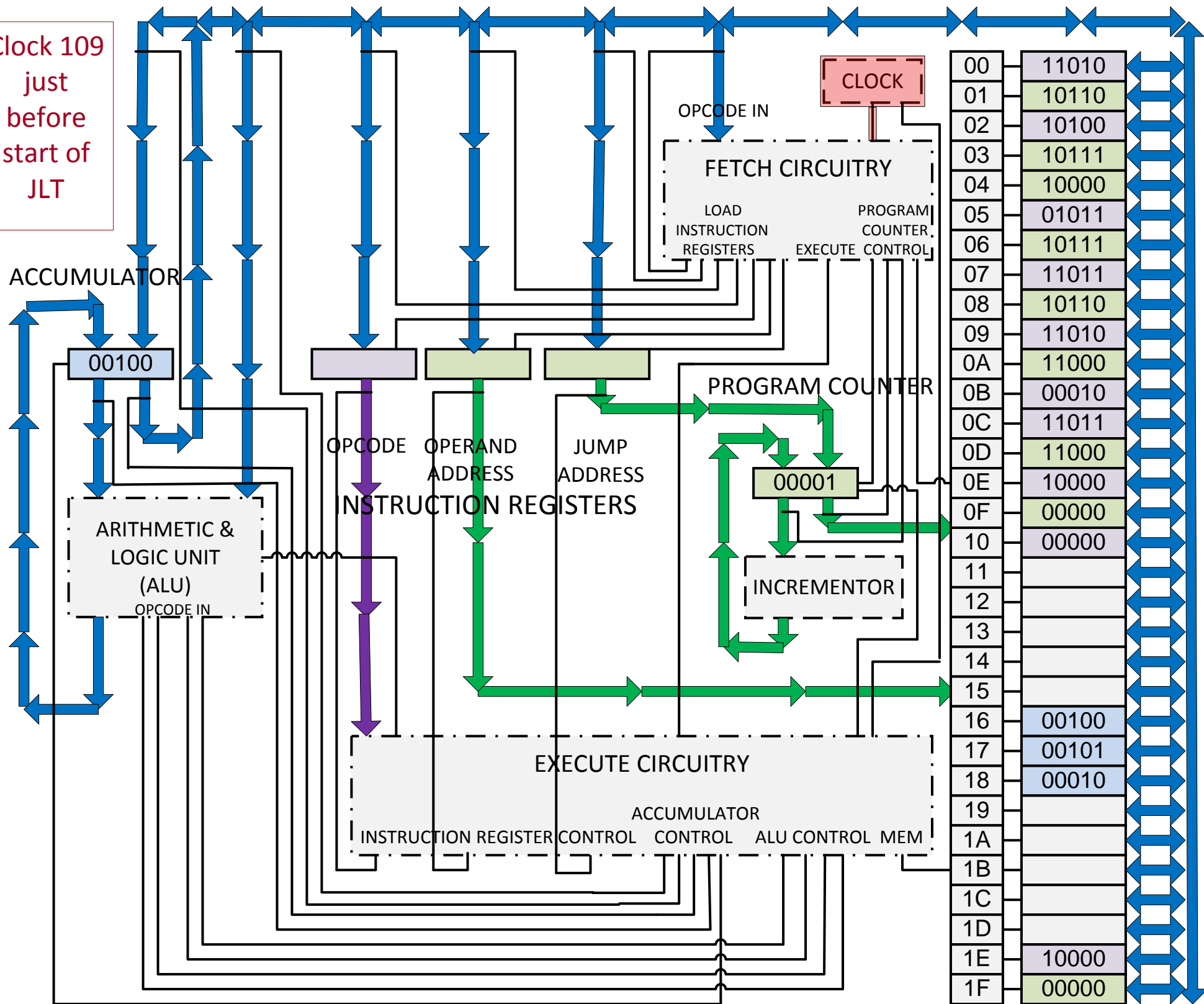


Clock 109
just
before
start of
JLT



Assembly Language
Addresses: Program:

Address	Instruction
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 4
17	nickel 5
18	count 2
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

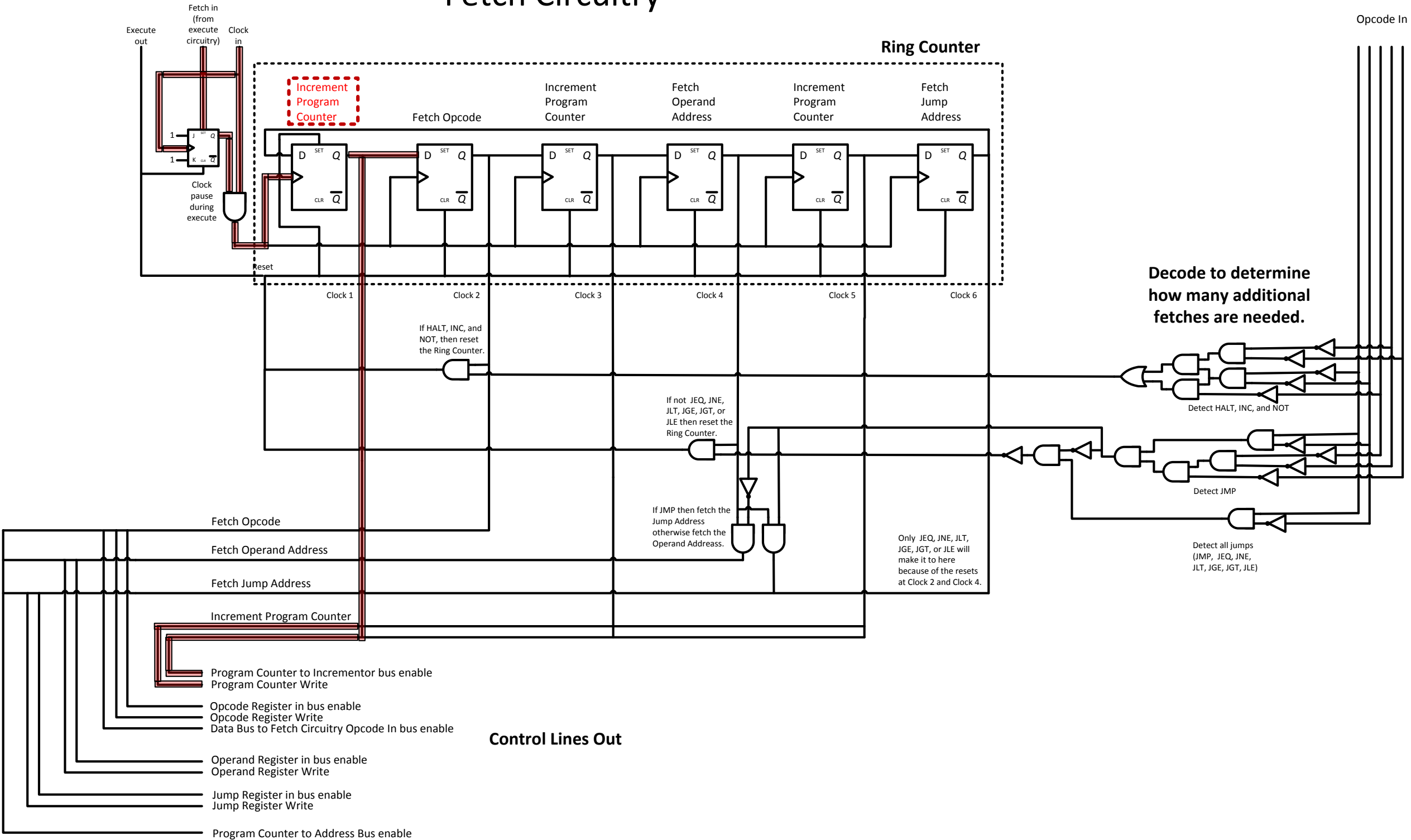
Explanation

copy the value at 'cents' to the Accumulator
if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'
subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator
copy the value in the Accumulator to 'cents'
copy the value at address 'count' to the Accumulator
add 1 to the Accumulator
copy the value in the Accumulator to 'count'
unconditional jump to the address 'Again:'
stop the processor – end of program
variable -- the name 'cents' is the address and 14 is the value
variable -- the name 'cents' is the address and 5 is the value
variable -- the name 'count' is the address and 0 is the value
unconditional jump to the address 'Again:'

Hex

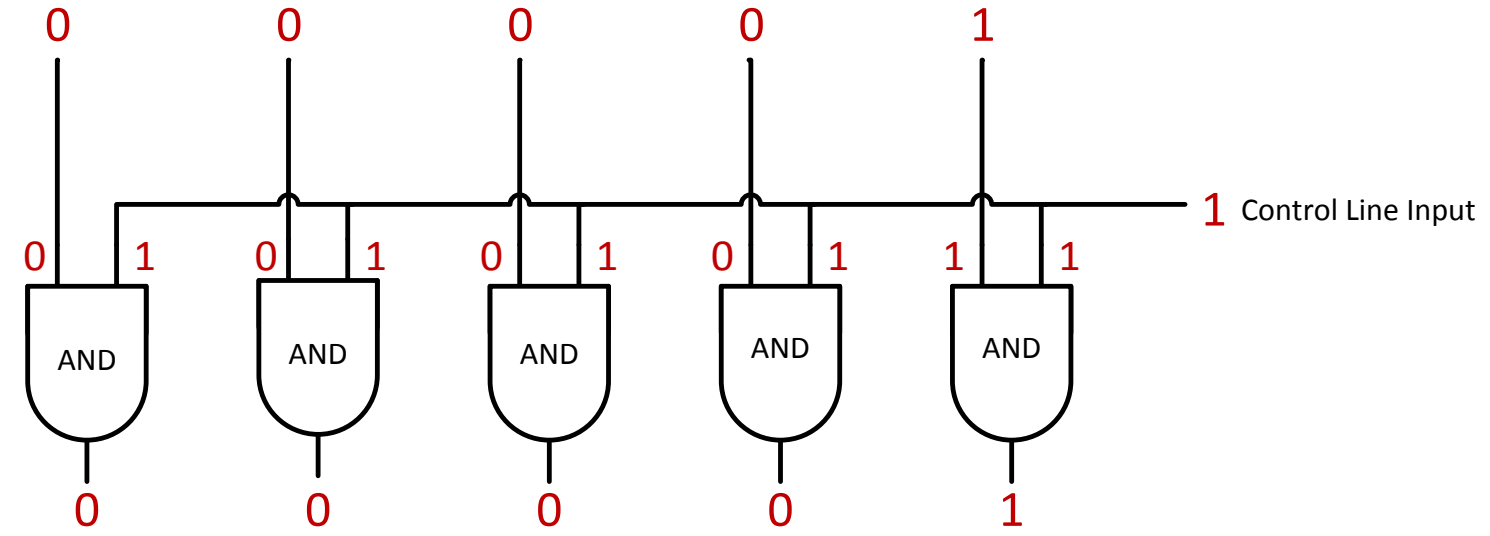
00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
10
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F

