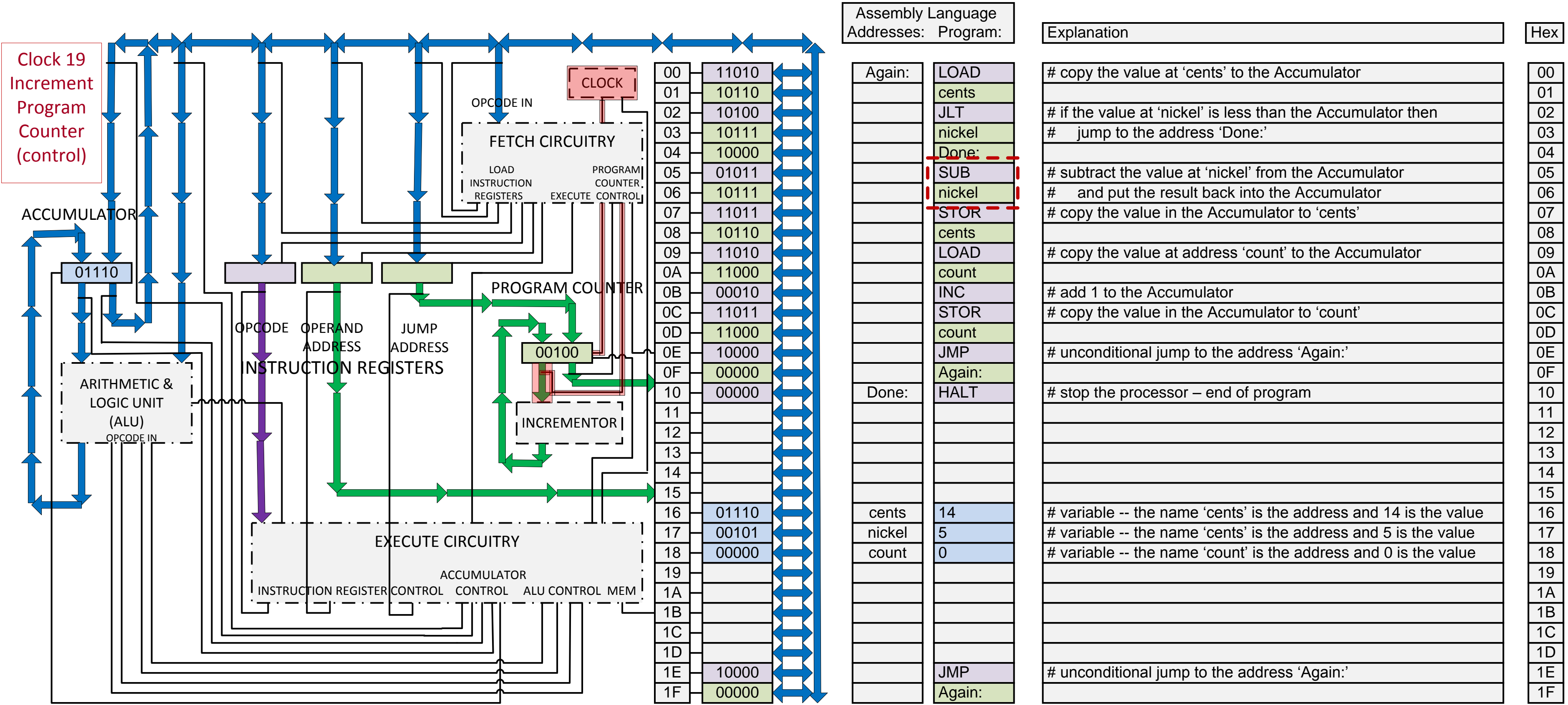
I n p u t s F r o m B u s



O u t p u t

All output is 0 if Control Line is 0

Identical to Inputs From Bus if Control Line is 1



# Incrementor

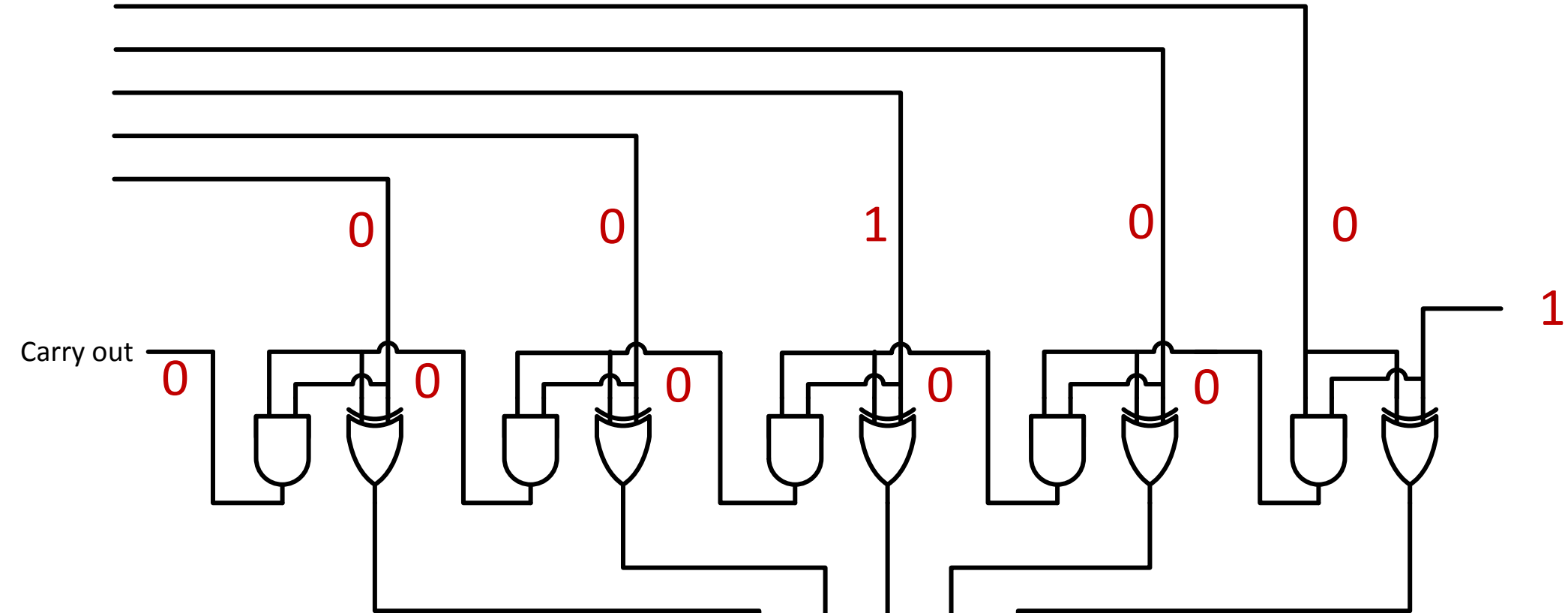
INC opcode

00100 + 00001

4 + 1

Left bus into ALU

0 0 1 0 0



Decimal equivalent

0 0 1 0 0  
+ 0 0 0 0 1  

---

0 0 1 0 1

4  
+ 1  

---

5

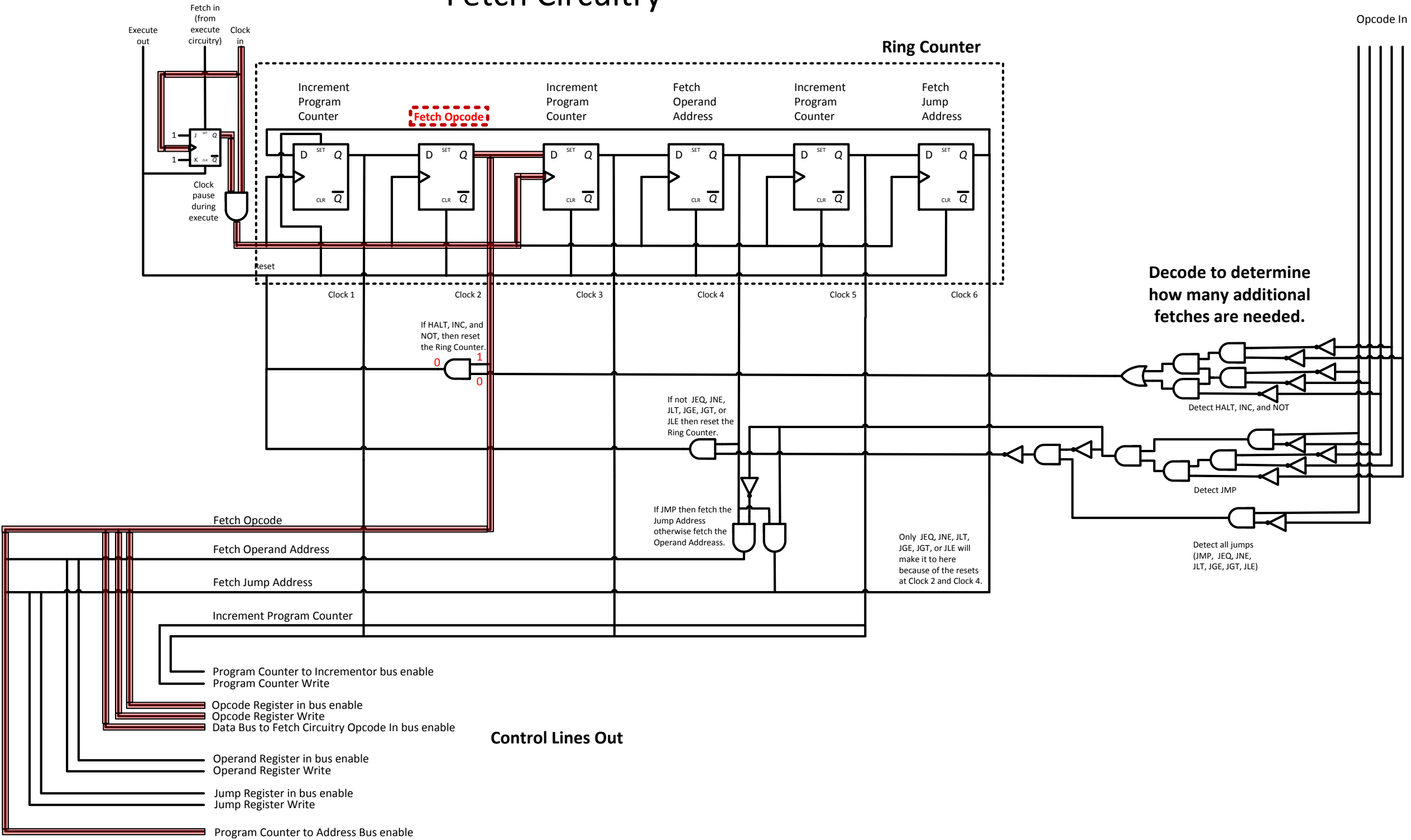
0 0 1 0 1  
Output  
Input number + 1



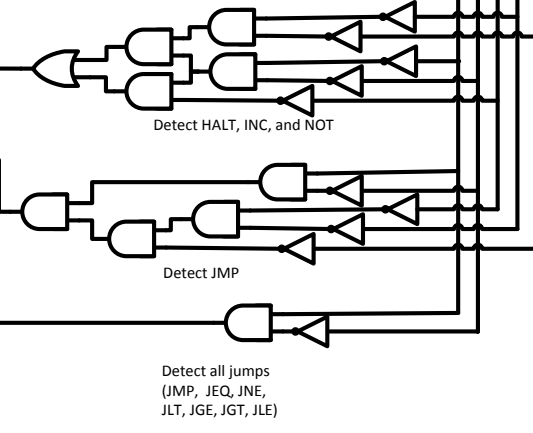




# Fetch Circuitry

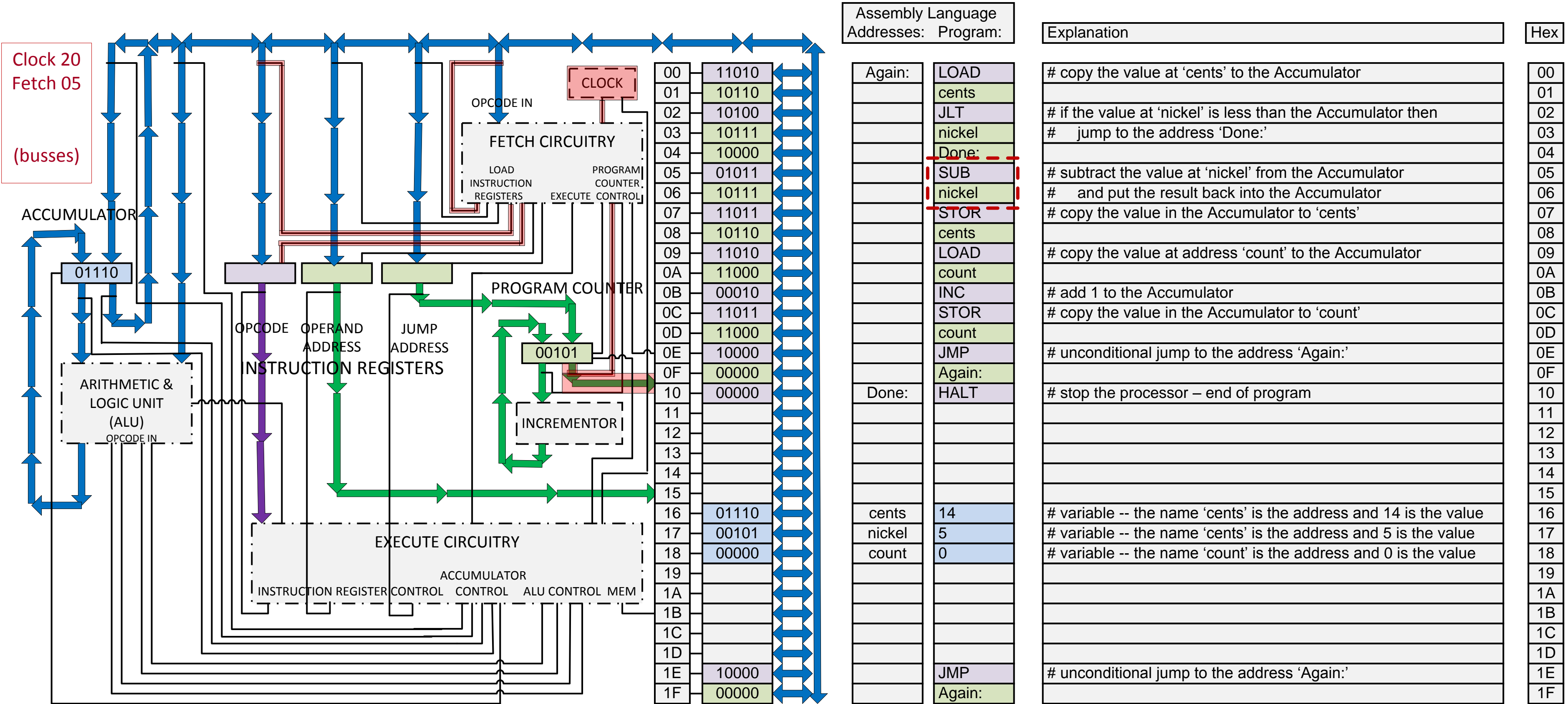


**Decode to determine how many additional fetches are needed.**



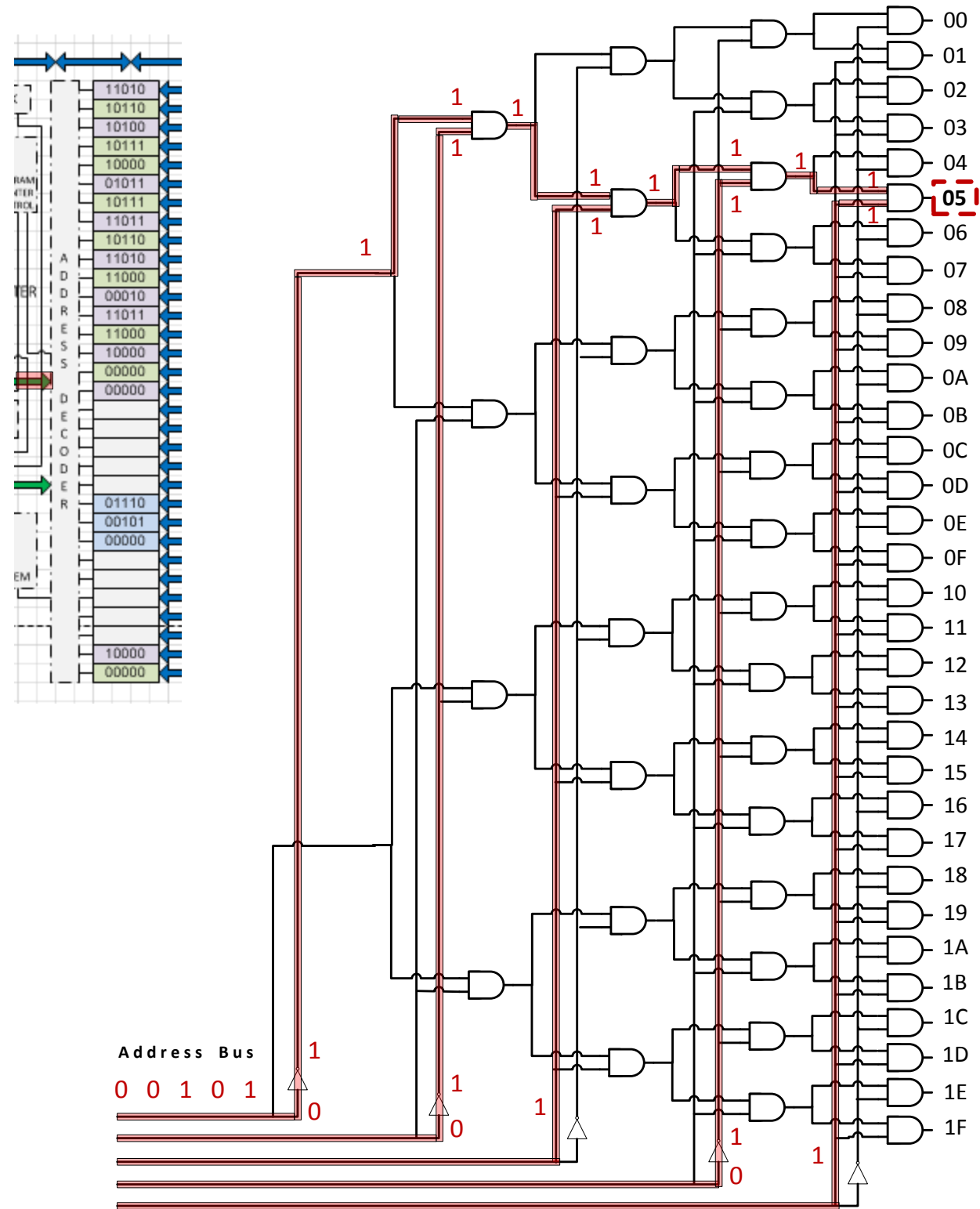
Opcode In

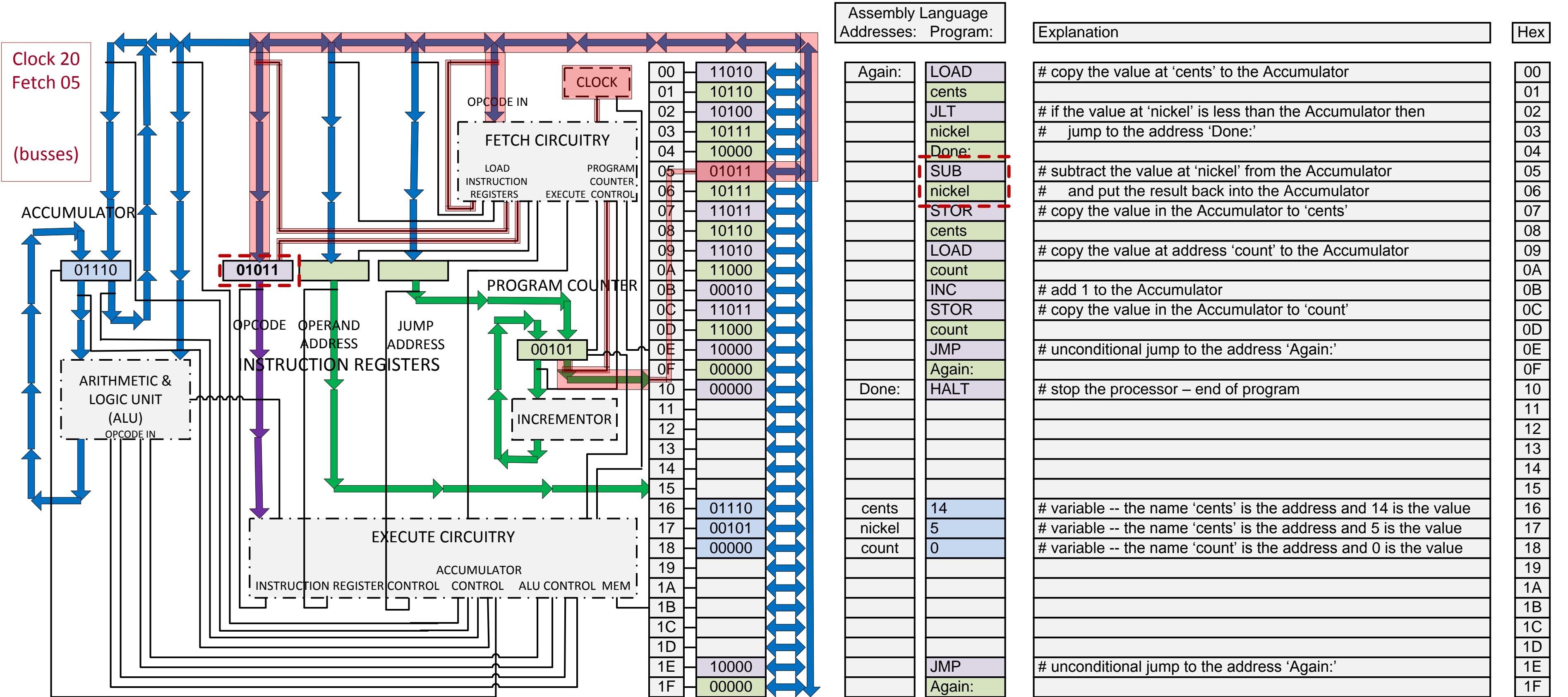


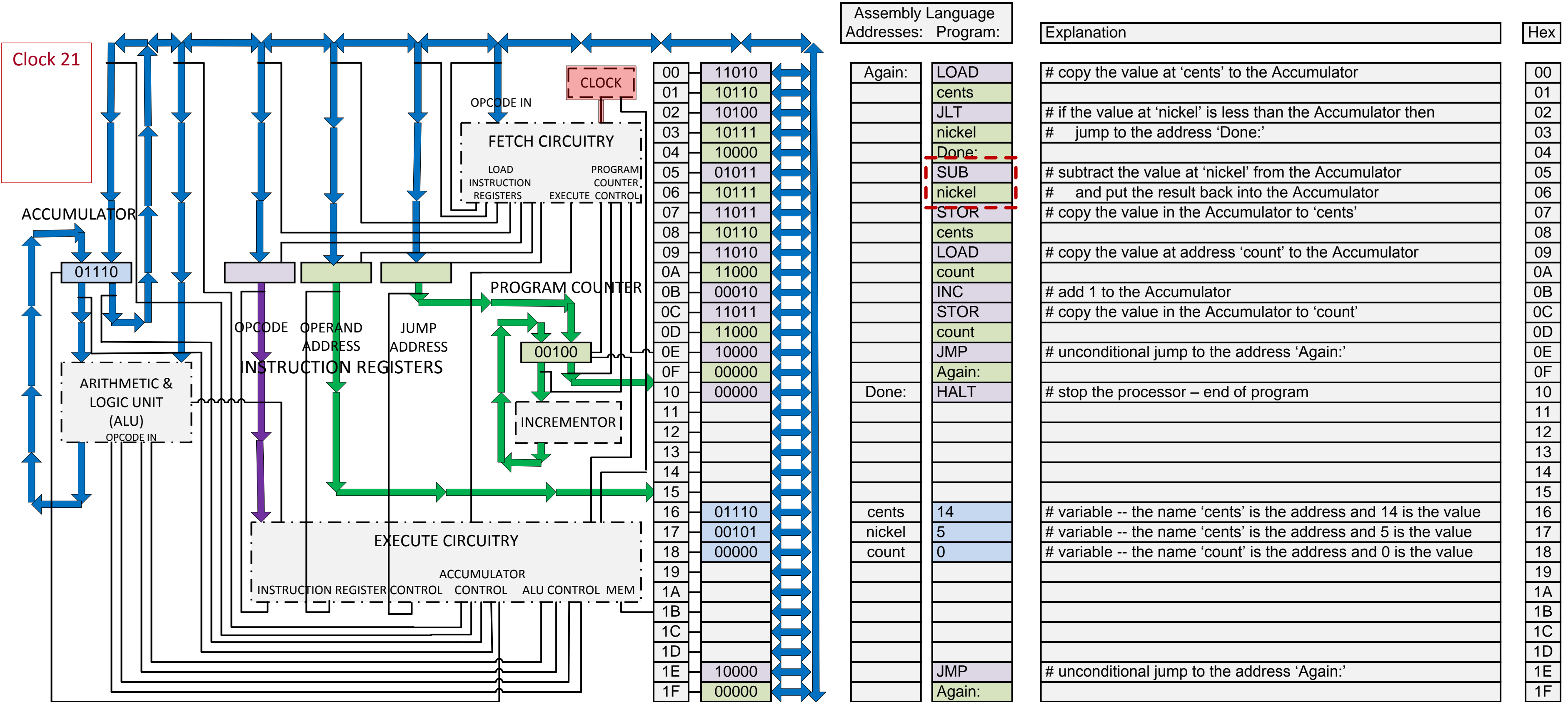


# Address Decoder

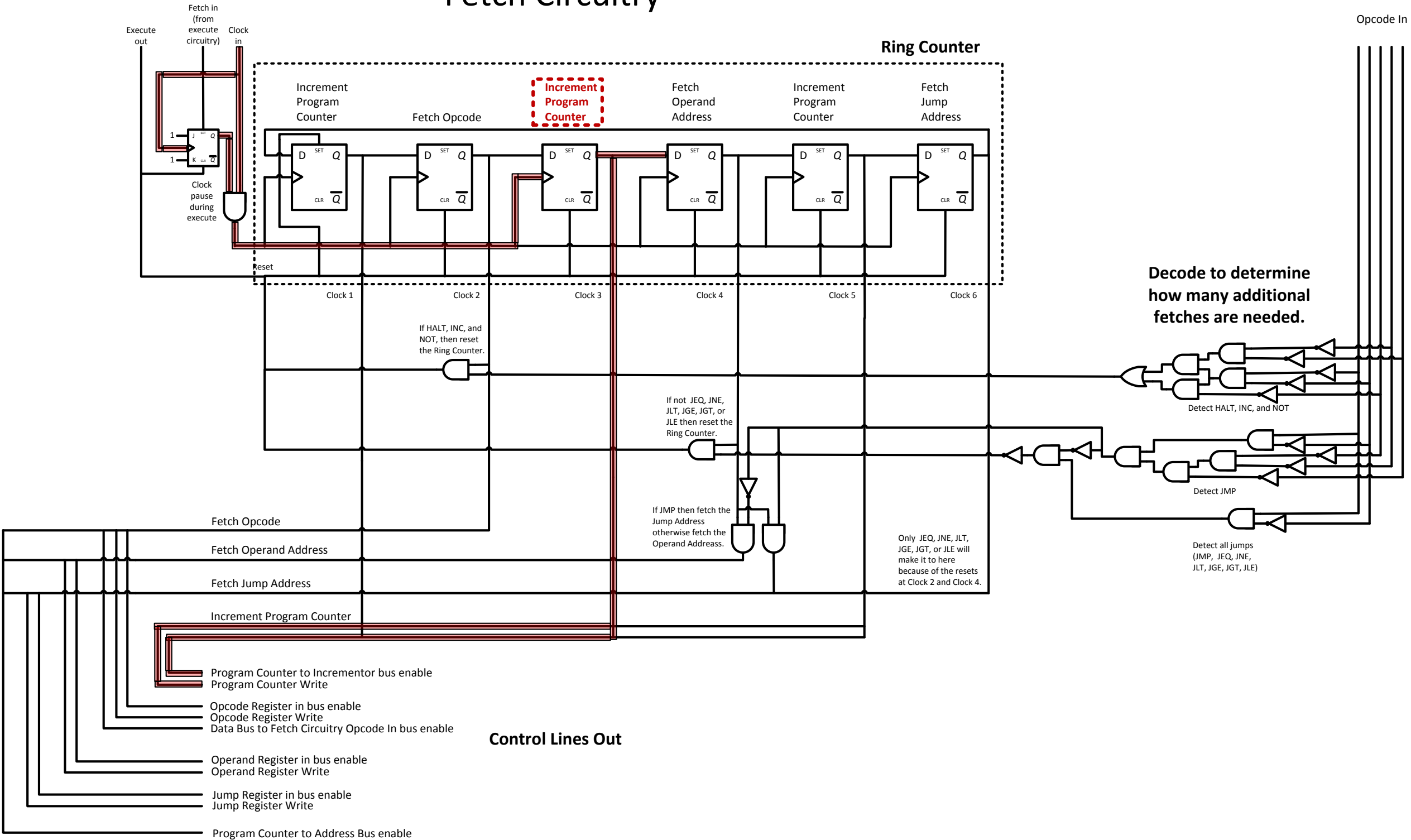