Fetch Circuitry



Bus Control

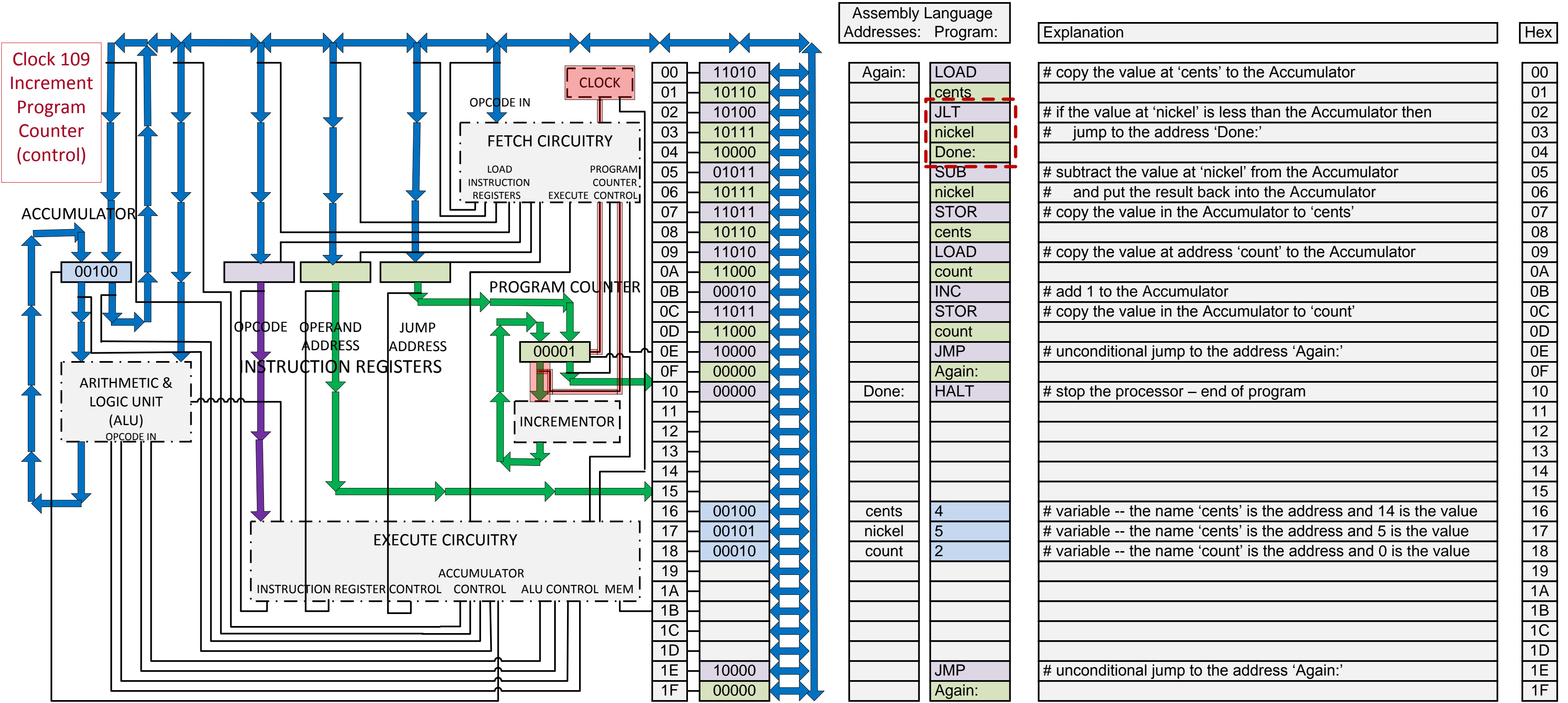
I n p u t s F r o m B u s



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

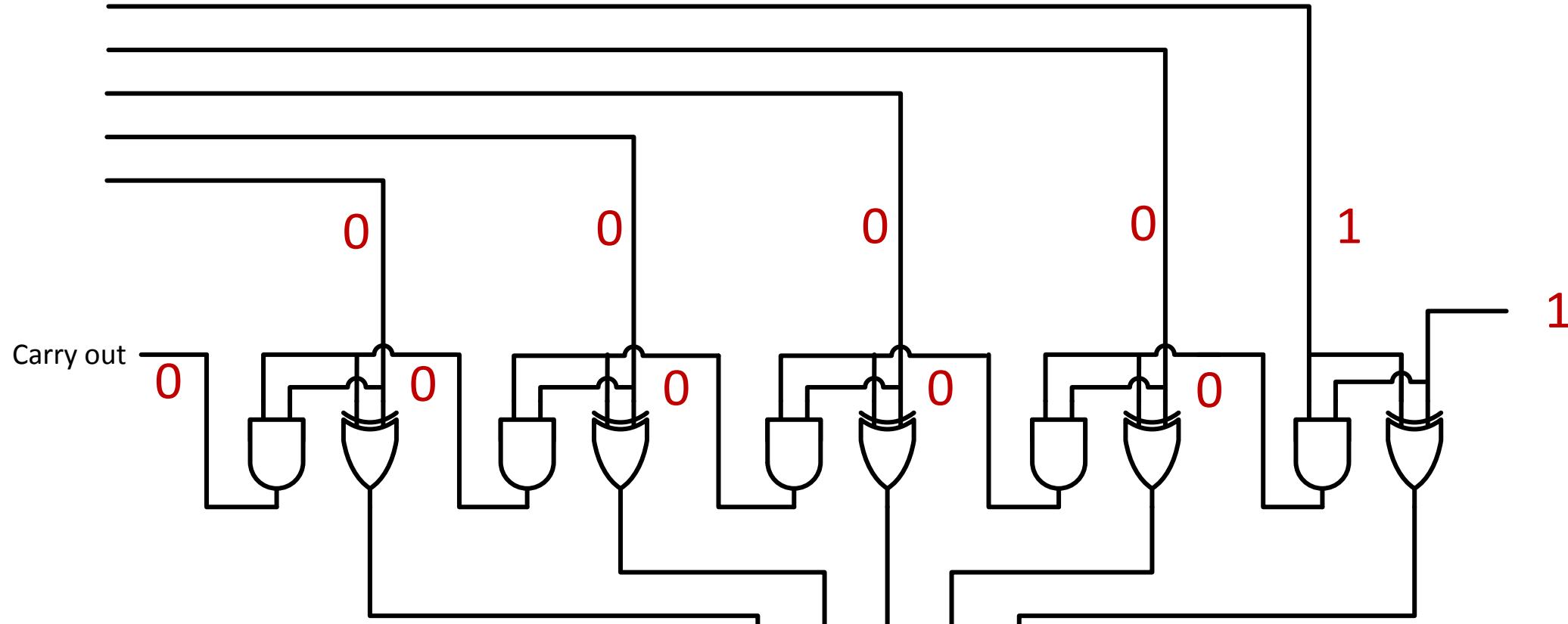
INC opcode

00001 + 00001

1 + 1

Left bus into ALU

0 0 0 0 1



Decimal equivalent

0 0 0 0 1
+ 0 0 0 0 1

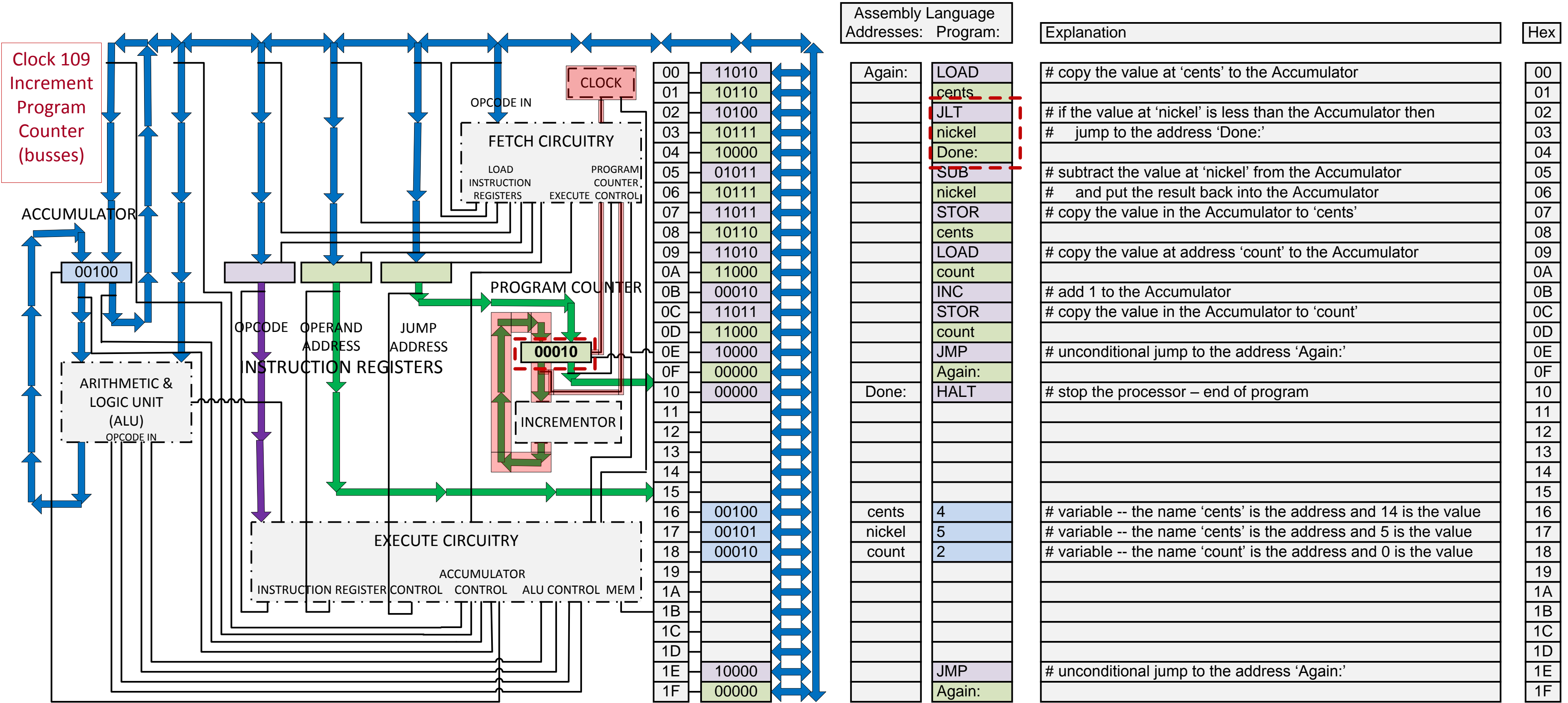
0 0 0 1 0

+ 1

2

0 0 0 1 0

Output
Input number + 1



Clock 109
Increment
Program
Counter
(busses)

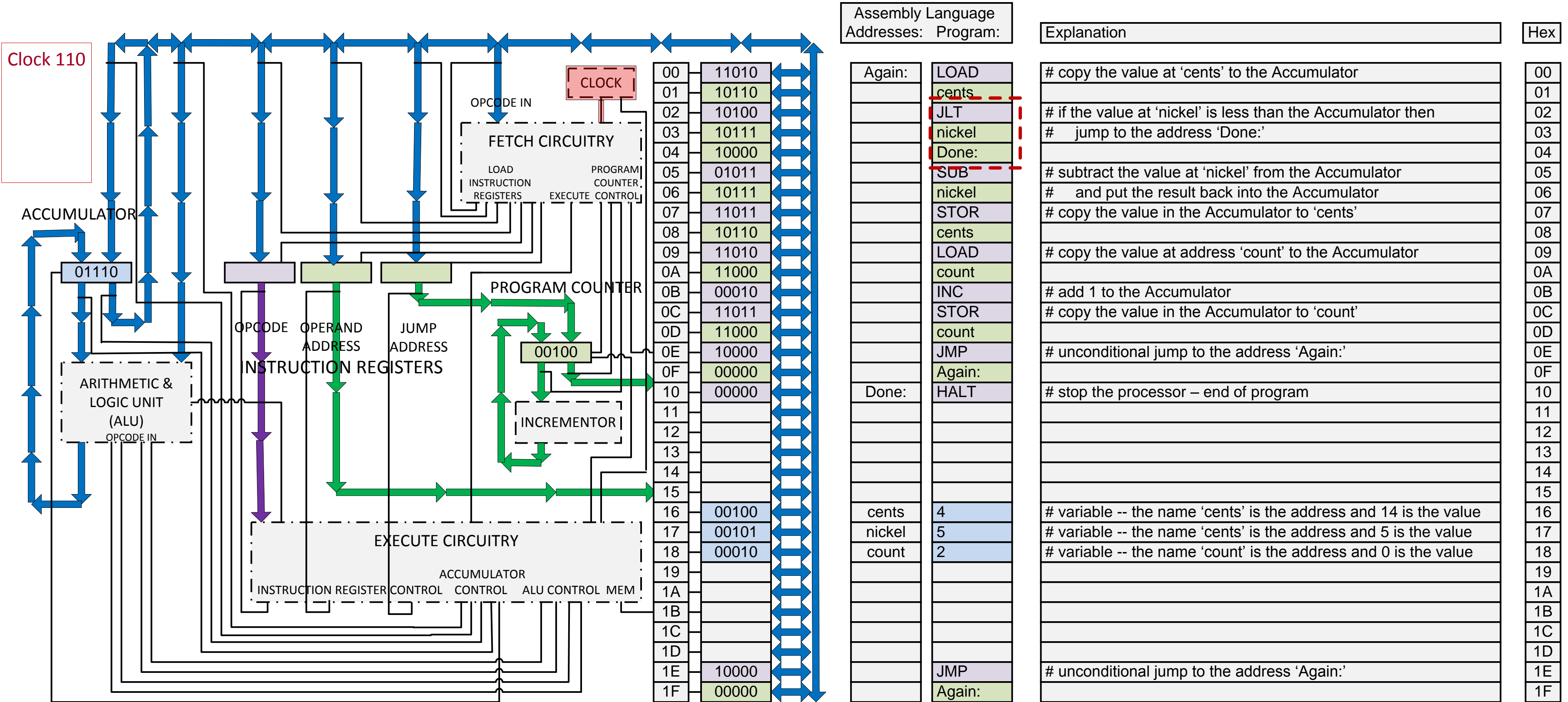
Assembly Language
Addresses: Program:

Explanation

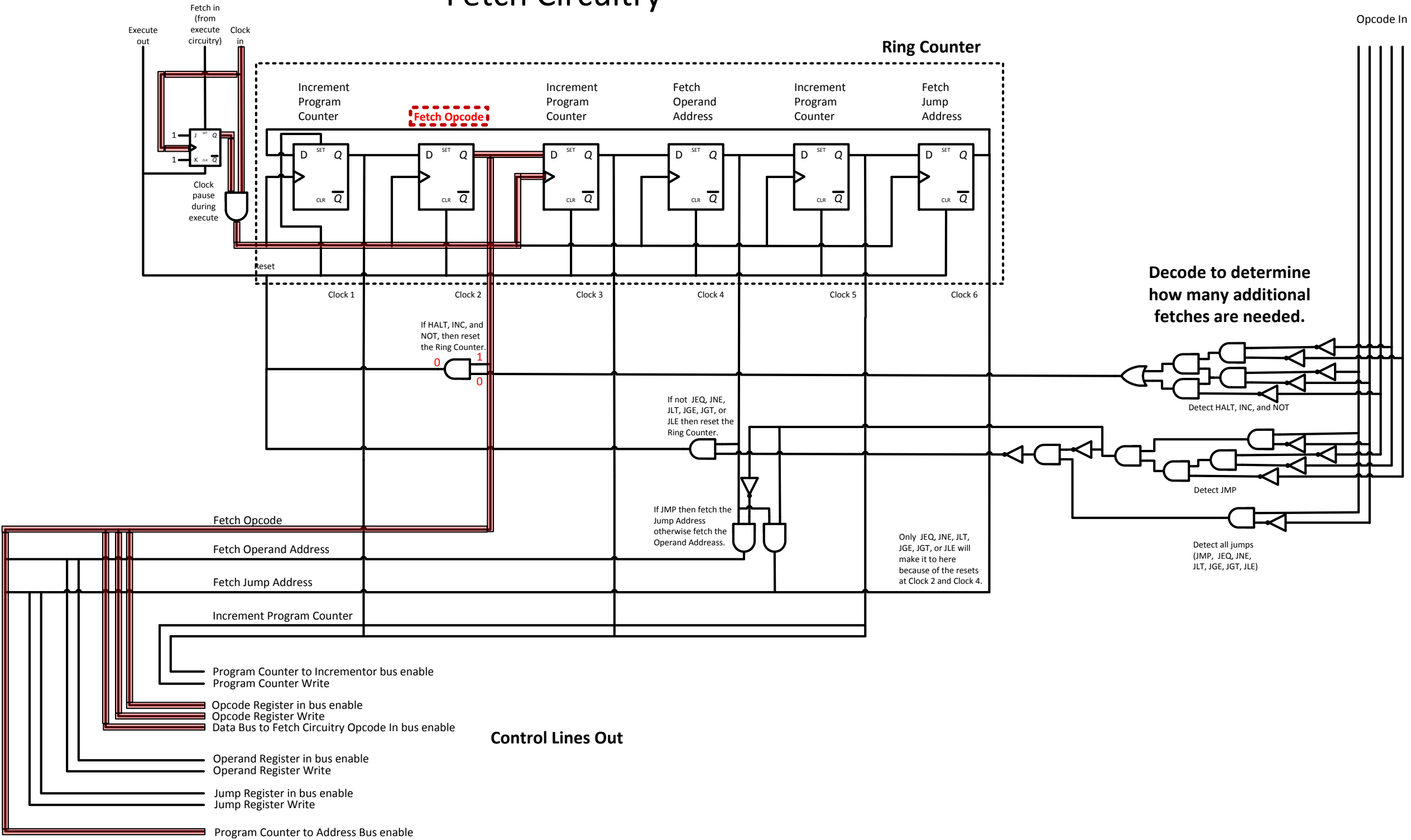
Hex

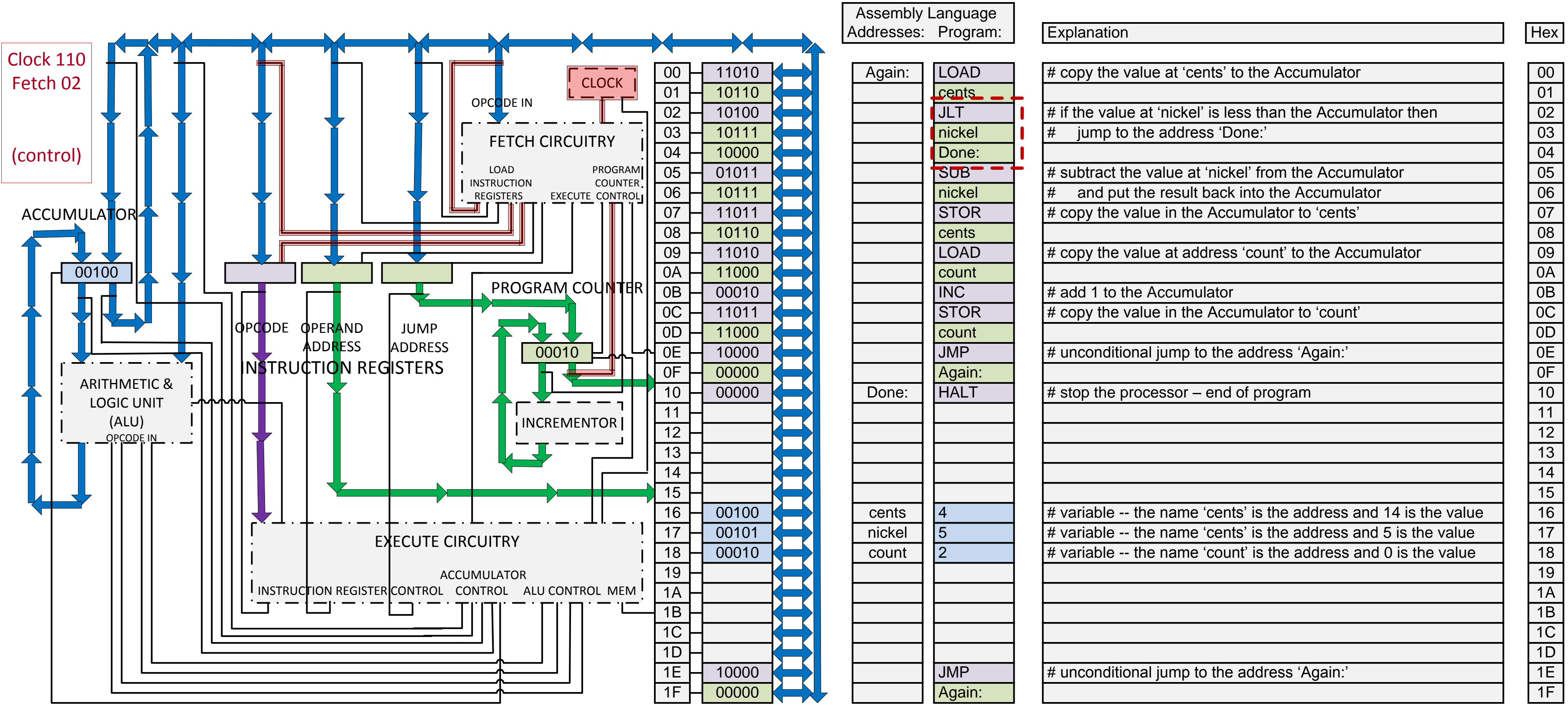
Address	Hex	Explanation	Hex
00	11010		00
01	10110	# copy the value at 'cents' to the Accumulator	01
02	10100		02
03	10111	# if the value at 'nickel' is less than the Accumulator then	03
04	10000	# jump to the address 'Done:.'	04
05	01011		05
06	10111	# subtract the value at 'nickel' from the Accumulator	06
07	11011	# and put the result back into the Accumulator	07
08	10110	# copy the value in the Accumulator to 'cents'	08
09	11010		09
0A	11000	# copy the value at address 'count' to the Accumulator	0A
0B	00010	# add 1 to the Accumulator	0B
0C	11011	# copy the value in the Accumulator to 'count'	0C
0D	11000		0D
0E	10000	# unconditional jump to the address 'Again:.'	0E
0F	00000		0F
10	00000		10
11			11
12			12
13			13
14			14
15			15
16	00100	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	# variable -- the name 'cents' is the address and 5 is the value	17
18	00010	# variable -- the name 'count' is the address and 0 is the value	18
19			19
1A			1A
1B			1B
1C			1C
1D			1D
1E	10000	# unconditional jump to the address 'Again:.'	1E
1F	00000		1F

Address	Hex	Explanation	Hex
00	11010		00
01	10110	# copy the value at 'cents' to the Accumulator	01
02	10100		02
03	10111	# if the value at 'nickel' is less than the Accumulator then	03
04	10000	# jump to the address 'Done:.'	04
05	01011		05
06	10111	# subtract the value at 'nickel' from the Accumulator	06
07	11011	# and put the result back into the Accumulator	07
08	10110	# copy the value in the Accumulator to 'cents'	08
09	11010		09
0A	11000	# copy the value at address 'count' to the Accumulator	0A
0B	00010	# add 1 to the Accumulator	0B
0C	11011	# copy the value in the Accumulator to 'count'	0C
0D	11000		0D
0E	10000	# unconditional jump to the address 'Again:.'	0E
0F	00000		0F
10	00000		10
11			11
12			12
13			13
14			14
15			15
16	00100	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	# variable -- the name 'cents' is the address and 5 is the value	17
18	00010	# variable -- the name 'count' is the address and 0 is the value	18
19			19
1A			1A
1B			1B
1C			1C
1D			1D
1E	10000	# unconditional jump to the address 'Again:.'	1E
1F	00000		1F