Input from Address Bus

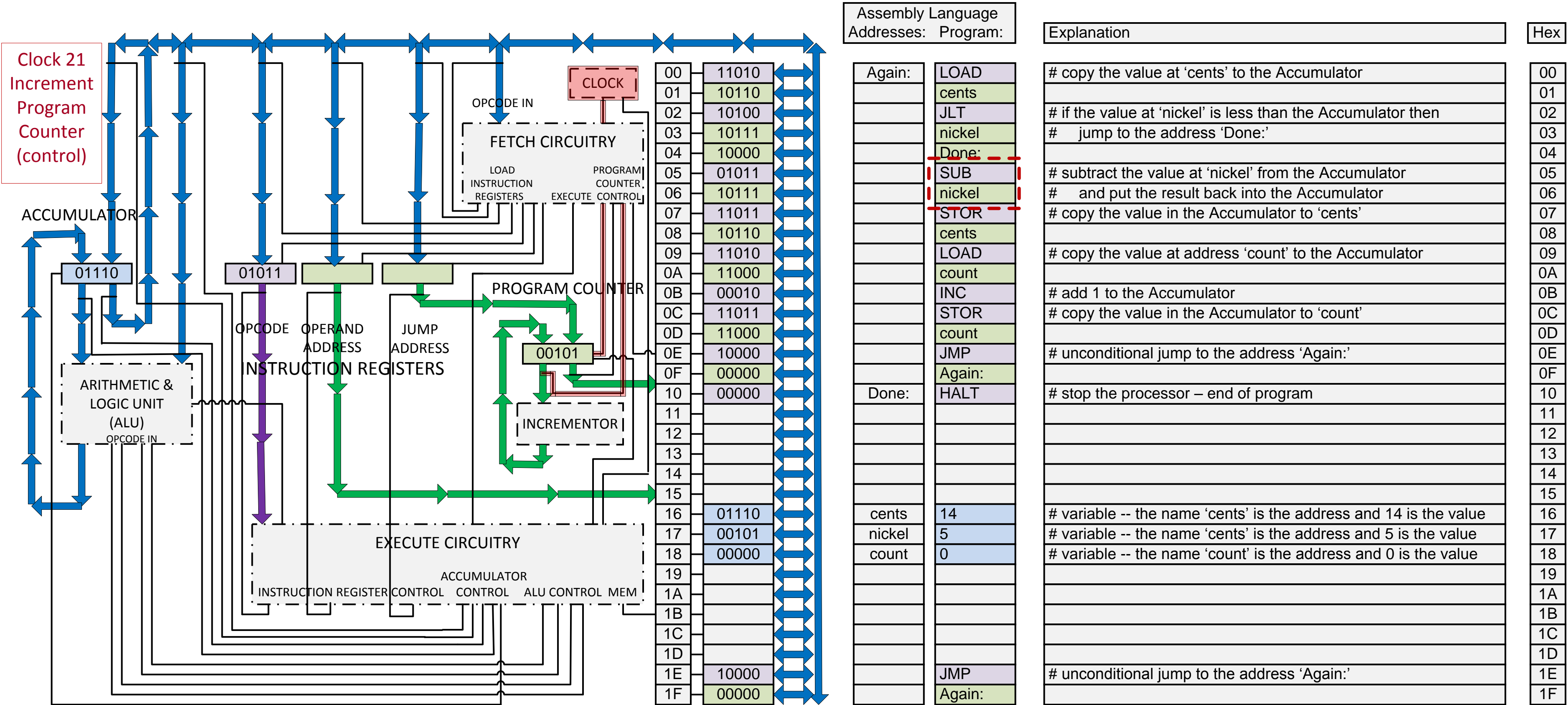






# Fetch Circuitry







# Incrementor

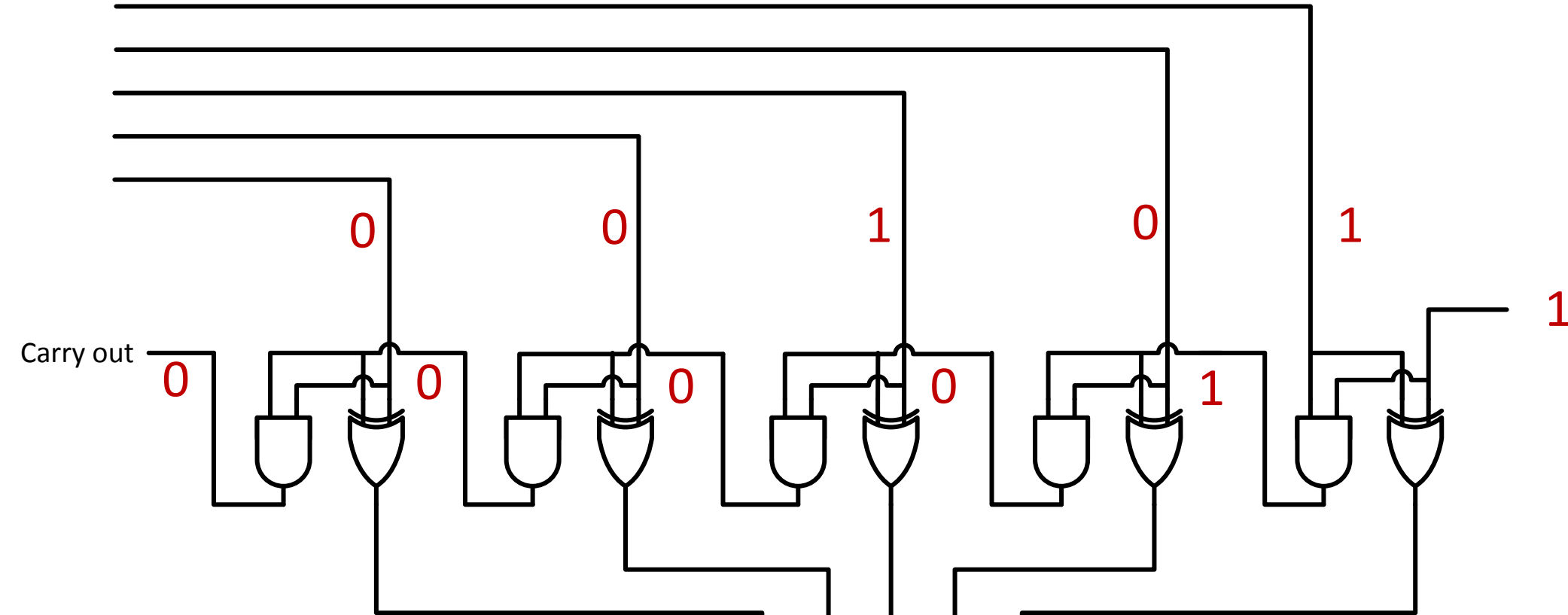
INC opcode

00101 + 00001

5 + 1

Left bus into ALU

0 0 1 0 1



Decimal equivalent

0 0 1 0 1  
+ 0 0 0 0 0  

---

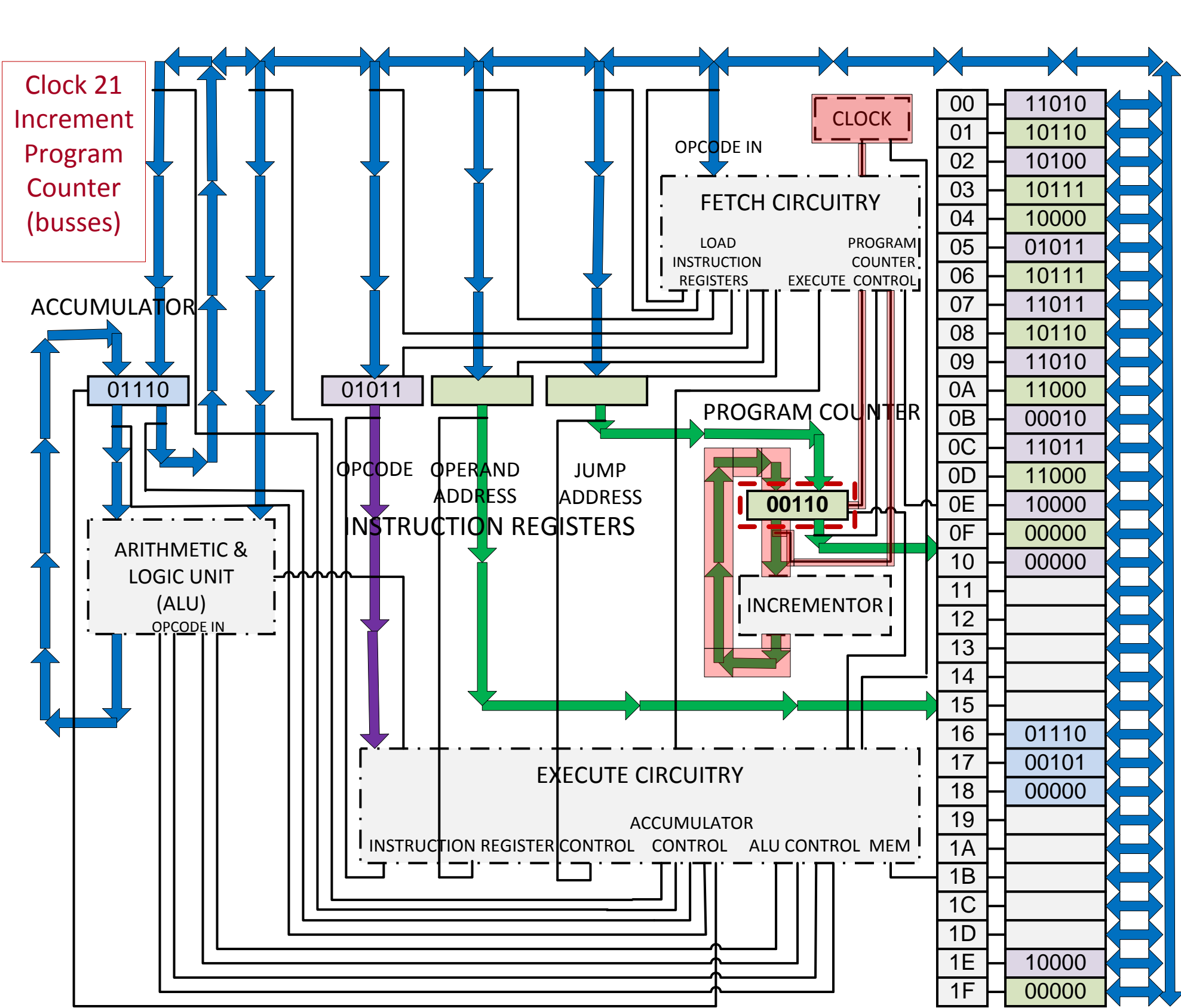
0 0 1 1 0

5  
+ 1  

---

6

0 0 1 1 0  
Output  
Input number + 1



Assembly Language Addresses: Program:

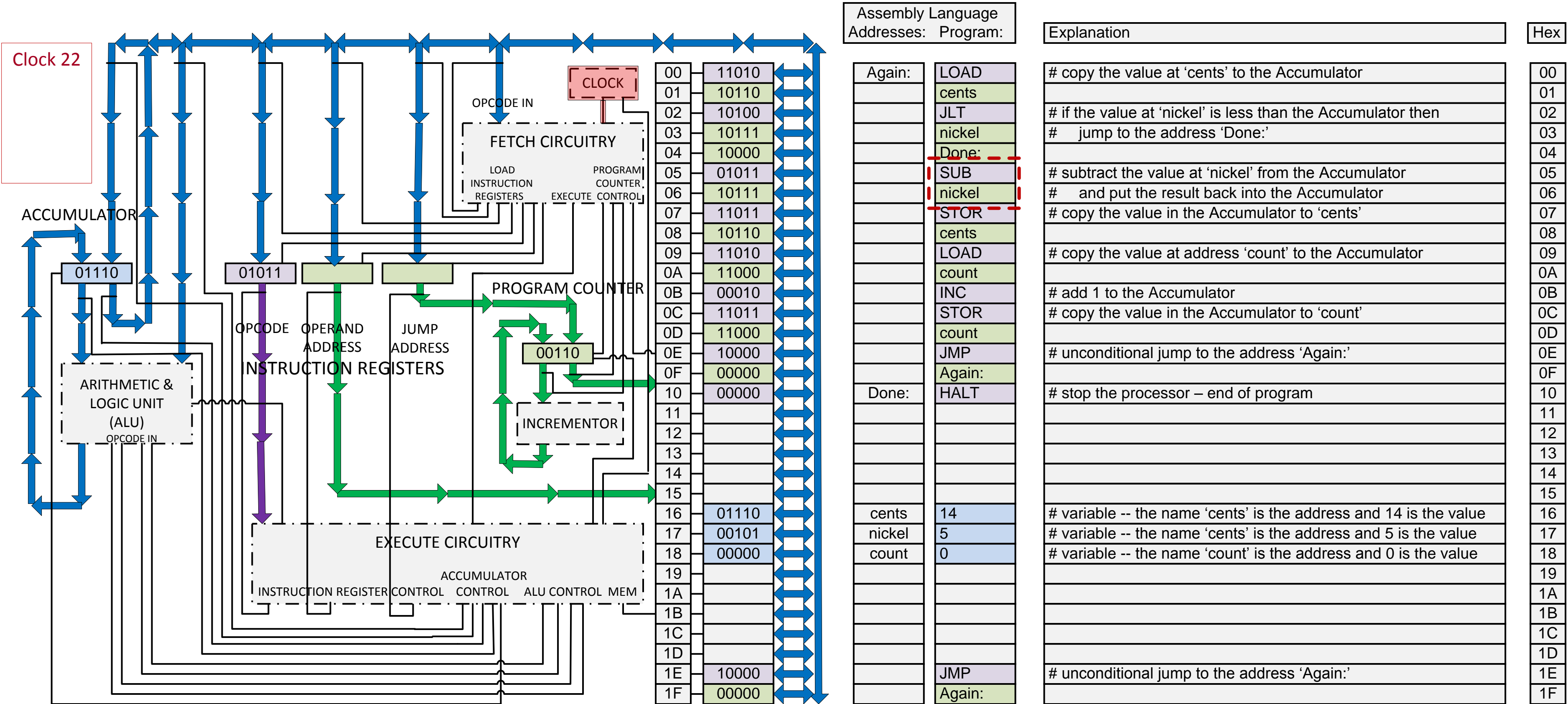
Address	Opcode	Operand
00	LOAD	
01	cents	
02	JLT	
03	nickel	
04	Done:	
05	SUB	
06	nickel	
07	STOR	
08	cents	
09	LOAD	
0A	count	
0B	INC	
0C	STOR	
0D	count	
0E	JMP	
0F	Again:	
10	HALT	
11		
12		
13		
14		
15		
16	cents	14
17	nickel	5
18	count	0
19		
1A		
1B		
1C		
1D		
1E	JMP	
1F	Again:	

Explanation

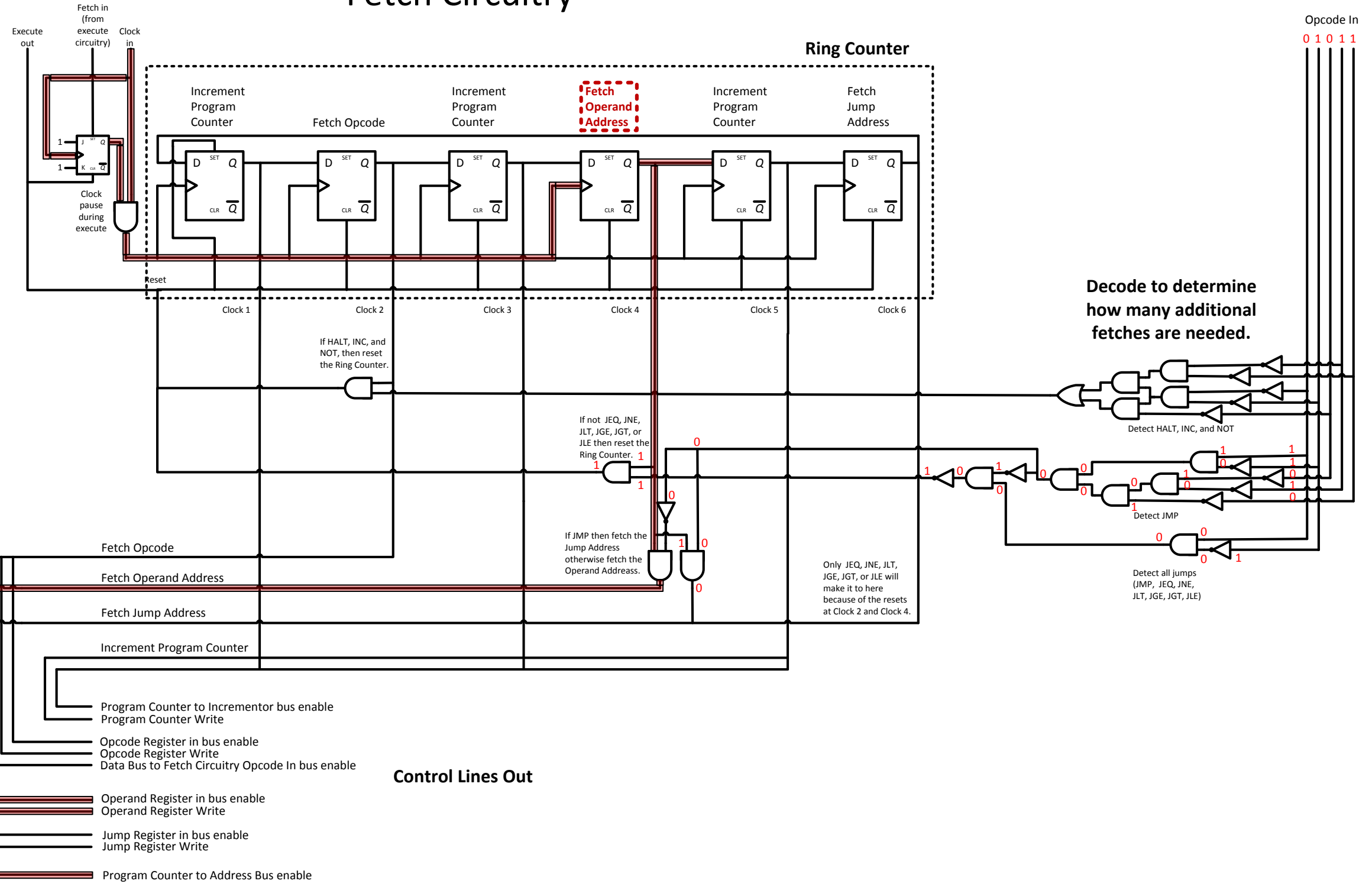
Address	Explanation
00	# copy the value at 'cents' to the Accumulator
01	
02	# if the value at 'nickel' is less than the Accumulator then
03	# jump to the address 'Done:'
04	
05	# subtract the value at 'nickel' from the Accumulator
06	# and put the result back into the Accumulator
07	# copy the value in the Accumulator to 'cents'
08	
09	# copy the value at address 'count' to the Accumulator
0A	
0B	# add 1 to the Accumulator
0C	# copy the value in the Accumulator to 'count'
0D	
0E	# unconditional jump to the address 'Again:'
0F	
10	# stop the processor – end of program
11	
12	
13	
14	
15	
16	# variable -- the name 'cents' is the address and 14 is the value
17	# variable -- the name 'cents' is the address and 5 is the value
18	# variable -- the name 'count' is the address and 0 is the value
19	
1A	
1B	
1C	
1D	
1E	# unconditional jump to the address 'Again:'
1F	

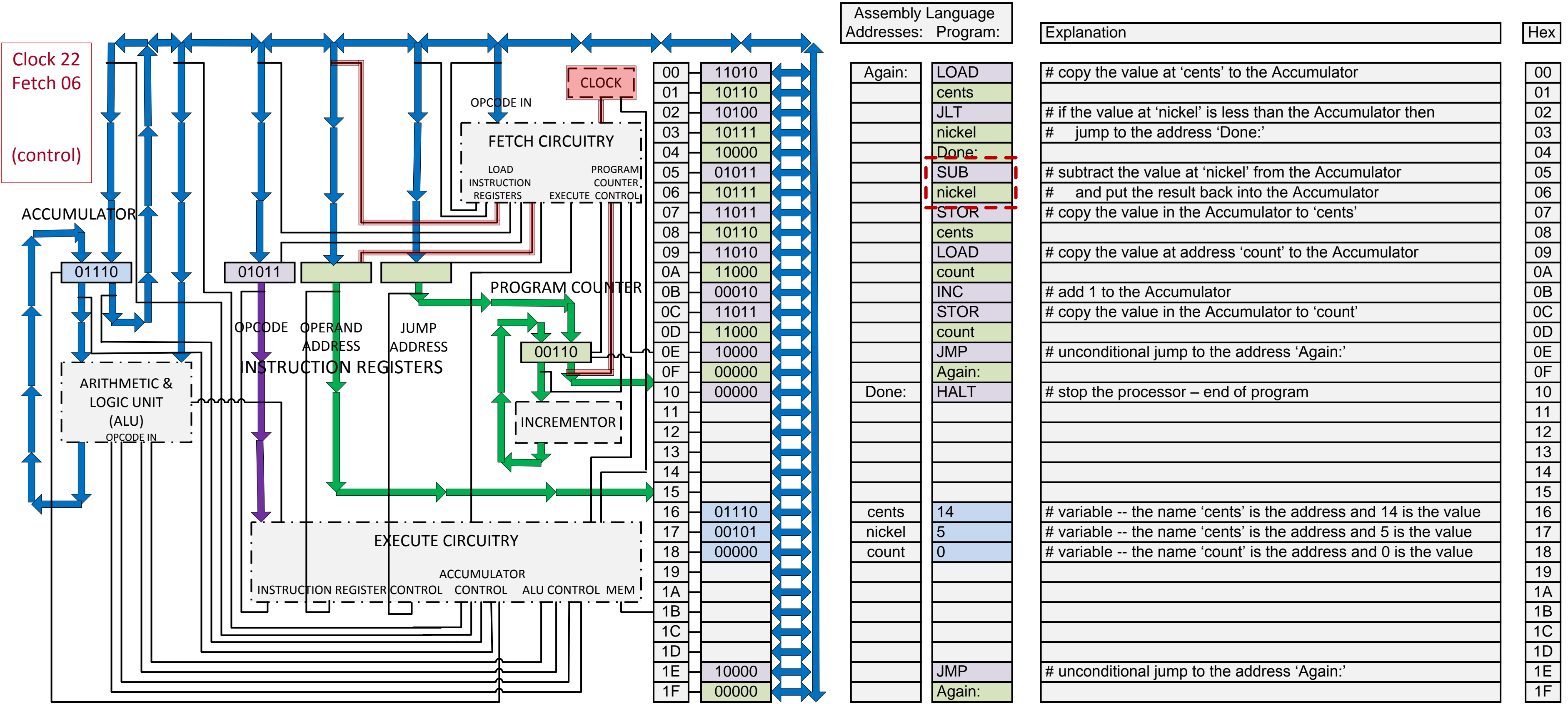
Hex

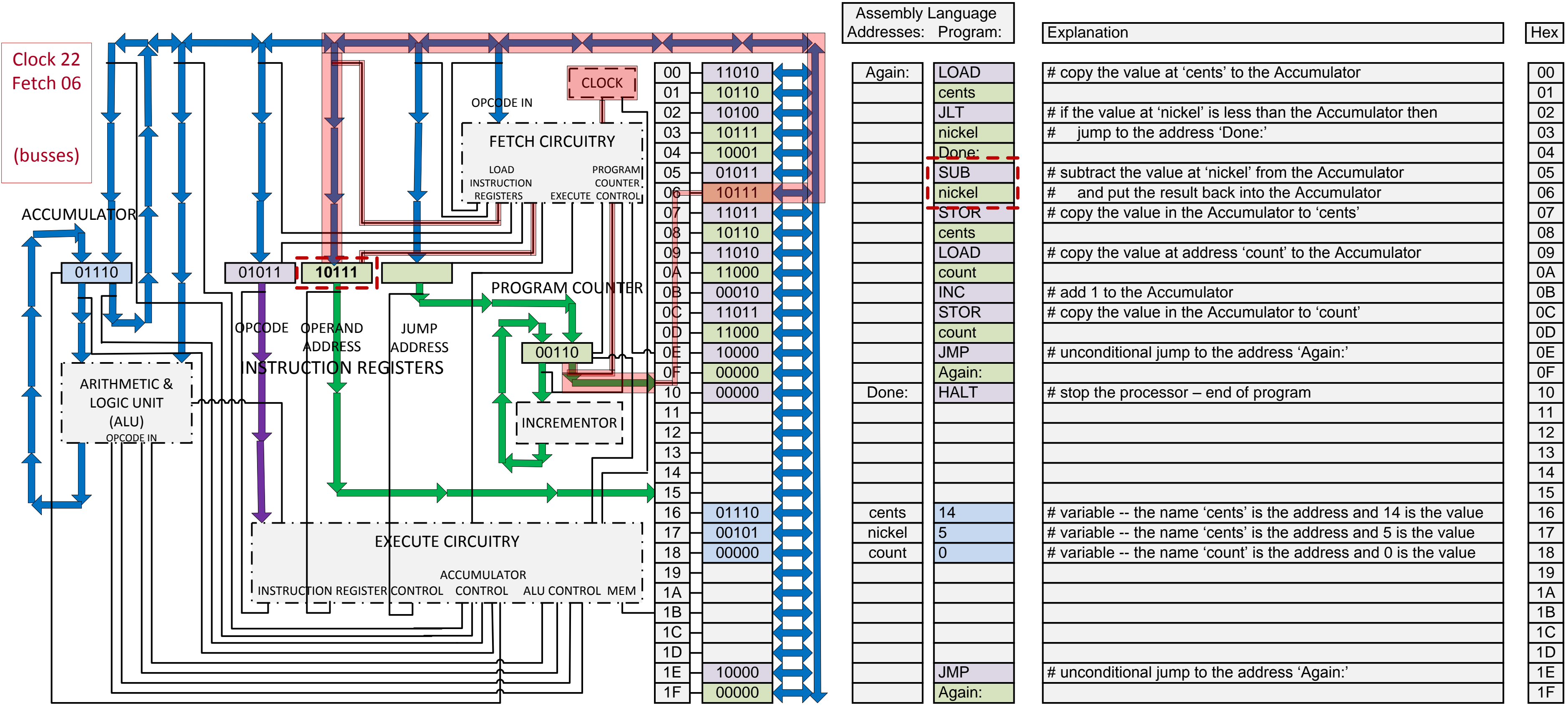
Address	Hex
00	00
01	01
02	02
03	03
04	04
05	05
06	06
07	07
08	08
09	09
0A	0A
0B	0B
0C	0C
0D	0D
0E	0E
0F	0F
10	10
11	11
12	12
13	13
14	14
15	15
16	16
17	17
18	18
19	19
1A	1A
1B	1B
1C	1C
1D	1D
1E	1E
1F	1F



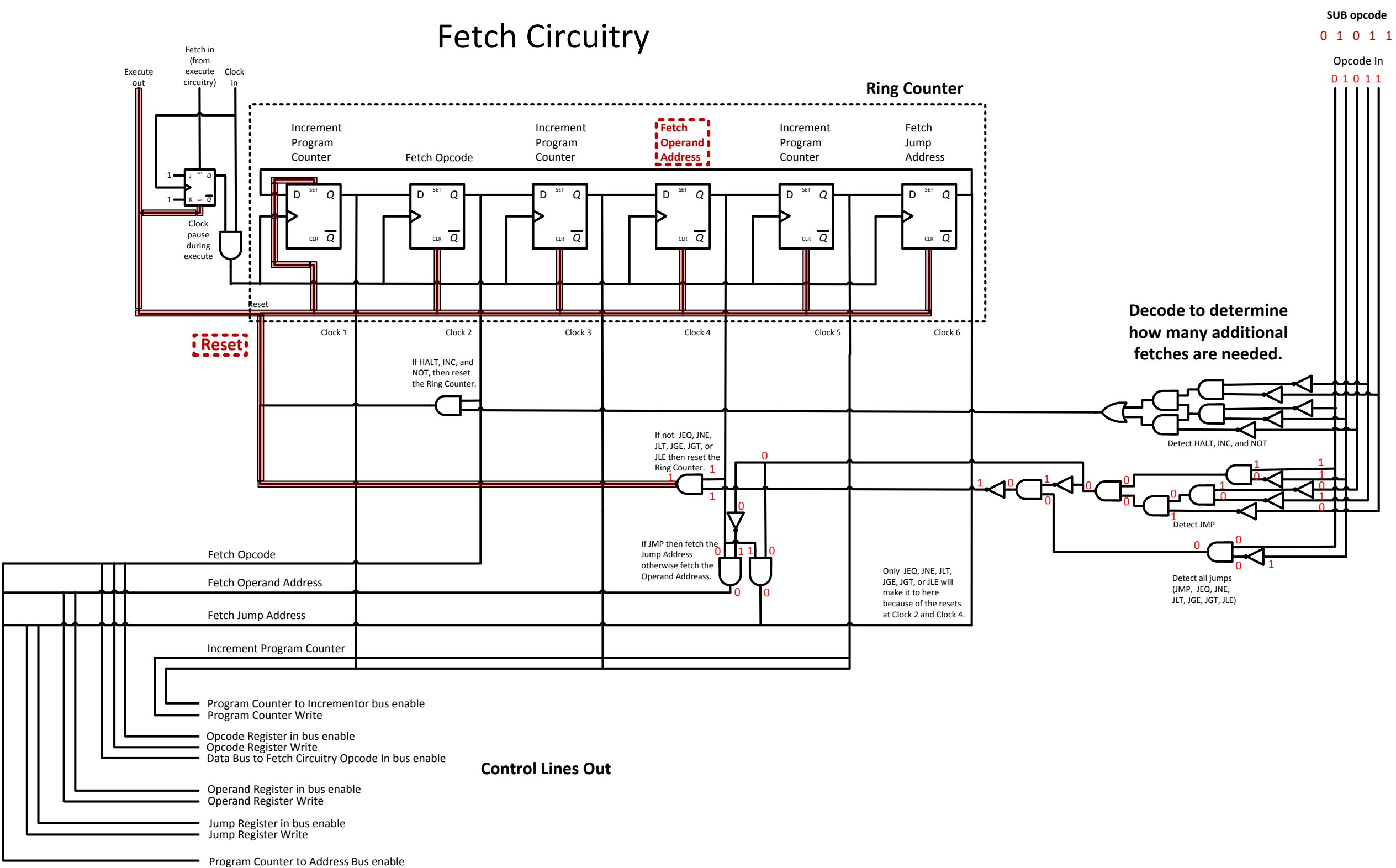
# Fetch Circuitry

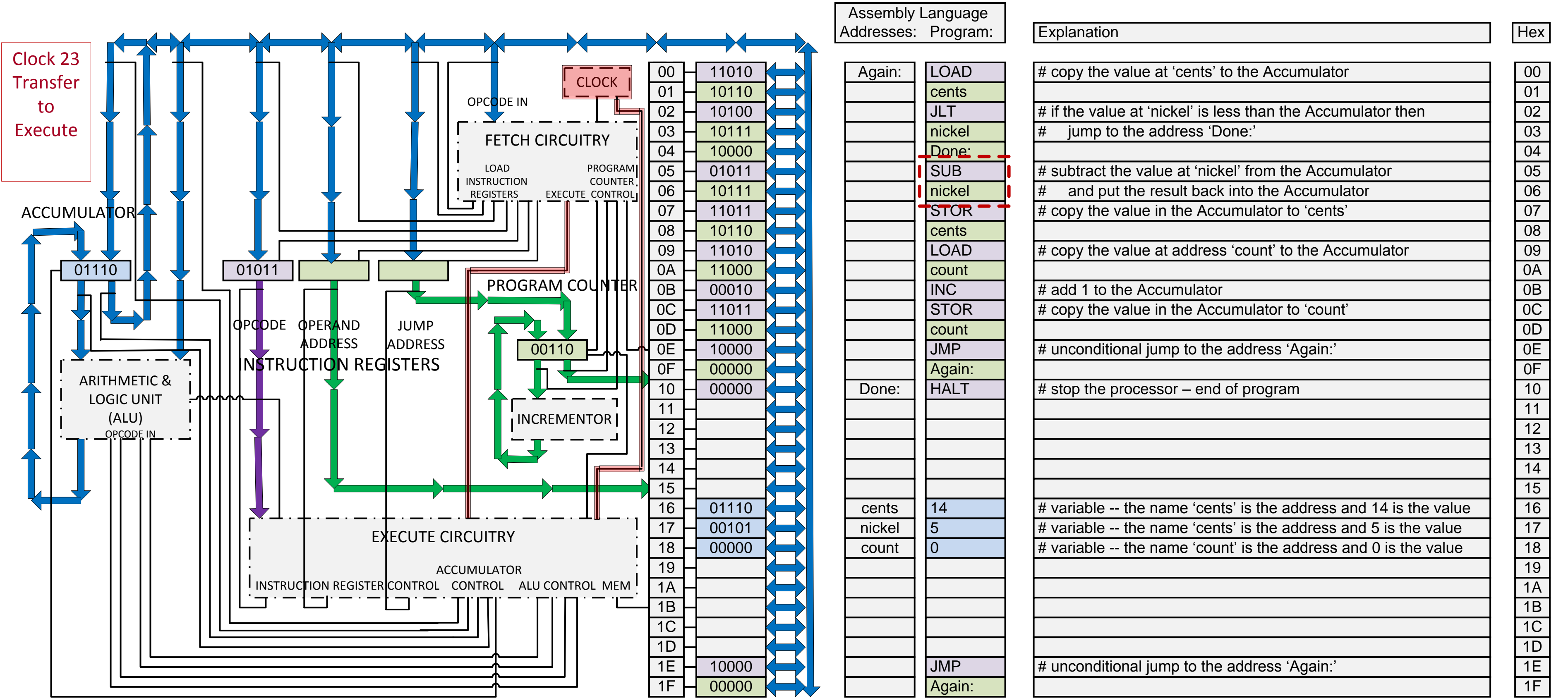






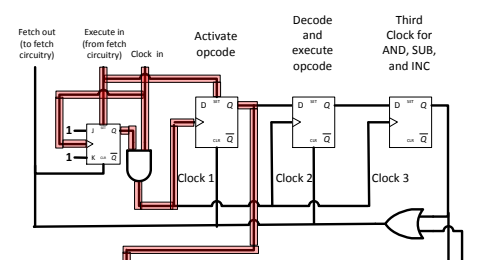
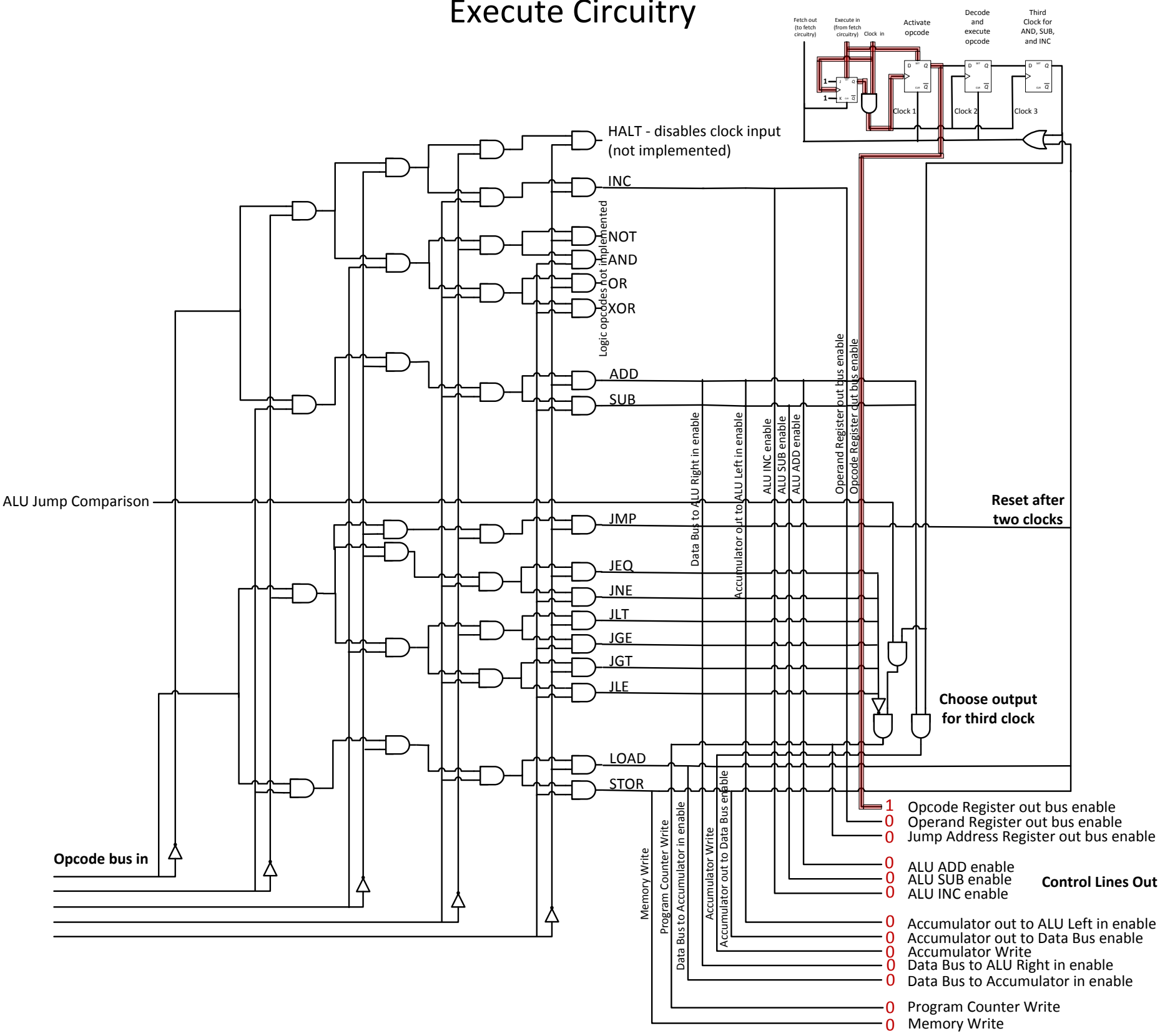
# Fetch Circuitry