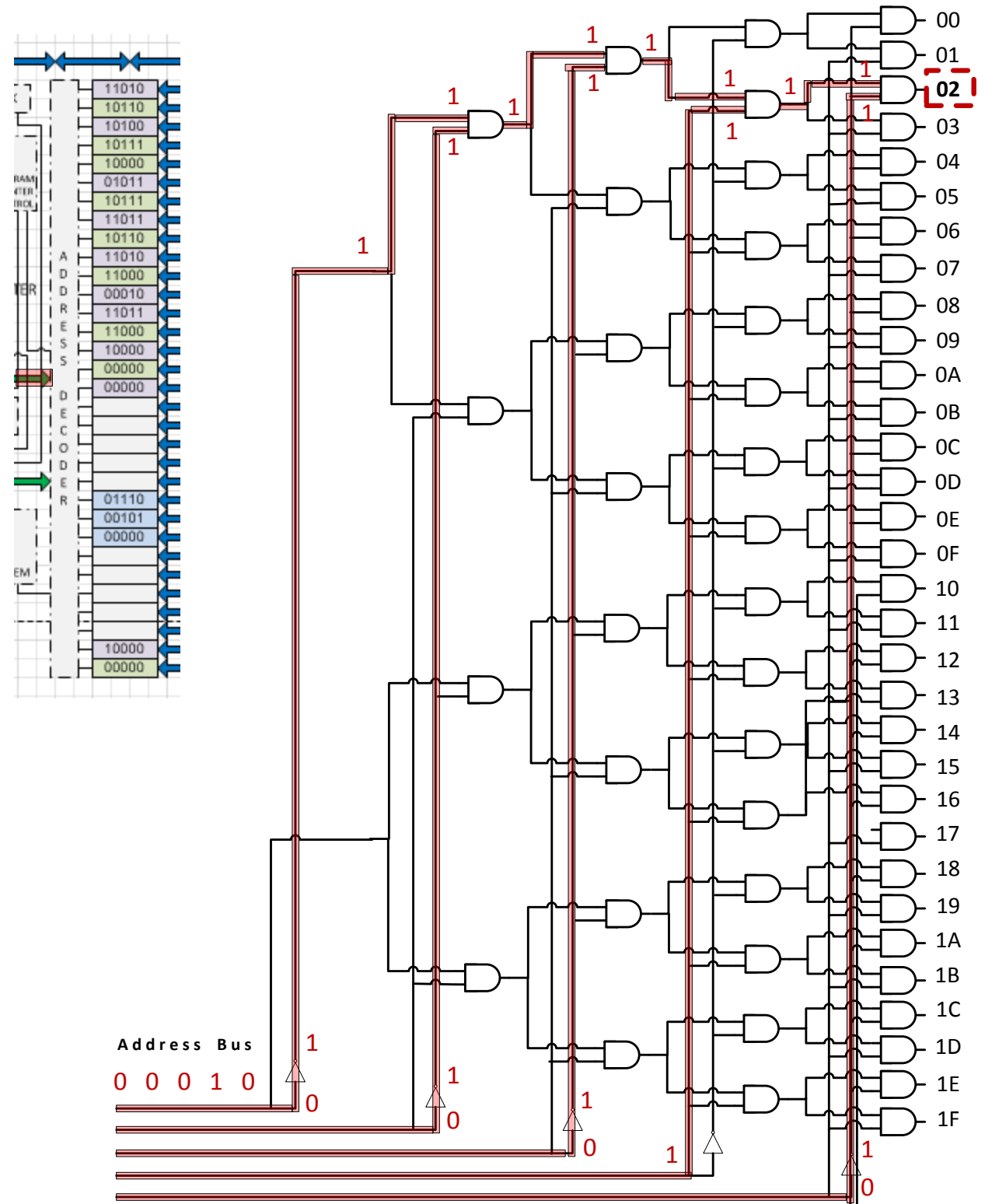
Fetch Circuitry

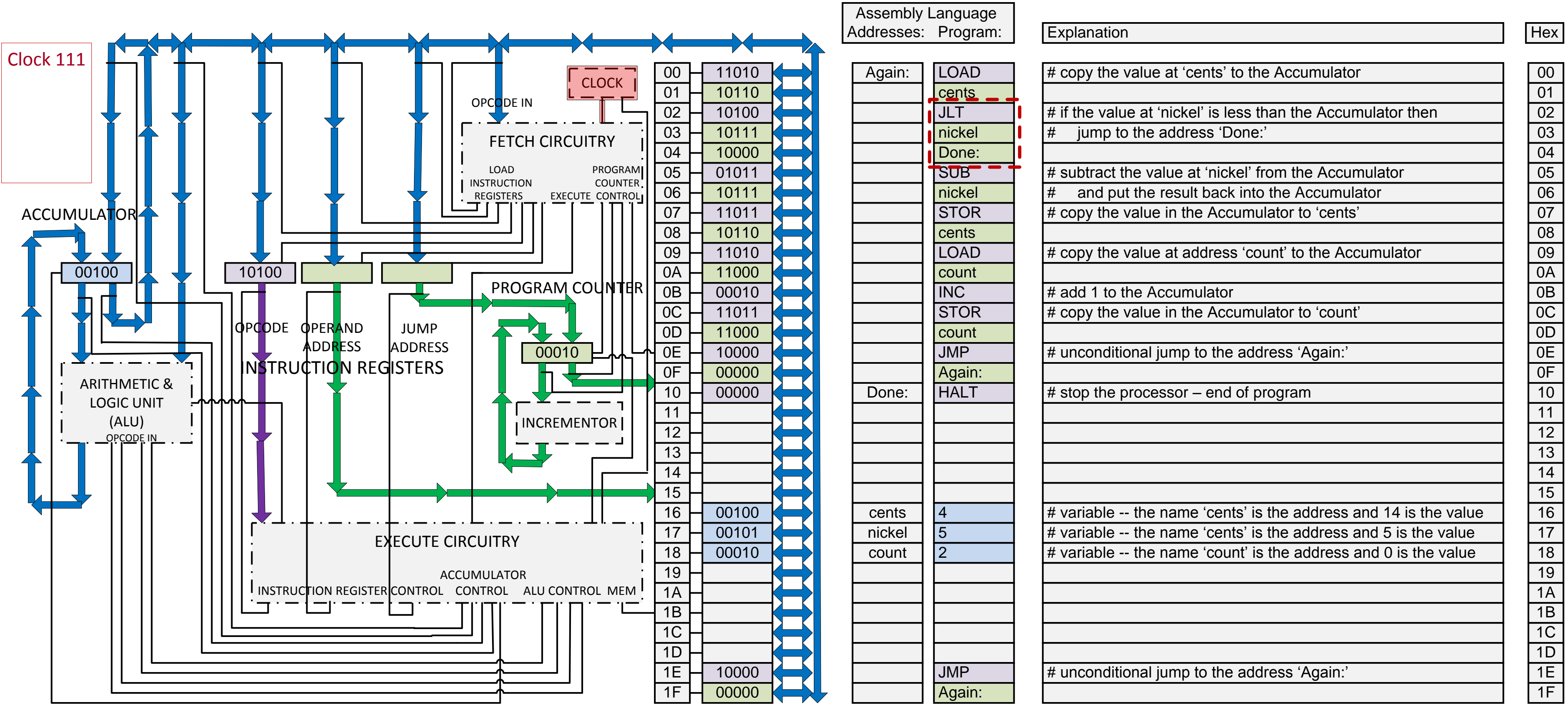




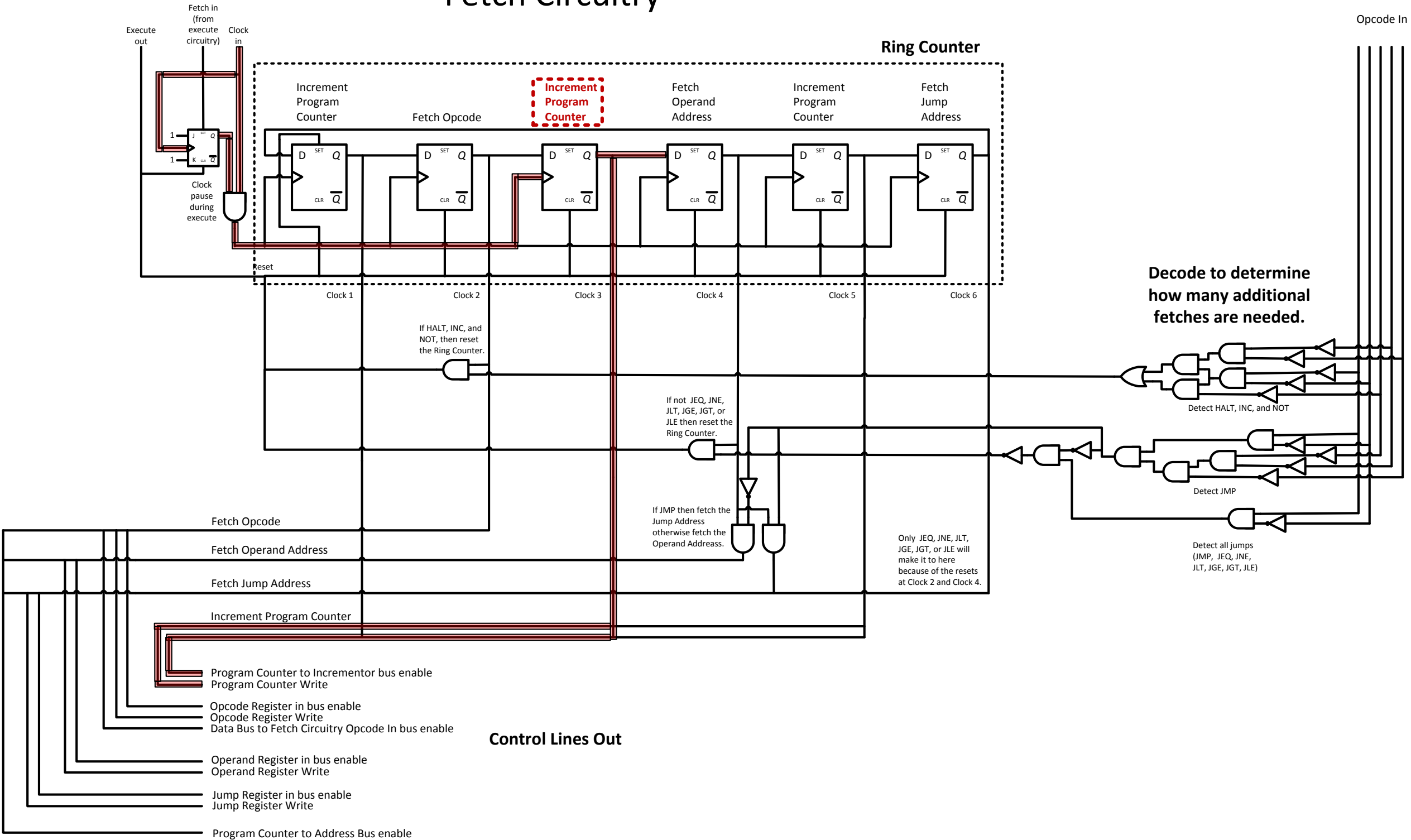
Address Decoder

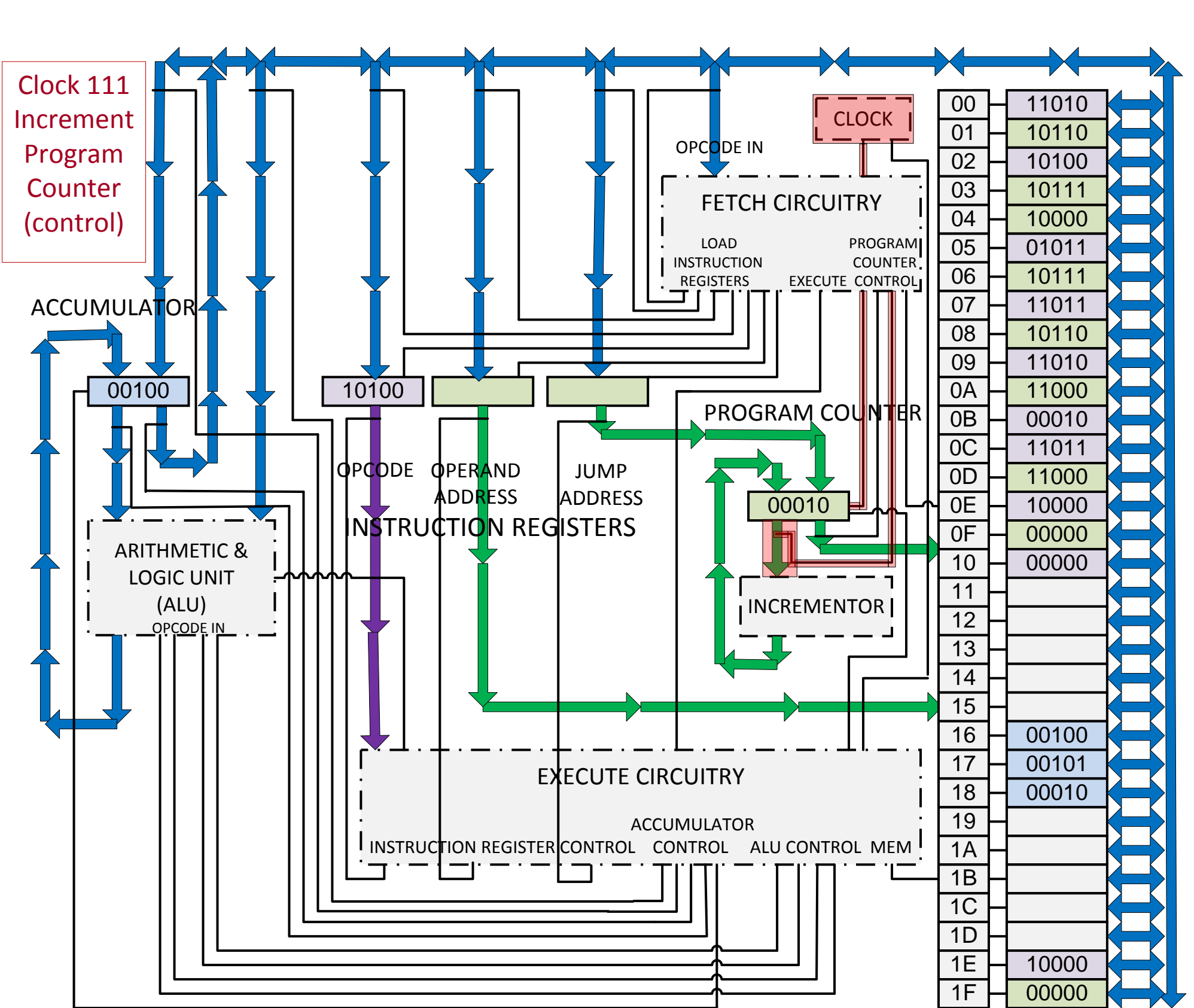
Input from Address Bus





Fetch Circuitry





Assembly Language
Addresses: Program:

Address	Instruction
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 4
17	nickel 5
18	count 2
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

Incrementor

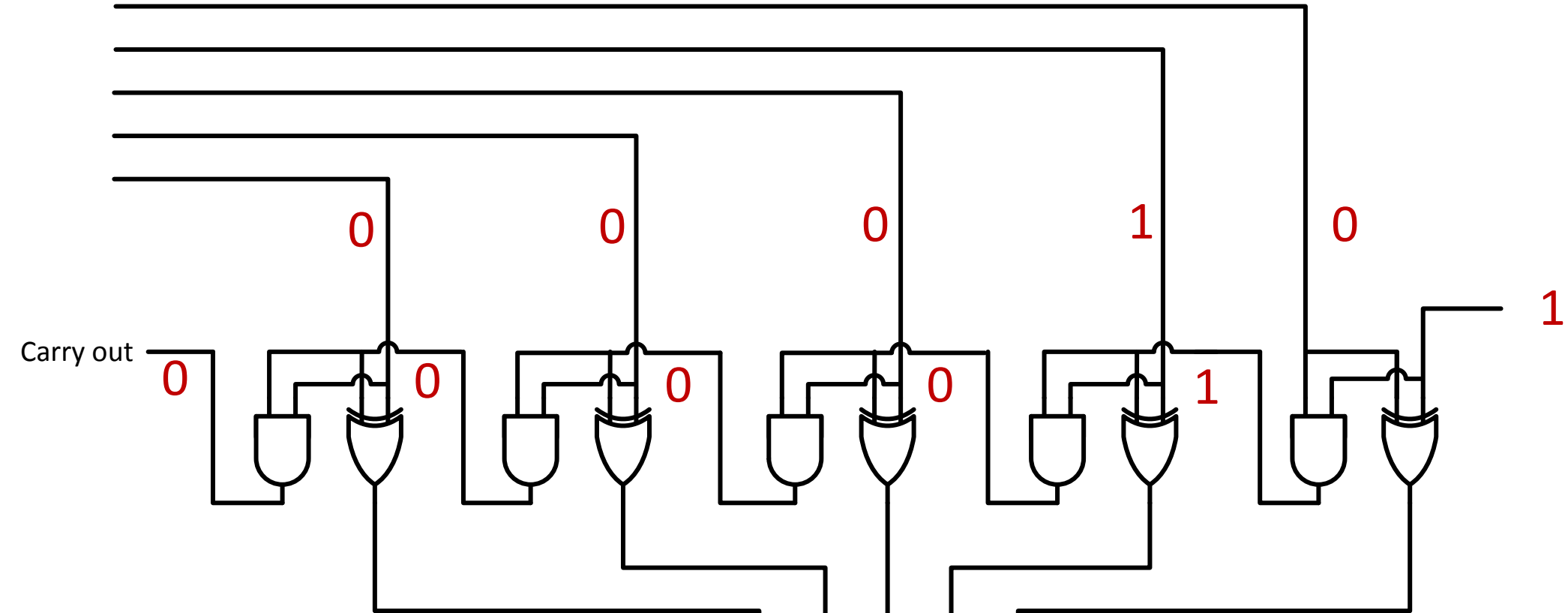
INC opcode

00010 + 00001

2 + 1

Left bus into ALU

0 0 0 1 0



Decimal equivalent

0 0 0 1 1
+ 0 0 0 0 1

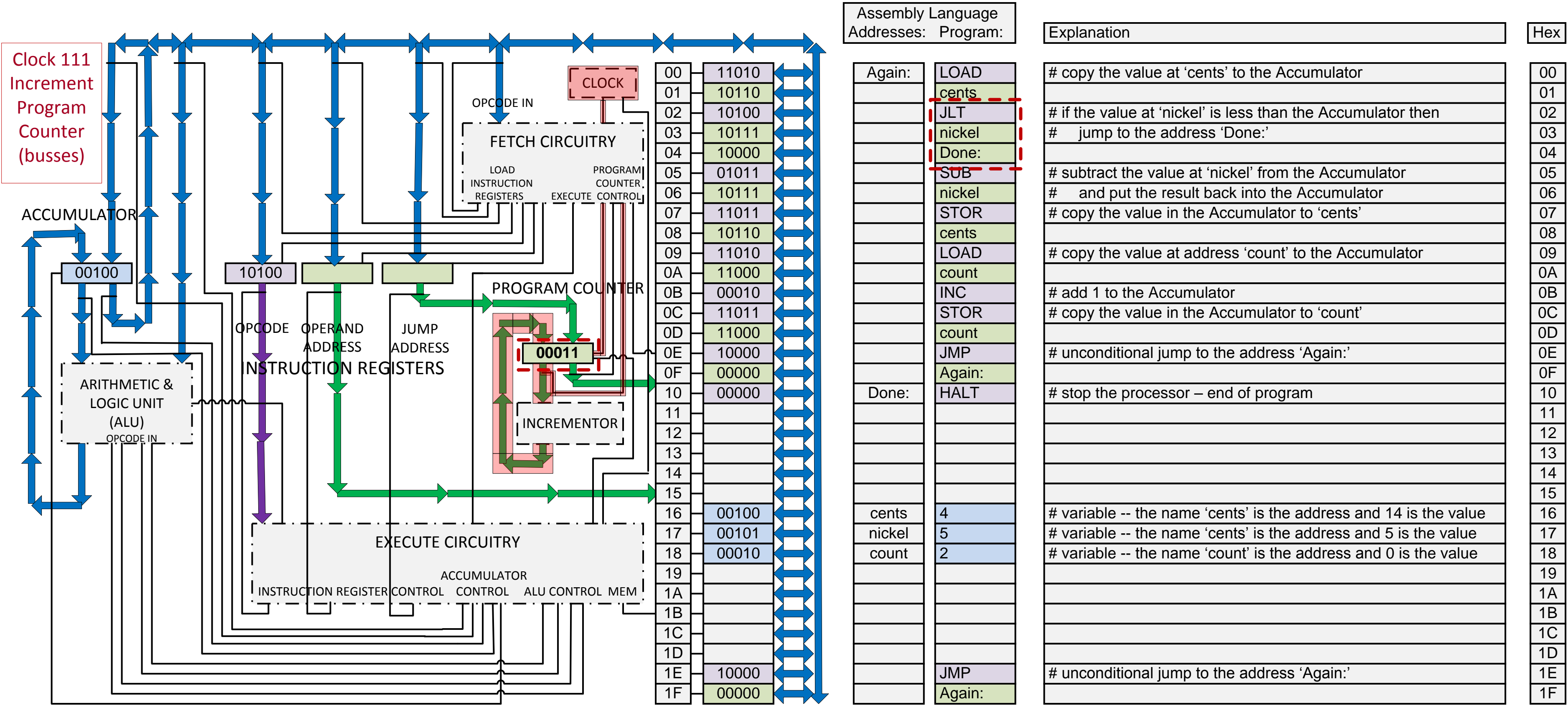
0 0 0 1 1

+ 1

3

0 0 0 1 1

Output
Input number + 1



Clock 111
Increment
Program
Counter
(busses)

ACCUMULATOR

00100

10100

PROGRAM COUNTER

00011

ARITHMETIC &
LOGIC UNIT
(ALU)

INCREMENTOR

EXECUTE CIRCUITRY

ACCUMULATOR
CONTROL

INSTRUCTION REGISTER
CONTROL

ALU CONTROL

MEM

OPCODE
INSTRUCTION
REGISTER

OPERAND
ADDRESS

JUMP
ADDRESS

FETCH CIRCUITRY

LOAD
INSTRUCTION
REGISTER

PROGRAM
COUNTER,
EXECUTE CONTROL

CLOCK

Assembly Language
Addresses: Program:

Again:

LOAD

cents

JLT

nickel

Done:

SUB

nickel

STOR

cents

LOAD

count

INC

STOR

count

JMP

Again:

Done:

HALT

cents 4

nickel 5

count 2

JMP

Again:

Explanation

copy the value at 'cents' to the Accumulator

if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'

subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator

copy the value in the Accumulator to 'cents'

copy the value at address 'count' to the Accumulator

add 1 to the Accumulator

copy the value in the Accumulator to 'count'

unconditional jump to the address 'Again:'

stop the processor – end of program

variable -- the name 'cents' is the address and 14 is the value

variable -- the name 'cents' is the address and 5 is the value

variable -- the name 'count' is the address and 0 is the value

unconditional jump to the address 'Again:'