# Execute Circuitry



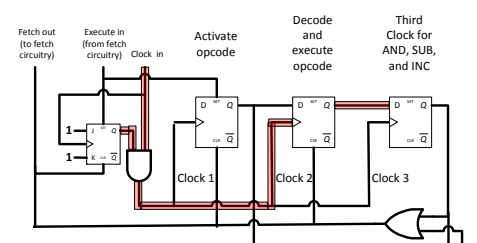
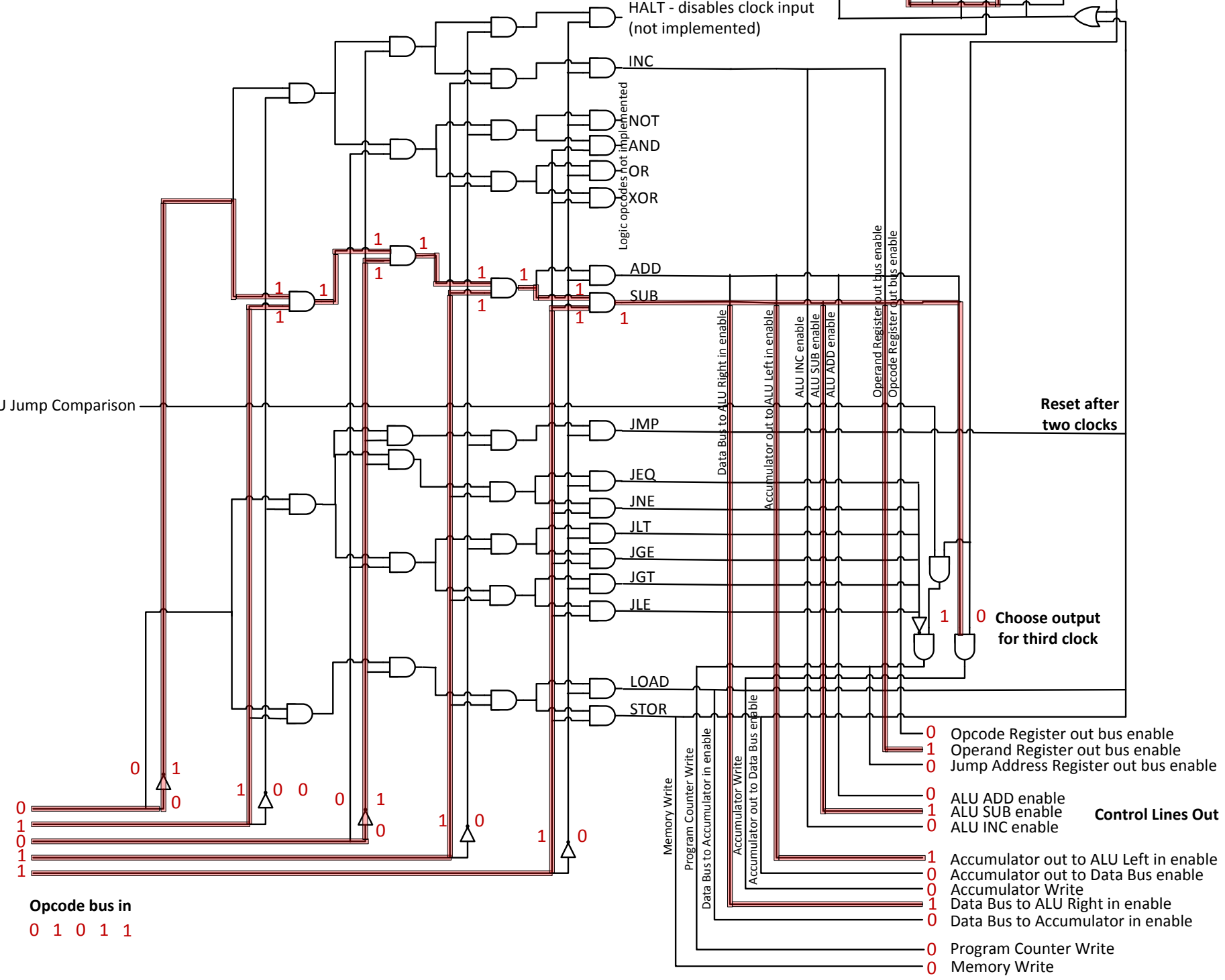
- Control Lines Out**
- 1 Opcode Register out bus enable
  - 0 Operand Register out bus enable
  - 0 Jump Address Register out bus enable
  - 0 ALU ADD enable
  - 0 ALU SUB enable
  - 0 ALU INC enable
  - 0 Accumulator out to ALU Left in enable
  - 0 Accumulator out to Data Bus enable
  - 0 Accumulator Write
  - 0 Data Bus to ALU Right in enable
  - 0 Data Bus to Accumulator in enable
  - 0 Program Counter Write
  - 0 Memory Write





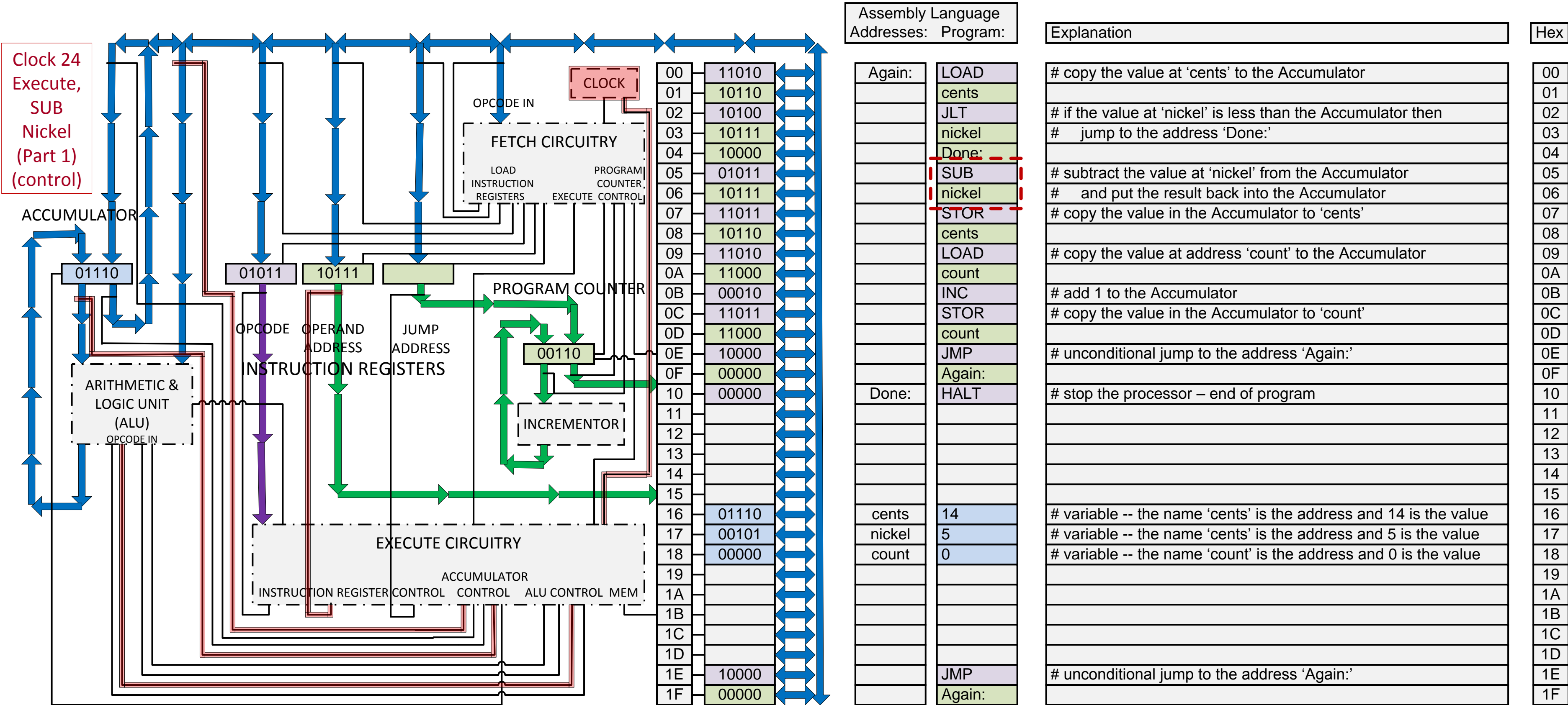
# Execute Circuitry

**SUB opcode**  
0 1 0 1 1



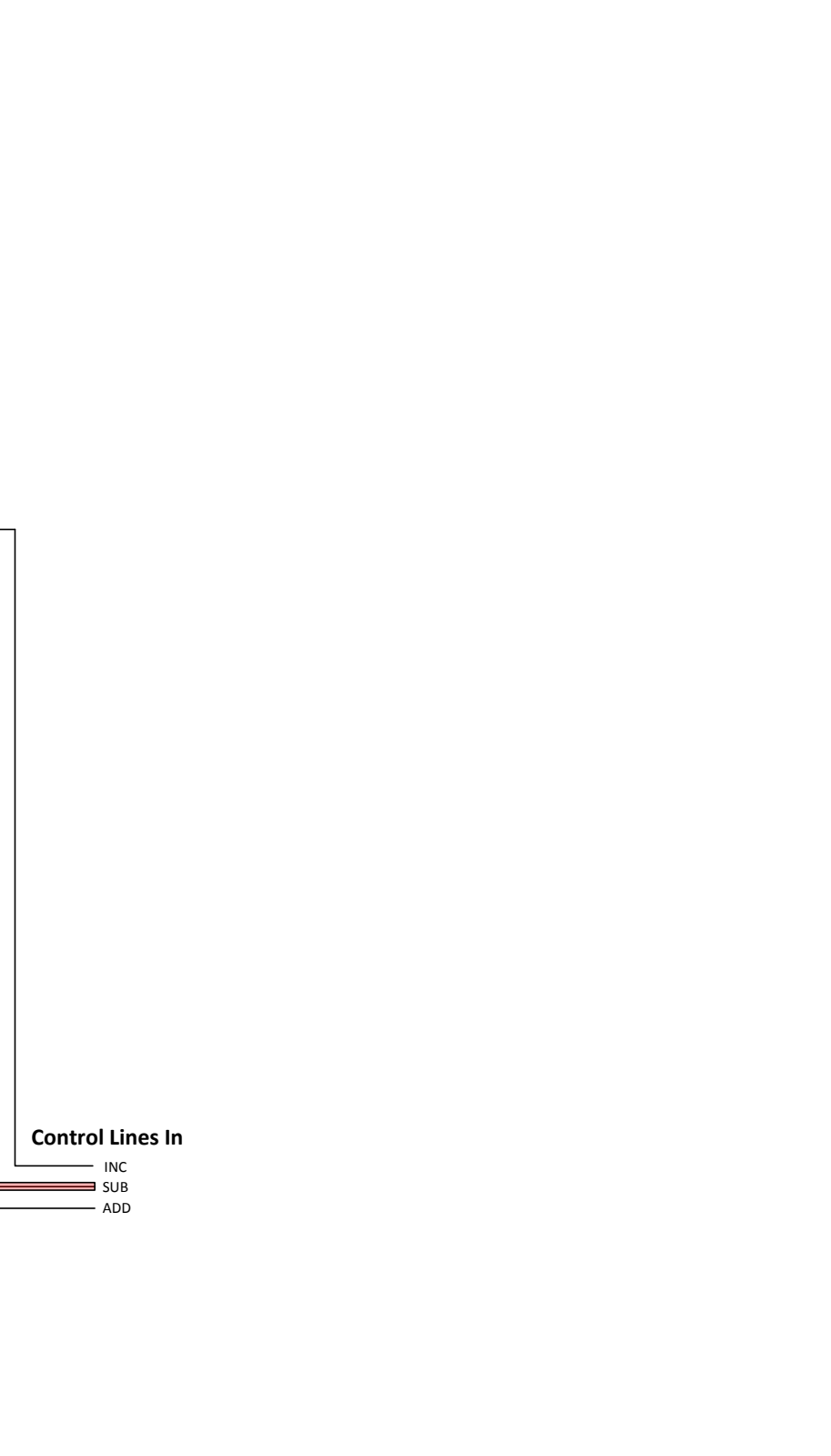
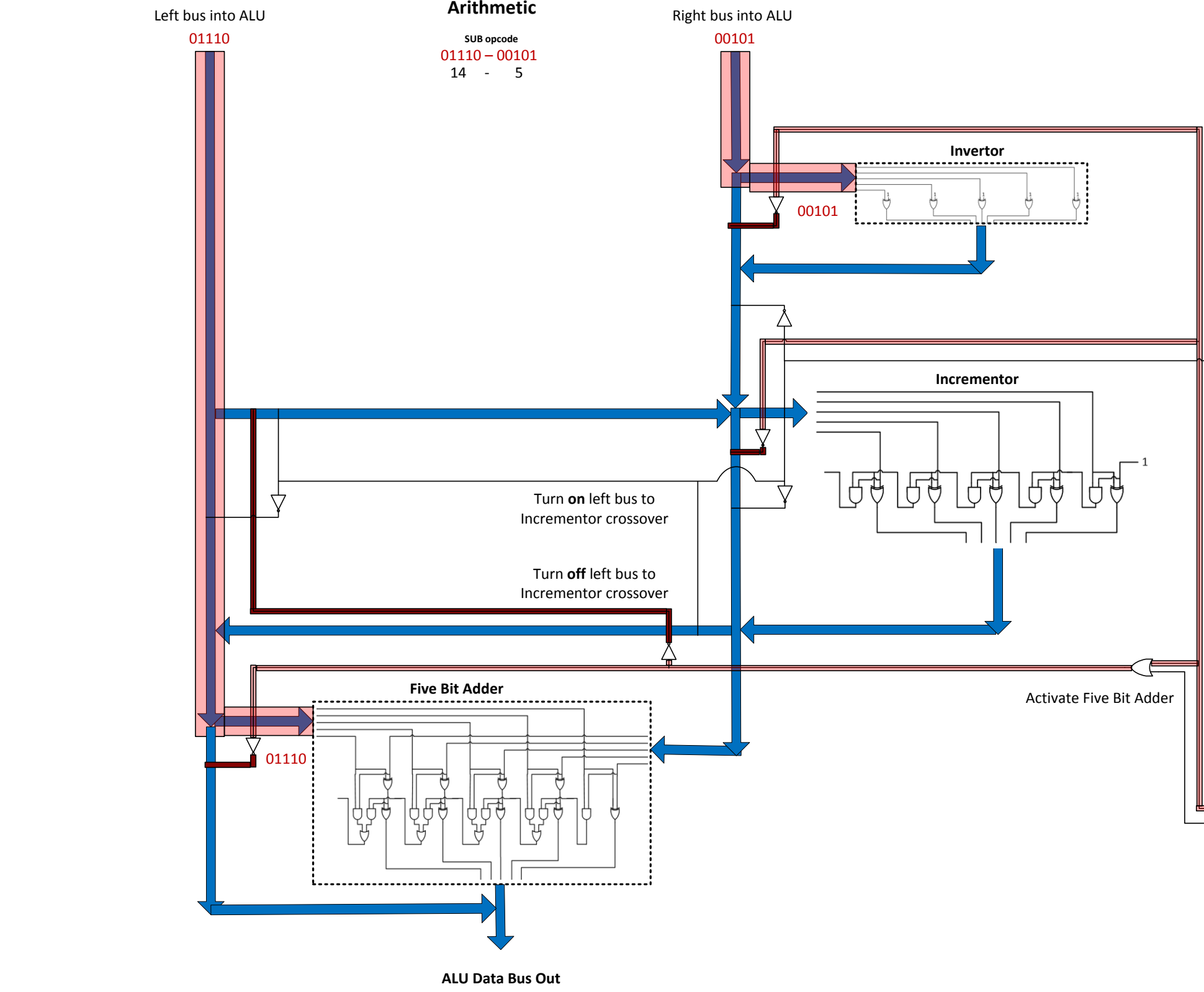
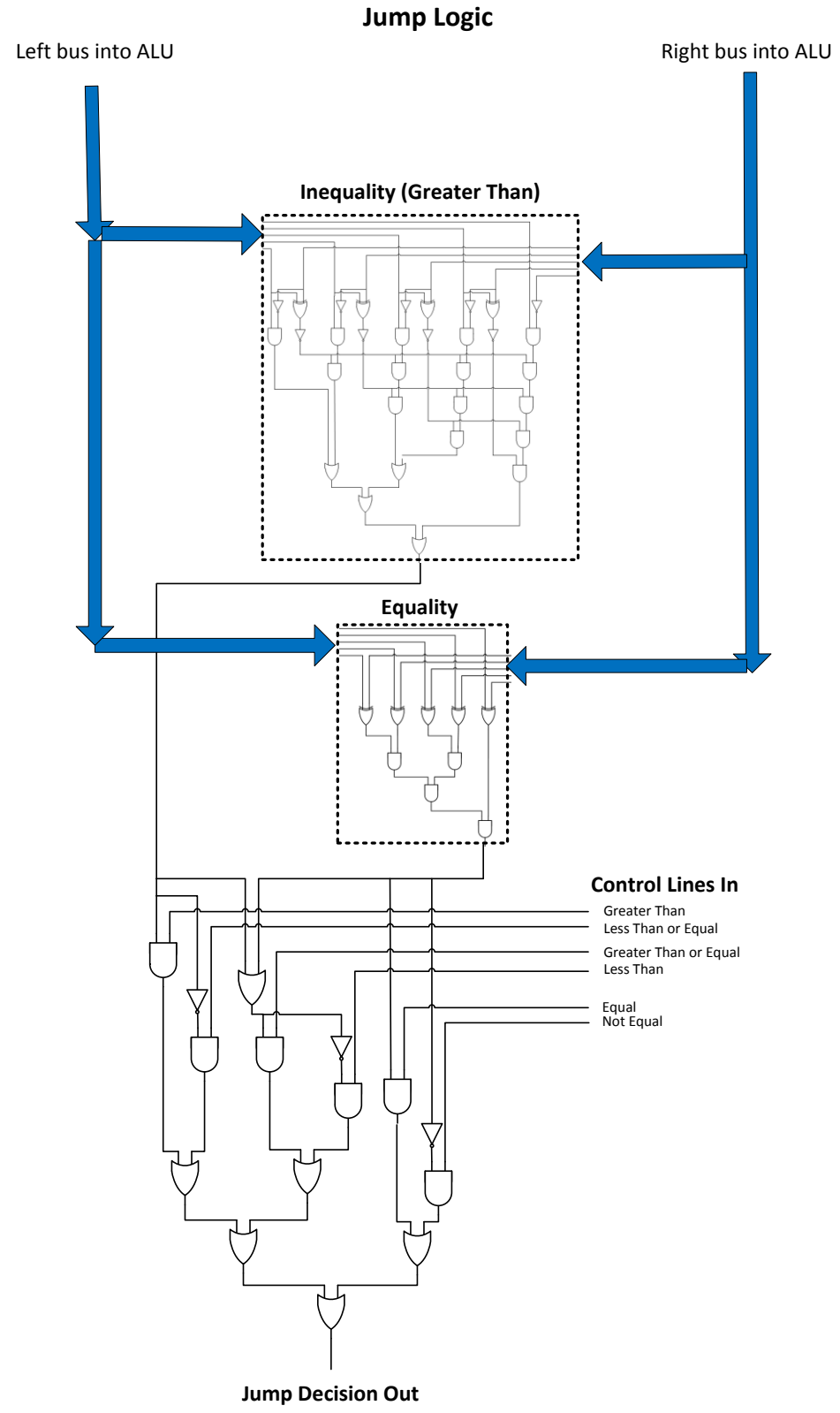
**Opcode bus in**  
0 1 0 1 1