Hex

00

01

02

03

04

05

06

07

08

09

0A

0B

0C

0D

0E

0F

10

11

12

13

14

15

16

17

18

19

1A

1B

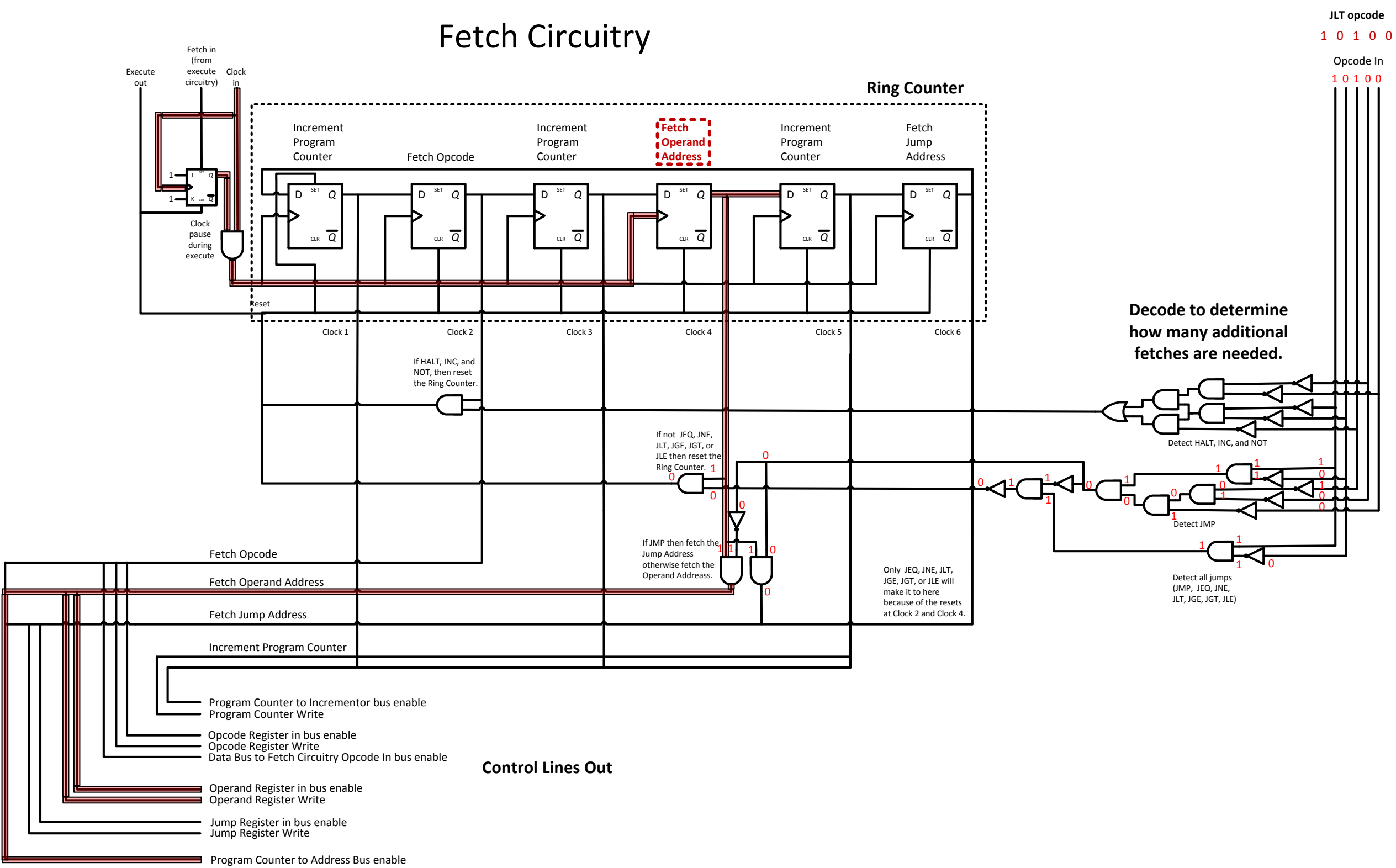
1C

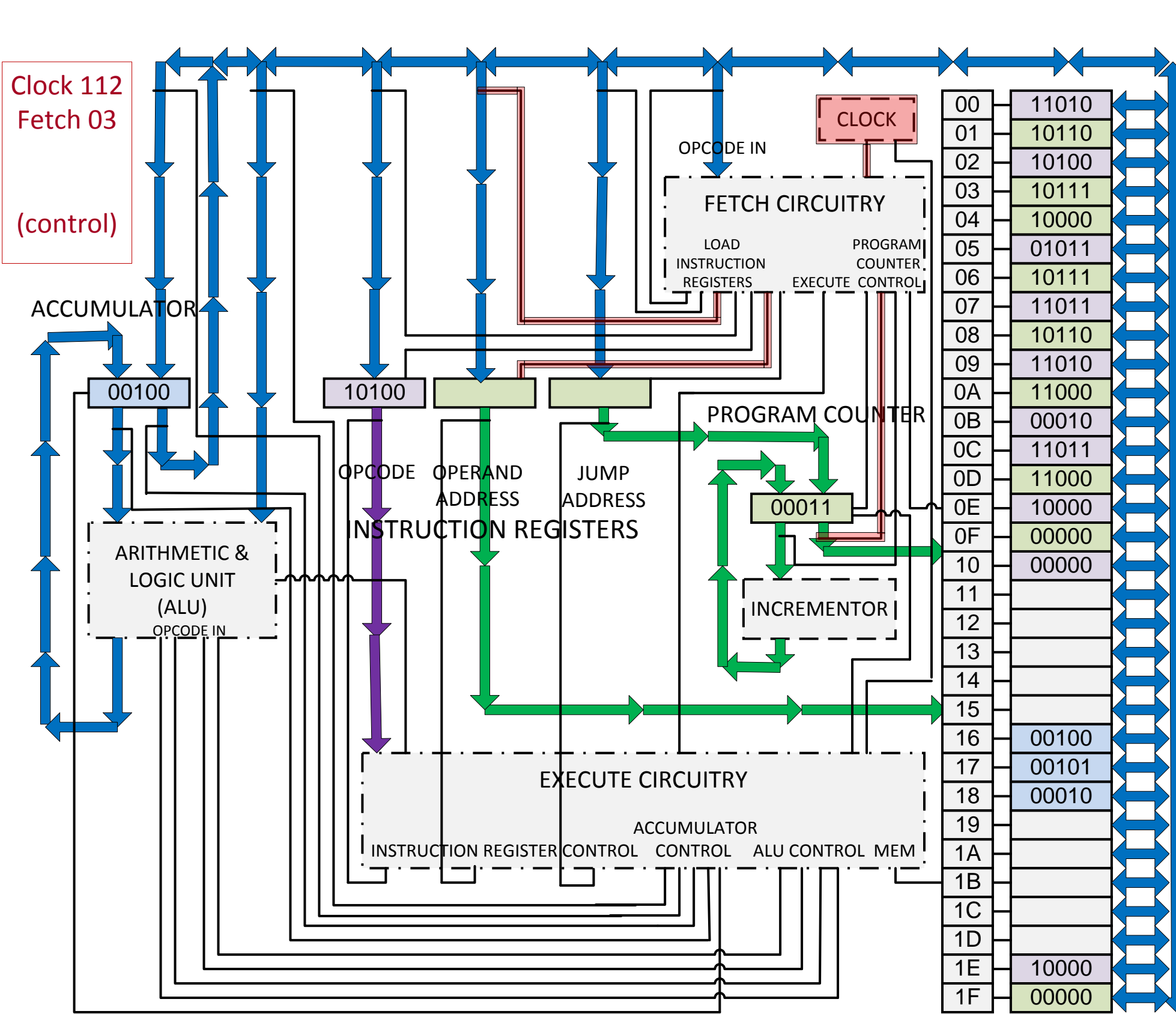
1D

1E

1F

Fetch Circuitry





Assembly Language Addresses: Program:

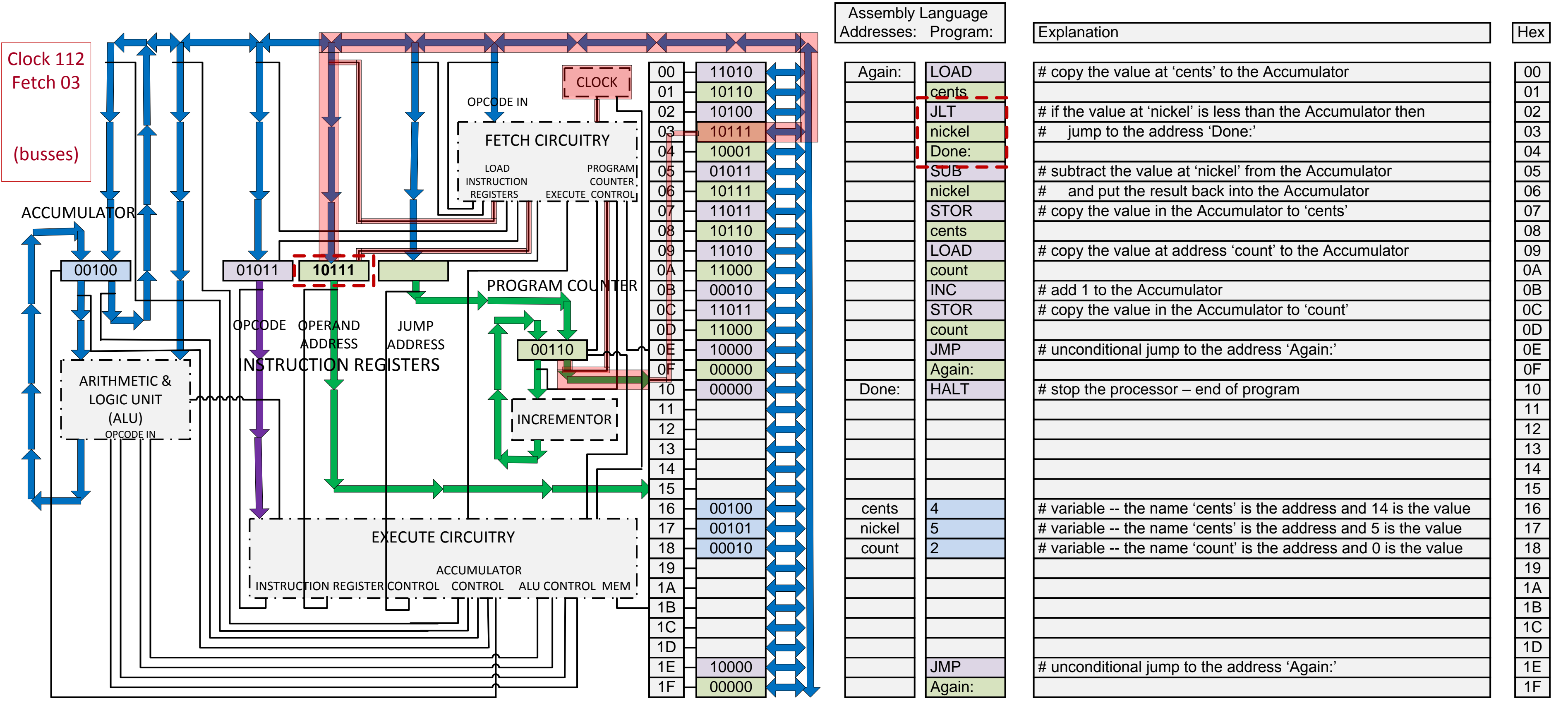
Address	Instruction
00	LOAD
01	cents
02	JLT
03	nickel
04	Done:
05	SUB
06	nickel
07	STOR
08	cents
09	LOAD
0A	count
0B	INC
0C	STOR
0D	count
0E	JMP
0F	Again:
10	HALT
11	
12	
13	
14	
15	
16	cents 4
17	nickel 5
18	count 2
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

Explanation

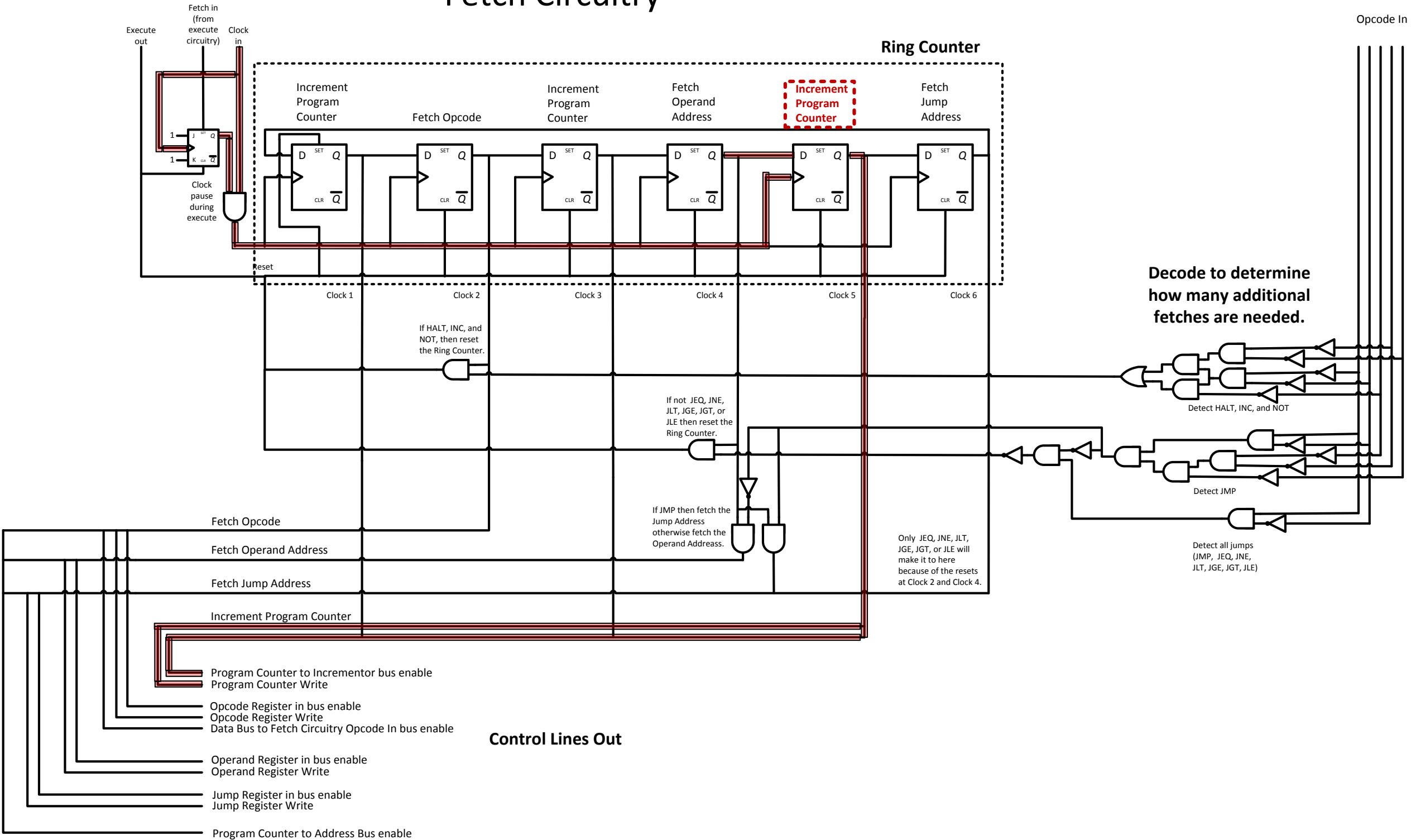
copy the value at 'cents' to the Accumulator
if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'
subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator
copy the value in the Accumulator to 'cents'
copy the value at address 'count' to the Accumulator
add 1 to the Accumulator
copy the value in the Accumulator to 'count'
unconditional jump to the address 'Again:'
stop the processor - end of program
variable -- the name 'cents' is the address and 14 is the value
variable -- the name 'cents' is the address and 5 is the value
variable -- the name 'count' is the address and 0 is the value
unconditional jump to the address 'Again:'

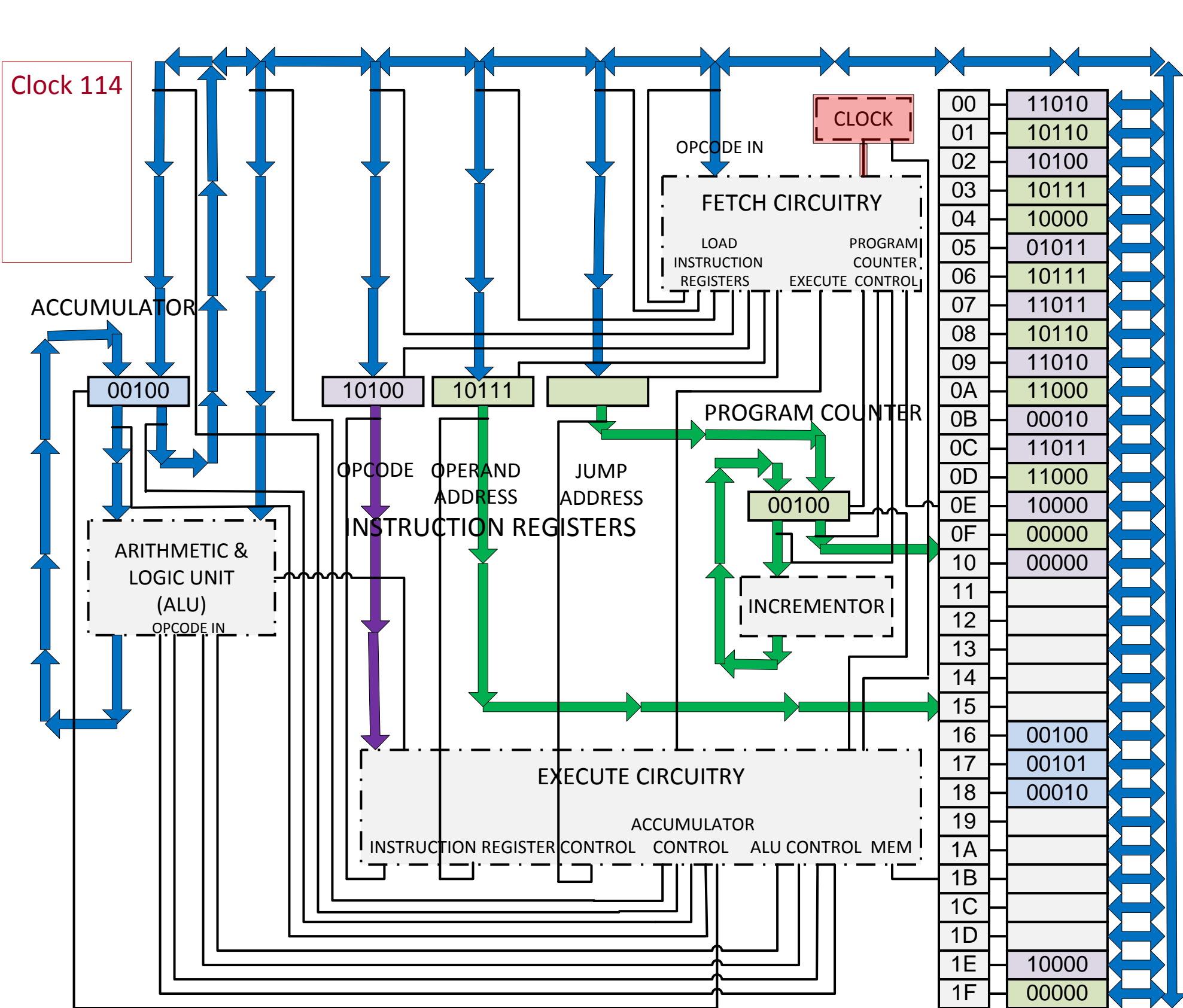
Hex

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



Fetch Circuitry





Assembly Language
Addresses: Program:

00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	00100
17	00101
18	00010
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

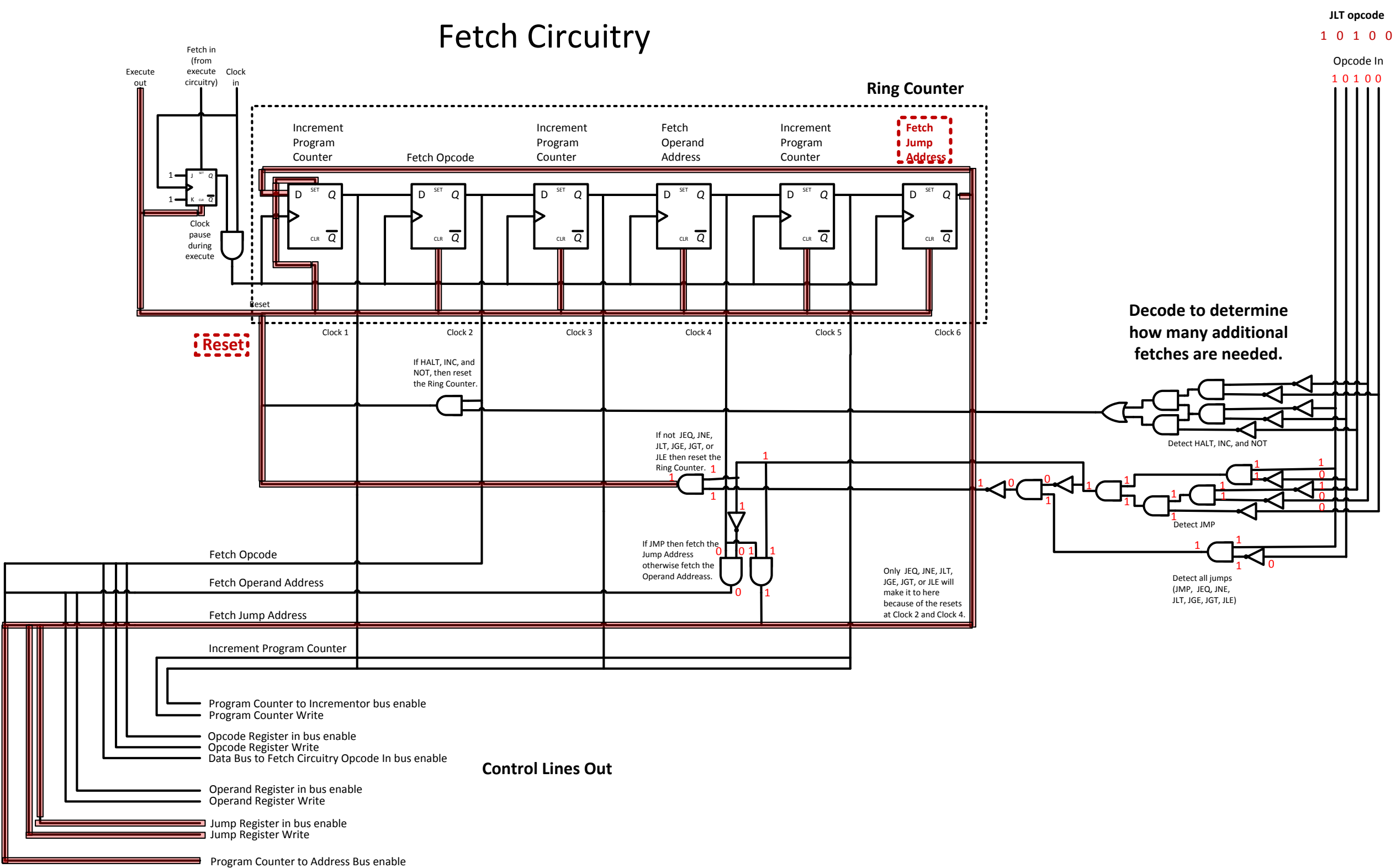
Explanation

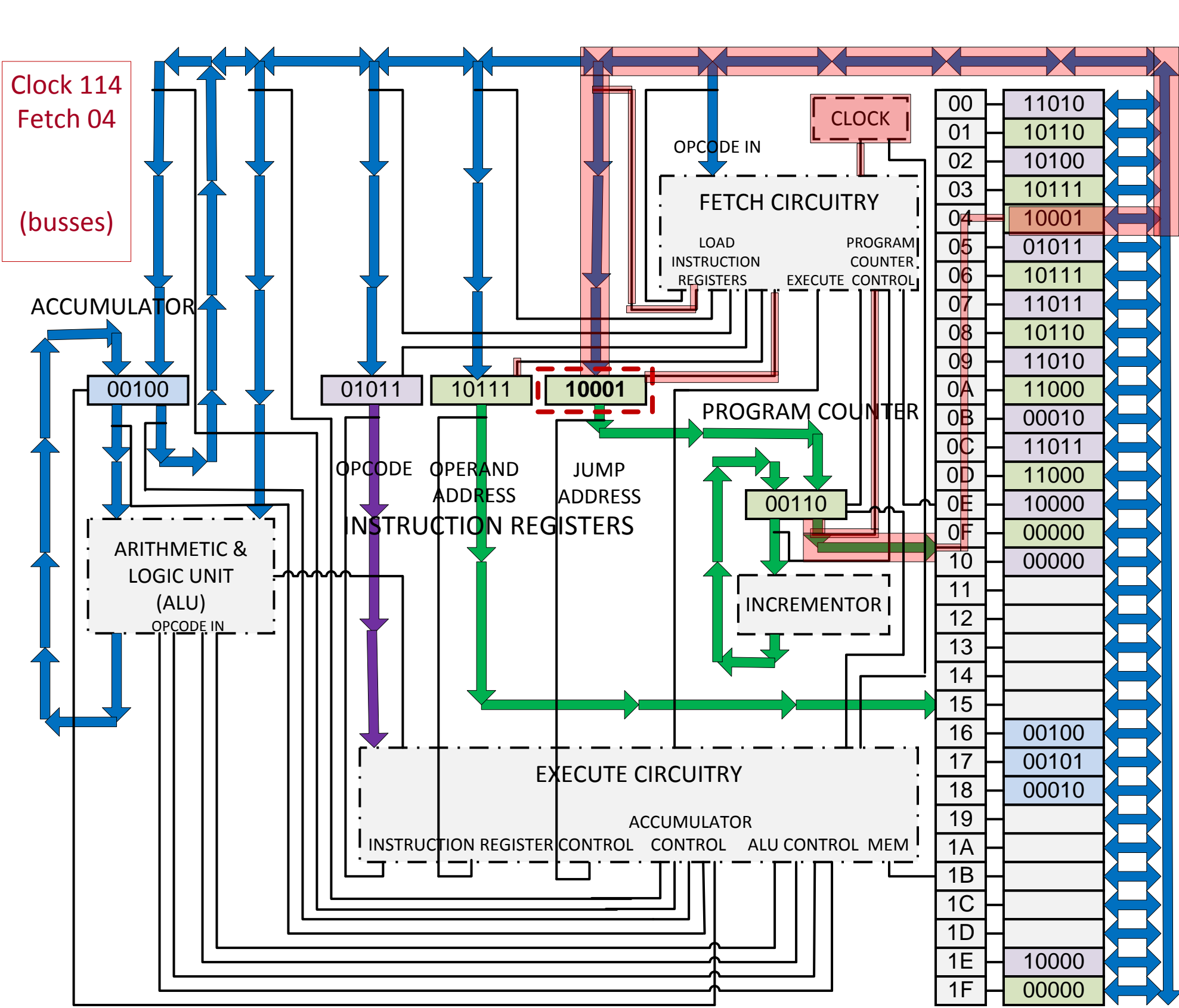
copy the value at 'cents' to the Accumulator
if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'
subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator
copy the value in the Accumulator to 'cents'
copy the value at address 'count' to the Accumulator
add 1 to the Accumulator
copy the value in the Accumulator to 'count'
unconditional jump to the address 'Again:'
stop the processor – end of program
variable -- the name 'cents' is the address and 14 is the value
variable -- the name 'cents' is the address and 5 is the value
variable -- the name 'count' is the address and 0 is the value
unconditional jump to the address 'Again:'