**Control Lines Out**

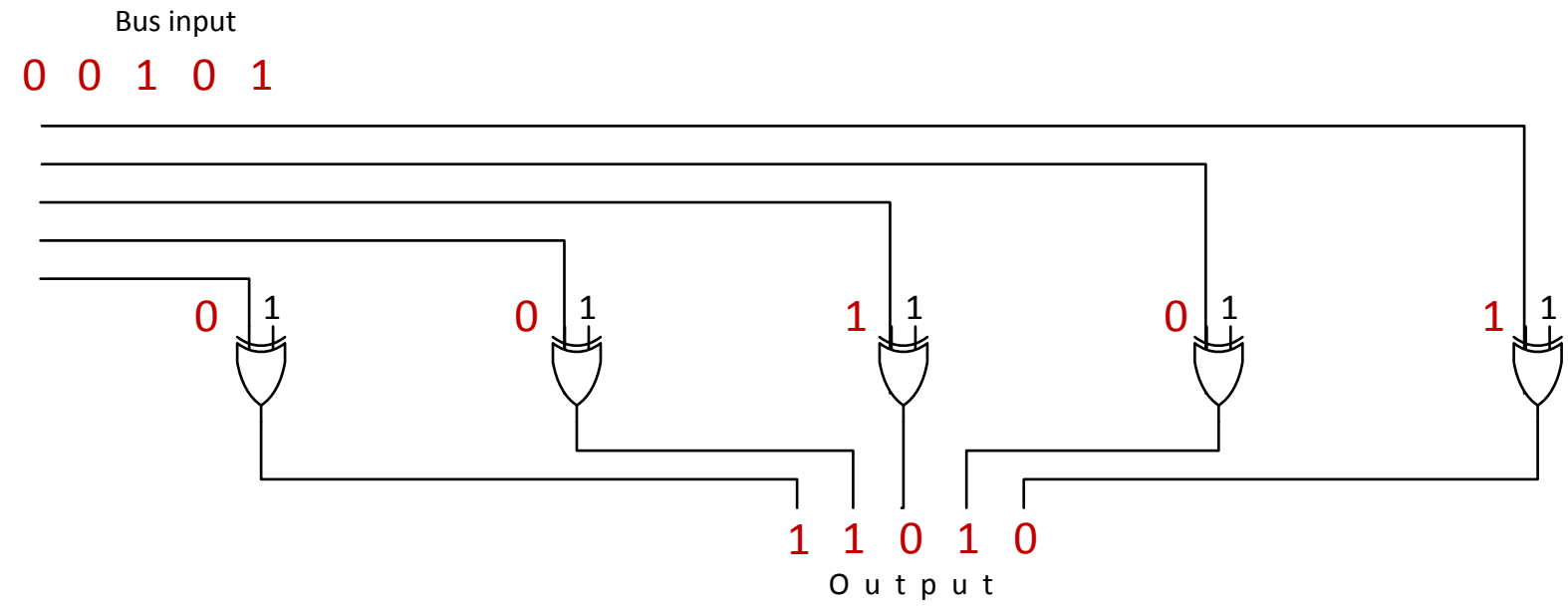




# ALU (Arithmetic and Logic Unit)

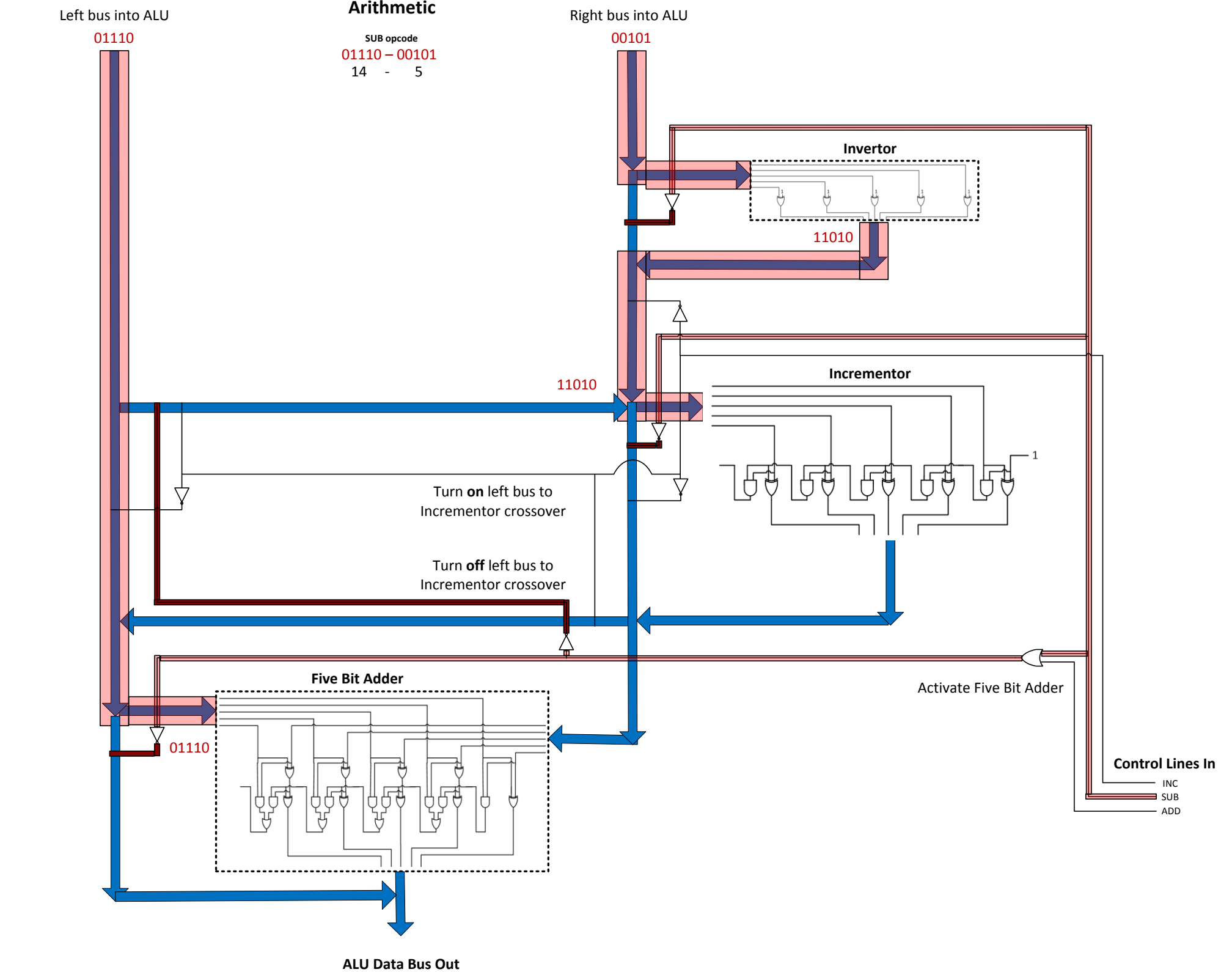
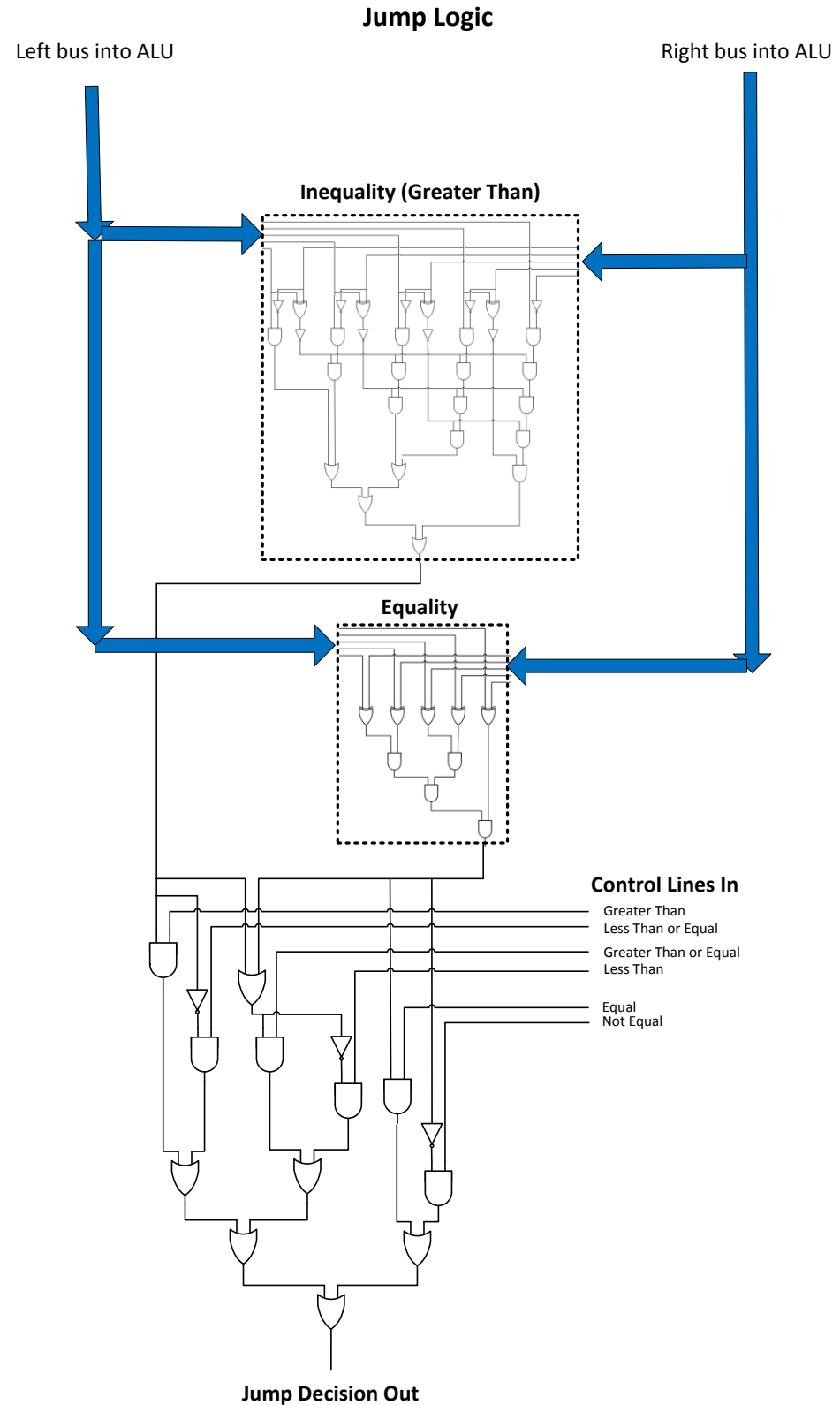


# Invertor





# ALU (Arithmetic and Logic Unit)



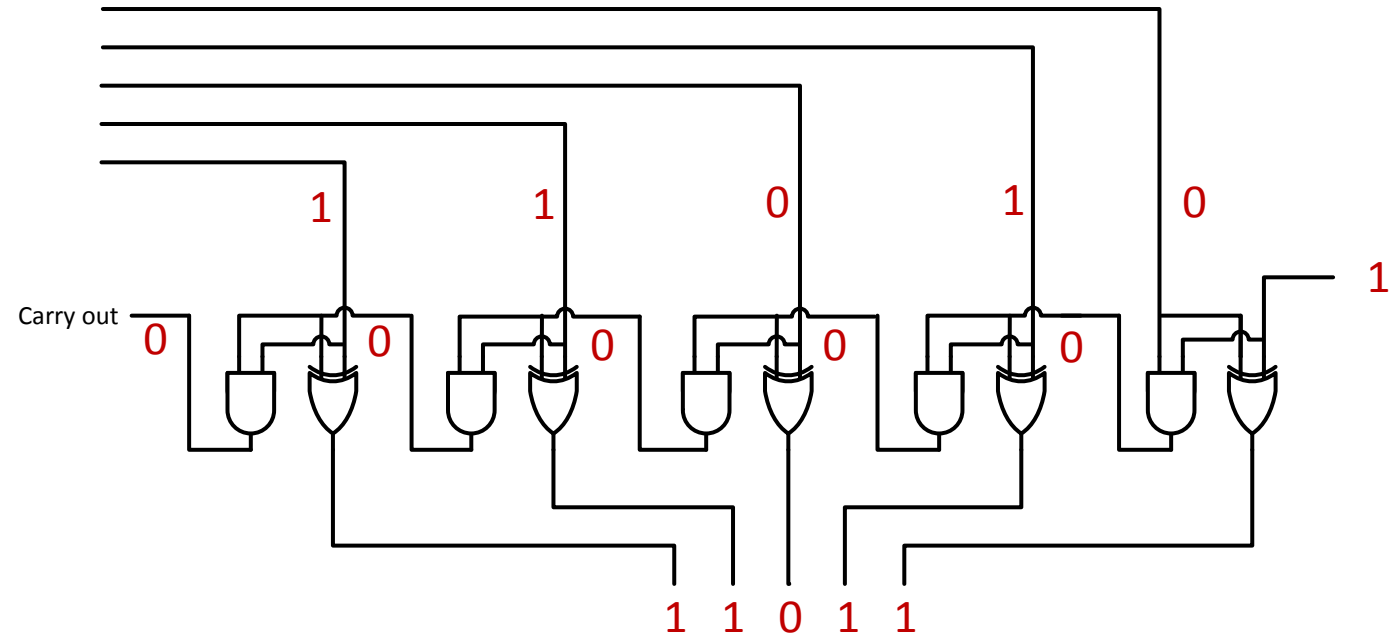
**SUB opcode**  
01110 - 00101  
14 - 5

# Incrementor

INC opcode  
11010 + 00001

Left bus into ALU

1 1 0 1 0



1 1 0 1 1

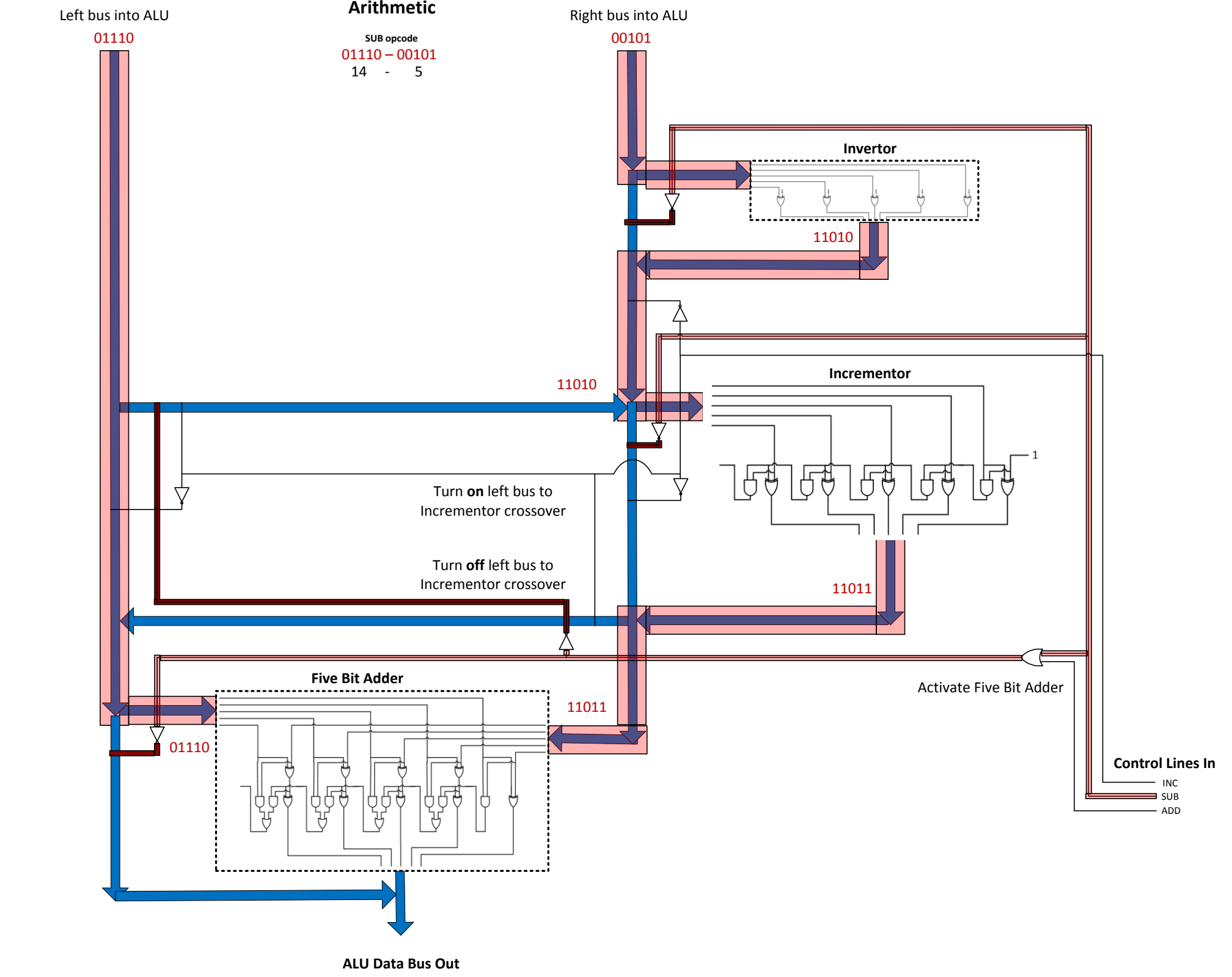
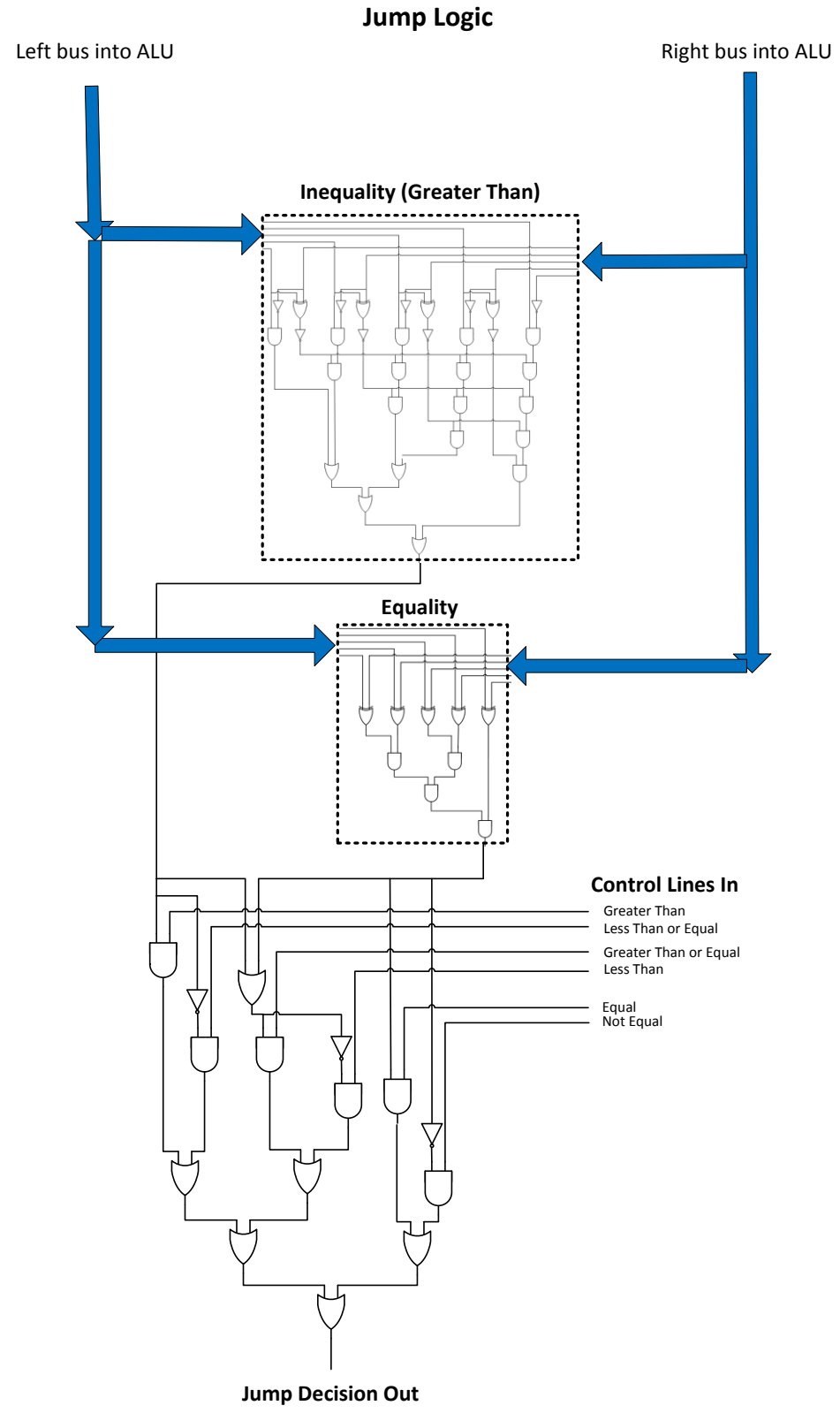
O u t p u t  
Input number + 1

1 1 0 1 0  
+ 0 0 0 0 1  

---

1 1 0 1 1

# ALU (Arithmetic and Logic Unit)



# Five Bit Adder

**SUB opcode**

01110 - 00101

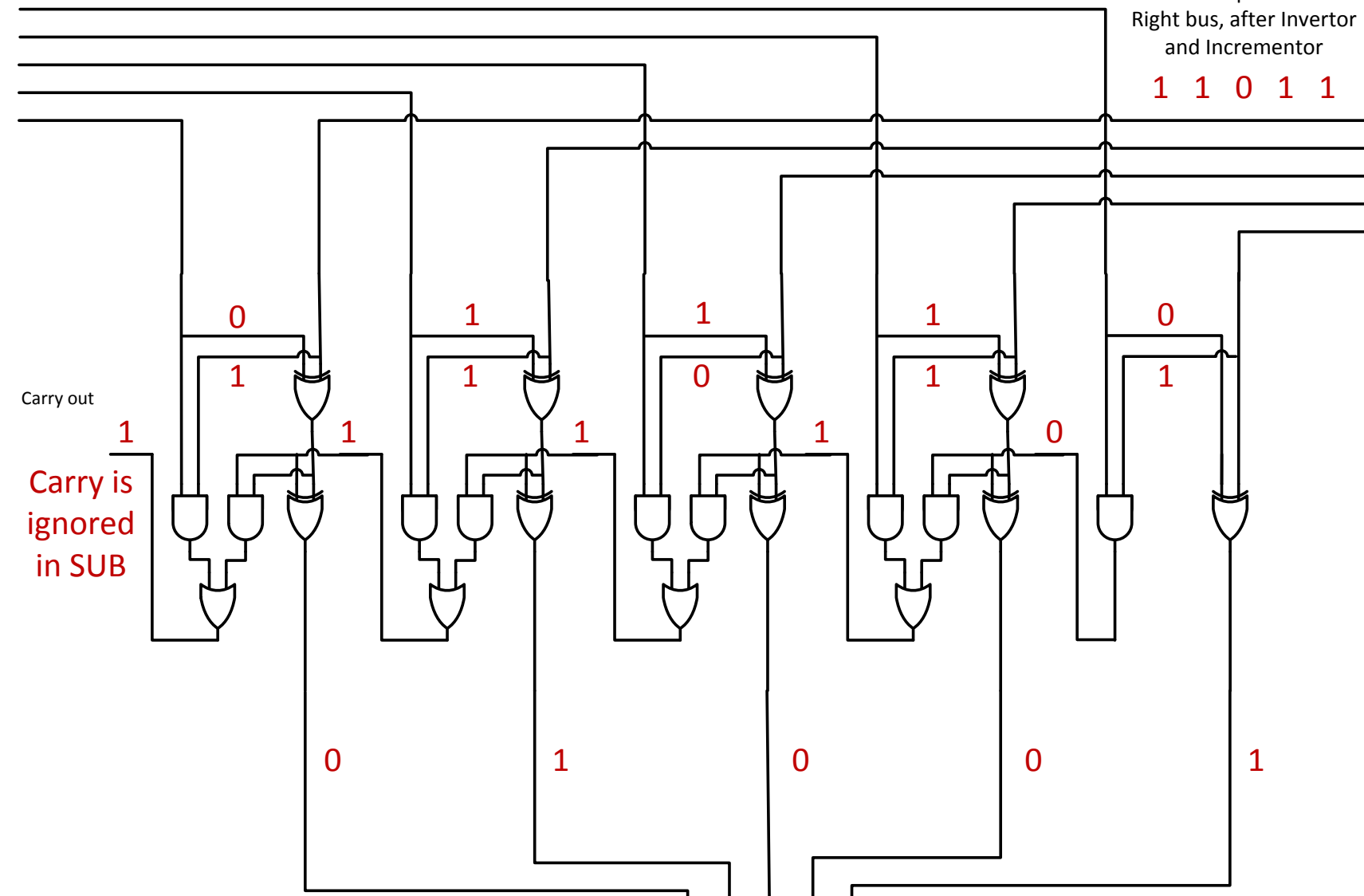
14 - 5

Two's Compiment from  
Right bus, after Invertor  
and Incrementor

1 1 0 1 1

Left bus from ALU

0 1 1 1 0



Carry out

Carry is  
ignored  
in SUB

Decimal  
equivalent

14  
+ (-5)  
9

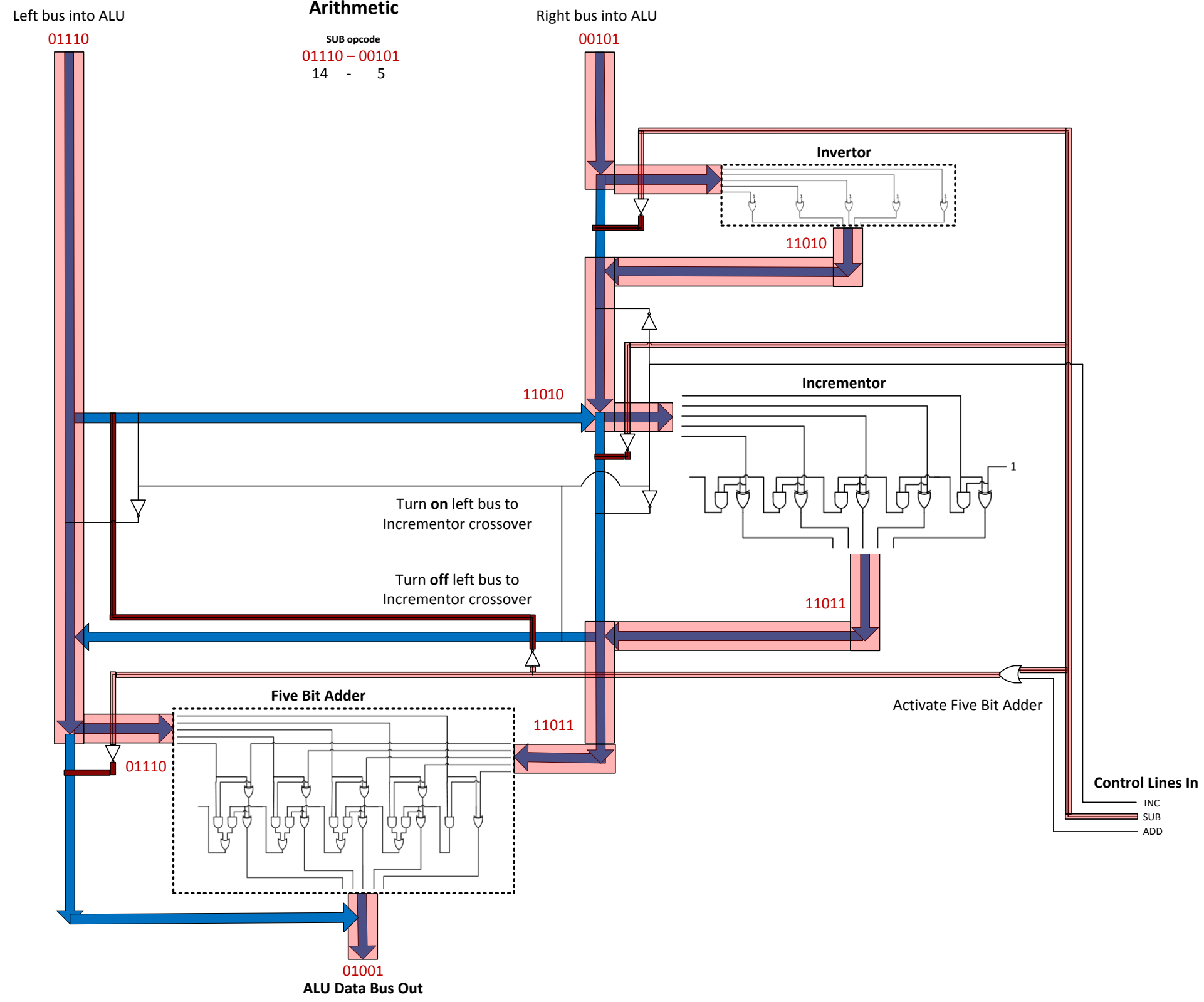
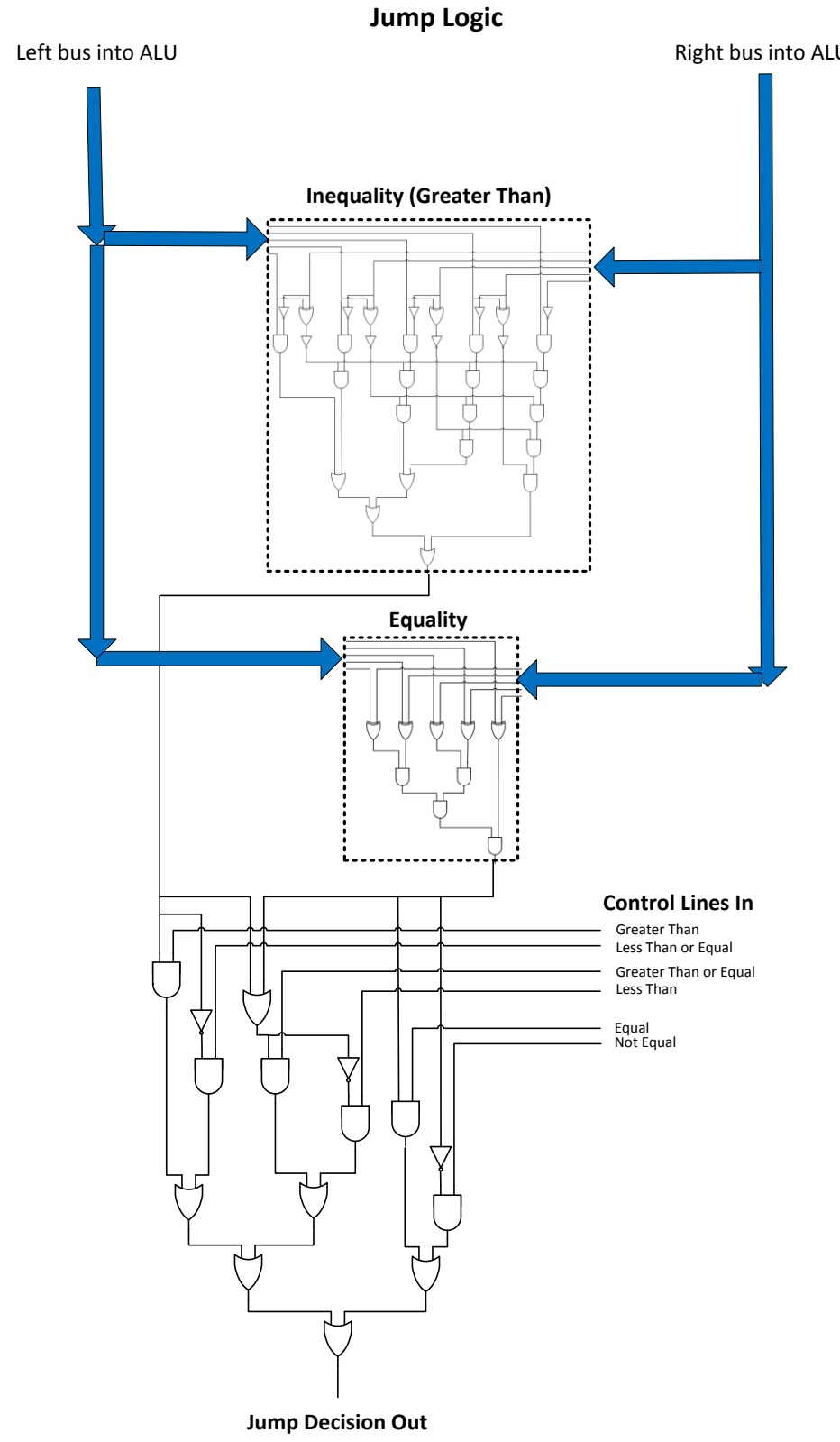
0 1 0 0 1

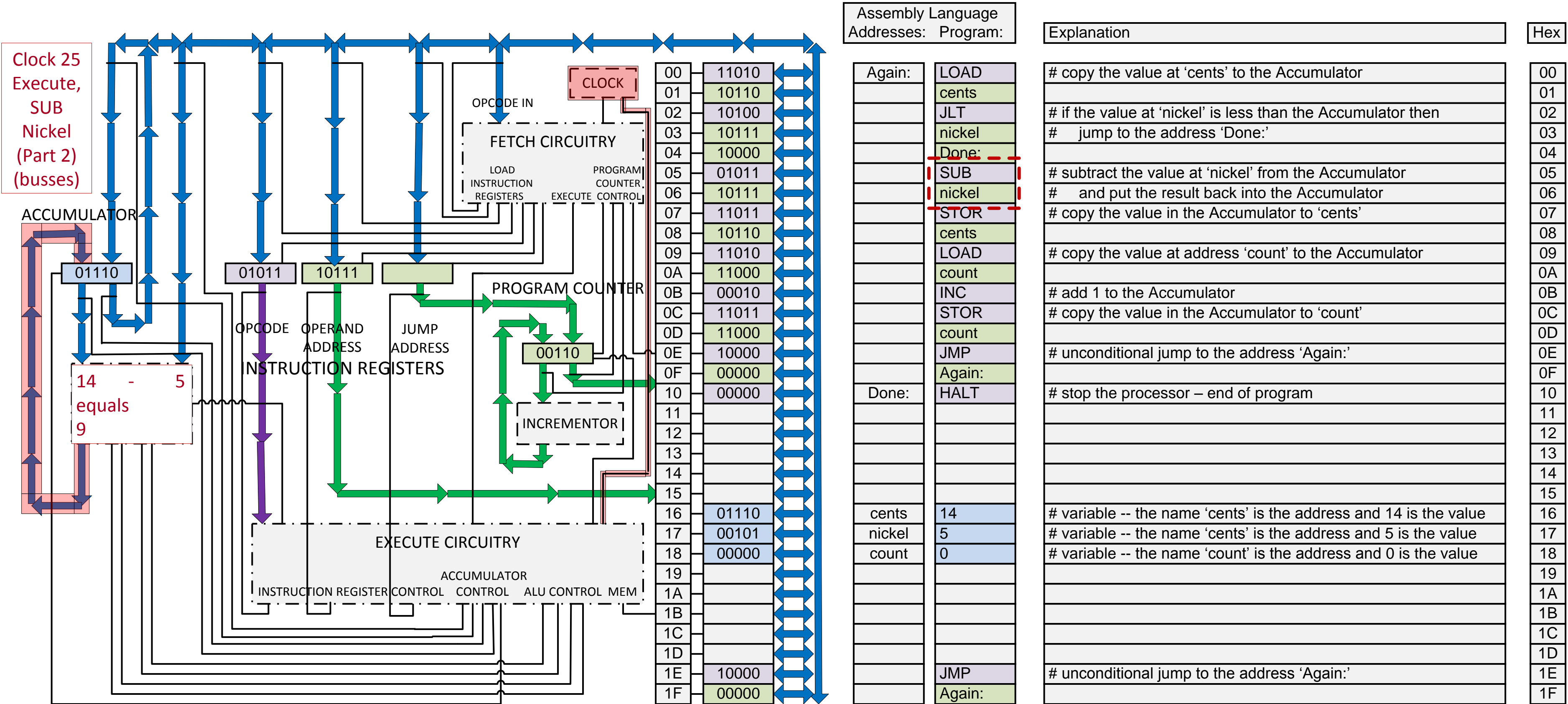
O u t p u t

Accurate addition of two 5-bit binary numbers.

0 1 1 1 0  
+ 1 1 0 1 1  
-----  
0 1 0 0 1

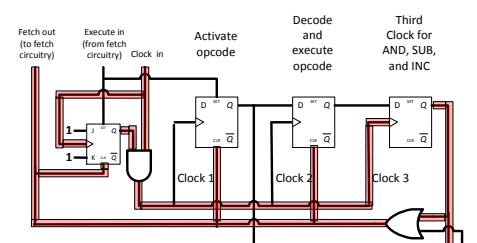
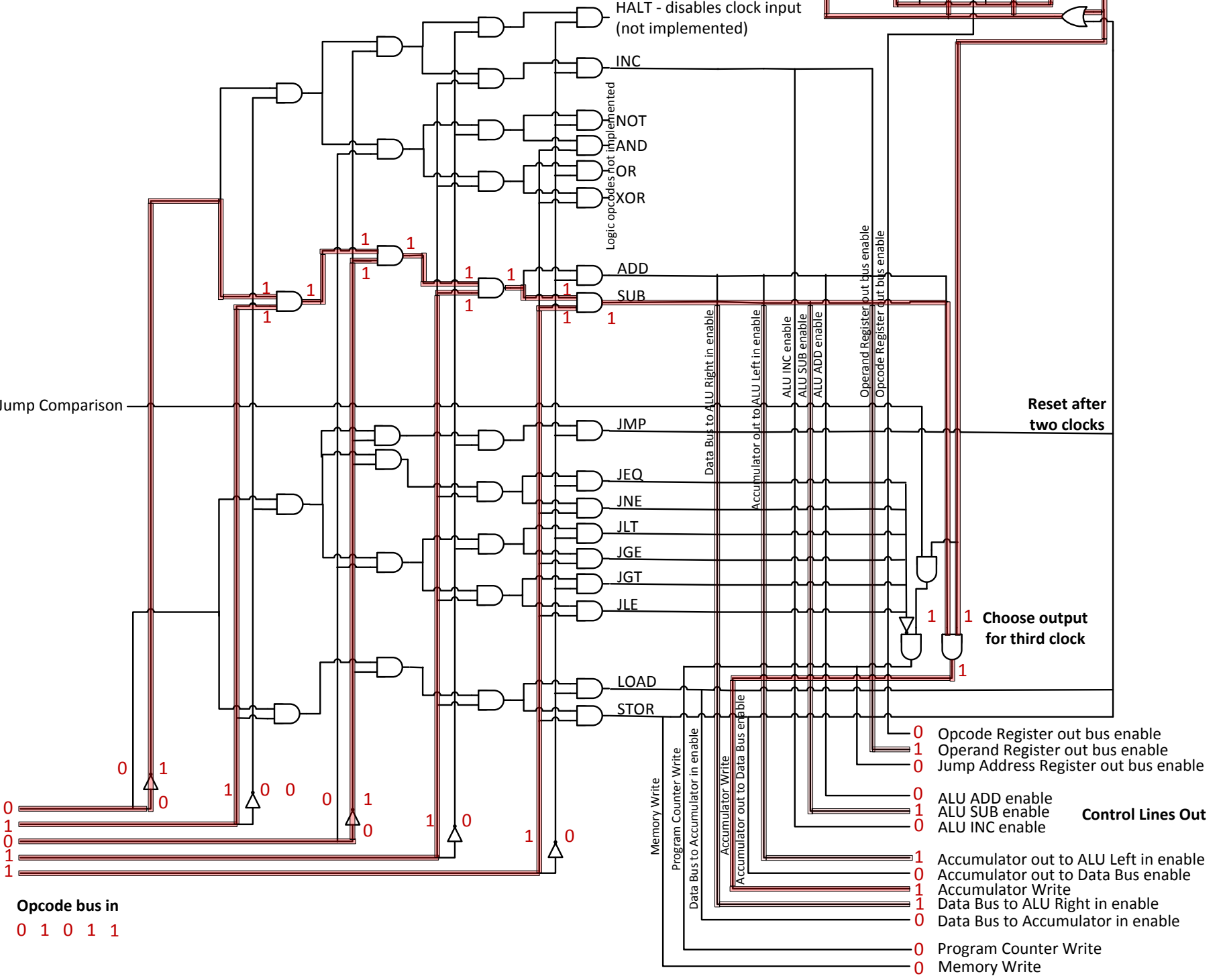
# ALU (Arithmetic and Logic Unit)





# Execute Circuitry

**SUB opcode**  
0 1 0 1 1



Fetch out (to fetch circuitry)  
Execute in (from fetch circuitry)  
Clock in  
Activate opcode  
Decode and execute opcode  
Third Clock for AND, SUB, and INC

HALT - disables clock input (not implemented)

INC

NOT  
AND  
OR  
XOR  
Logic opcodes not implemented

ADD  
SUB

JMP  
JEQ  
JNE  
JLT  
JGE  
JGT  
JLE

LOAD  
STOR

Reset after two clocks

Choose output for third clock

0 Opcode Register out bus enable  
1 Operand Register out bus enable  
0 Jump Address Register out bus enable

0 ALU ADD enable  
1 ALU SUB enable  
0 ALU INC enable  
**Control Lines Out**

1 Accumulator out to ALU Left in enable  
0 Accumulator out to Data Bus enable  
1 Accumulator Write  
1 Data Bus to ALU Right in enable  
0 Data Bus to Accumulator in enable

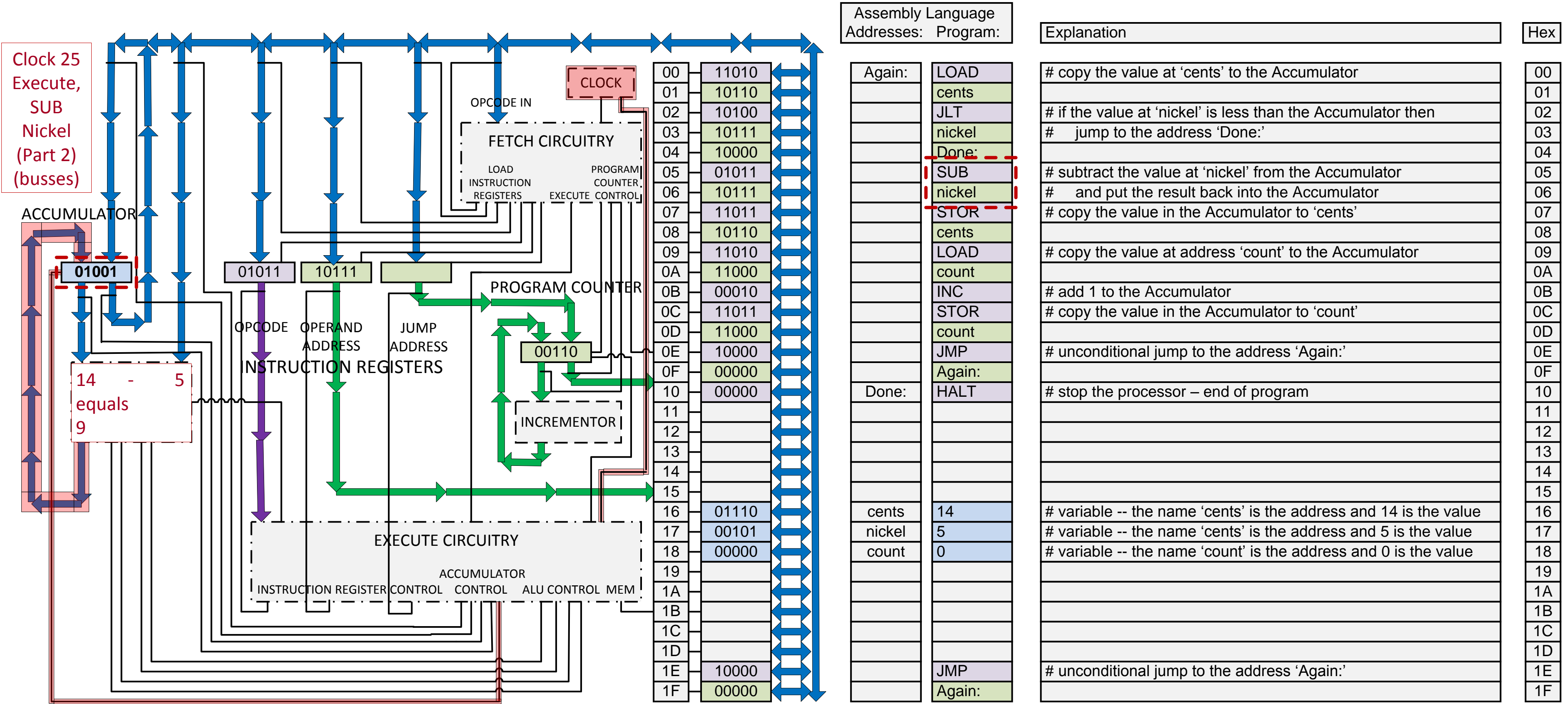
0 Program Counter Write  
0 Memory Write

**Opcode bus in**  
0 1 0 1 1

ALU Jump Comparison

Memory Write  
Program Counter Write  
Data Bus to Accumulator in enable  
Accumulator Write  
Accumulator out to Data Bus enable

Data Bus to ALU Right in enable  
Accumulator out to ALU Left in enable  
ALU INC enable  
ALU SUB enable  
ALU ADD enable  
Operand Register out bus enable  
Opcode Register out bus enable





Optional Next Presentations:

Execute last JLT opcode, clocks 109 to 117

or

Execute full program with separate control line and bus slides for each clock

**End of Presentation**