Hex

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

Fetch Circuitry





Assembly Language Addresses: Program:

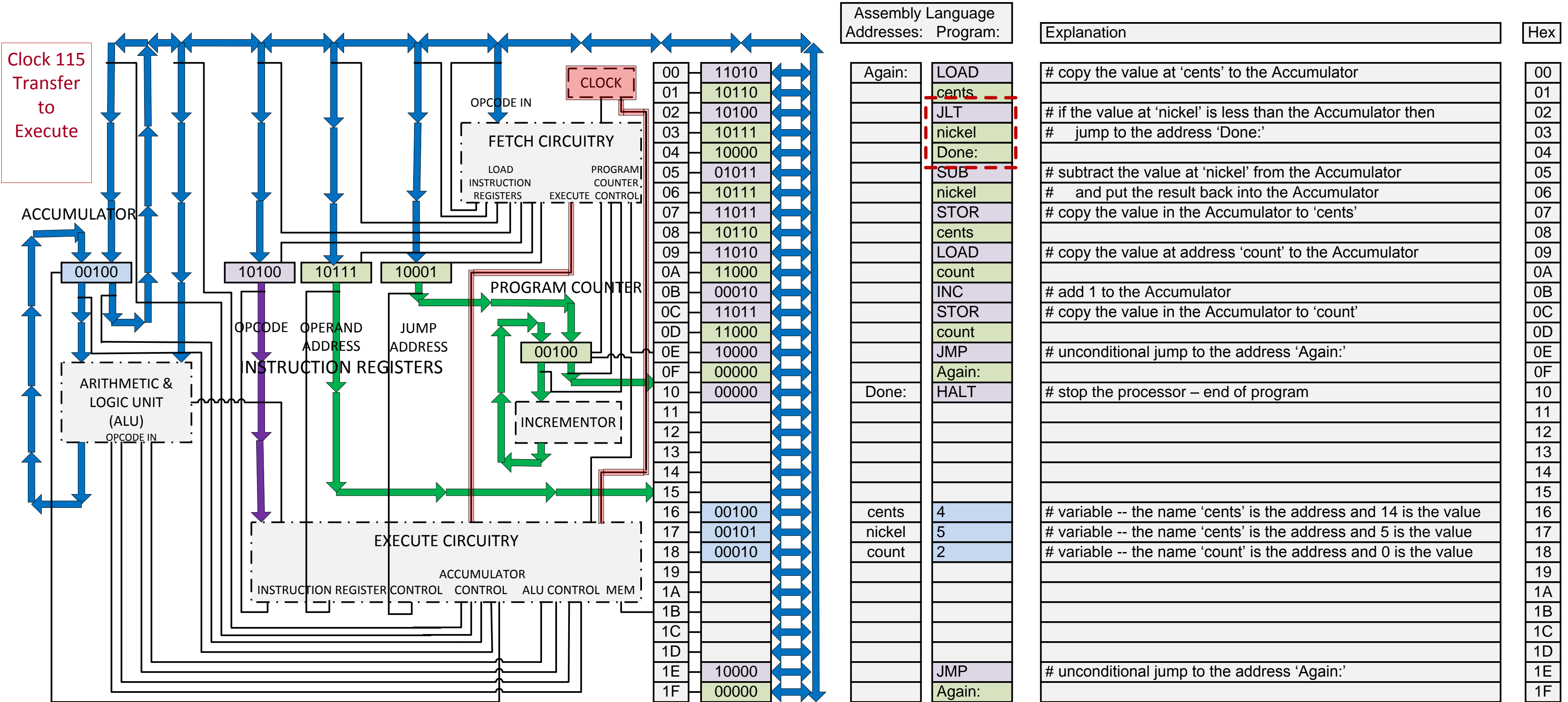
00	11010
01	10110
02	10100
03	10111
04	10001
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	00100
17	00101
18	00010
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

Explanation

copy the value at 'cents' to the Accumulator
if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'
subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator
copy the value in the Accumulator to 'cents'
copy the value at address 'count' to the Accumulator
add 1 to the Accumulator
copy the value in the Accumulator to 'count'
unconditional jump to the address 'Again:'
stop the processor – end of program
variable -- the name 'cents' is the address and 14 is the value
variable -- the name 'cents' is the address and 5 is the value
variable -- the name 'count' is the address and 0 is the value
unconditional jump to the address 'Again:'

Hex

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



Clock 115
Transfer
to
Execute

Assembly Language
Addresses: Program:

Explanation

Hex

ACCUMULATOR

CLOCK

FETCH CIRCUITRY

LOAD
INSTRUCTION
REGISTERS

PROGRAM
COUNTER,
EXECUTE CONTROL

PROGRAM COUNTER

INSTRUCTION REGISTERS

JUMP
ADDRESS

INCREMENTOR

EXECUTE CIRCUITRY

ACCUMULATOR

INSTRUCTION REGISTER CONTROL
ALU CONTROL MEM

ARITHMETIC &
LOGIC UNIT
(ALU)

OPCODE
OPERAND
ADDRESS

00100

10100

10111

10001

00100

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

11010
10110
10100
10111
10000
01011
10111
11011
10110
11010
11000
00010
11011
11000
10000
00000
00000
00100
00101
00010

10000
00000

Again:

Done:

cents
nickel
count

JMP
Again:

copy the value at 'cents' to the Accumulator

if the value at 'nickel' is less than the Accumulator then
jump to the address 'Done:'

subtract the value at 'nickel' from the Accumulator
and put the result back into the Accumulator
copy the value in the Accumulator to 'cents'

copy the value at address 'count' to the Accumulator

add 1 to the Accumulator
copy the value in the Accumulator to 'count'

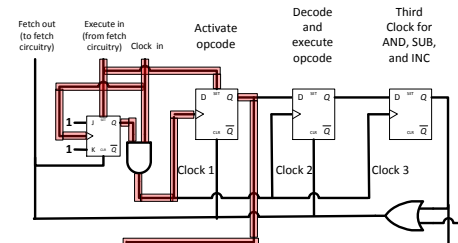
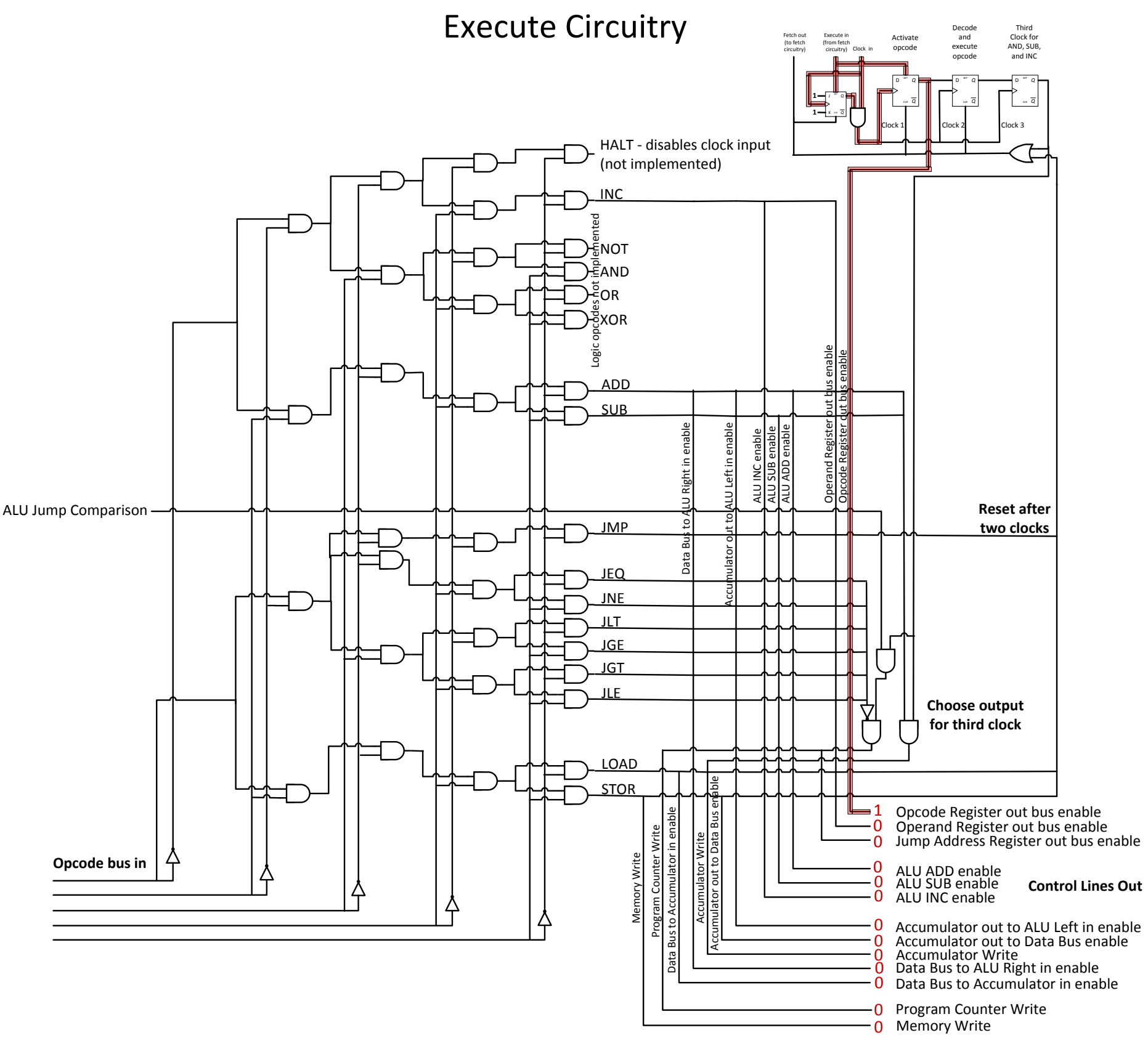
unconditional jump to the address 'Again:'

stop the processor – end of program

variable -- the name 'cents' is the address and 14 is the value
variable -- the name 'cents' is the address and 5 is the value
variable -- the name 'count' is the address and 0 is the value

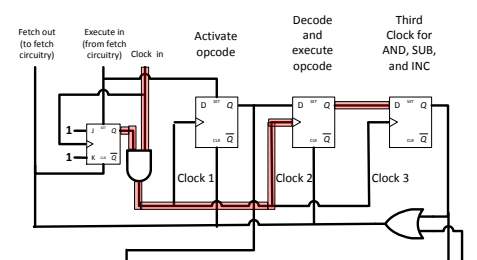
unconditional jump to the address 'Again:'

Execute Circuitry



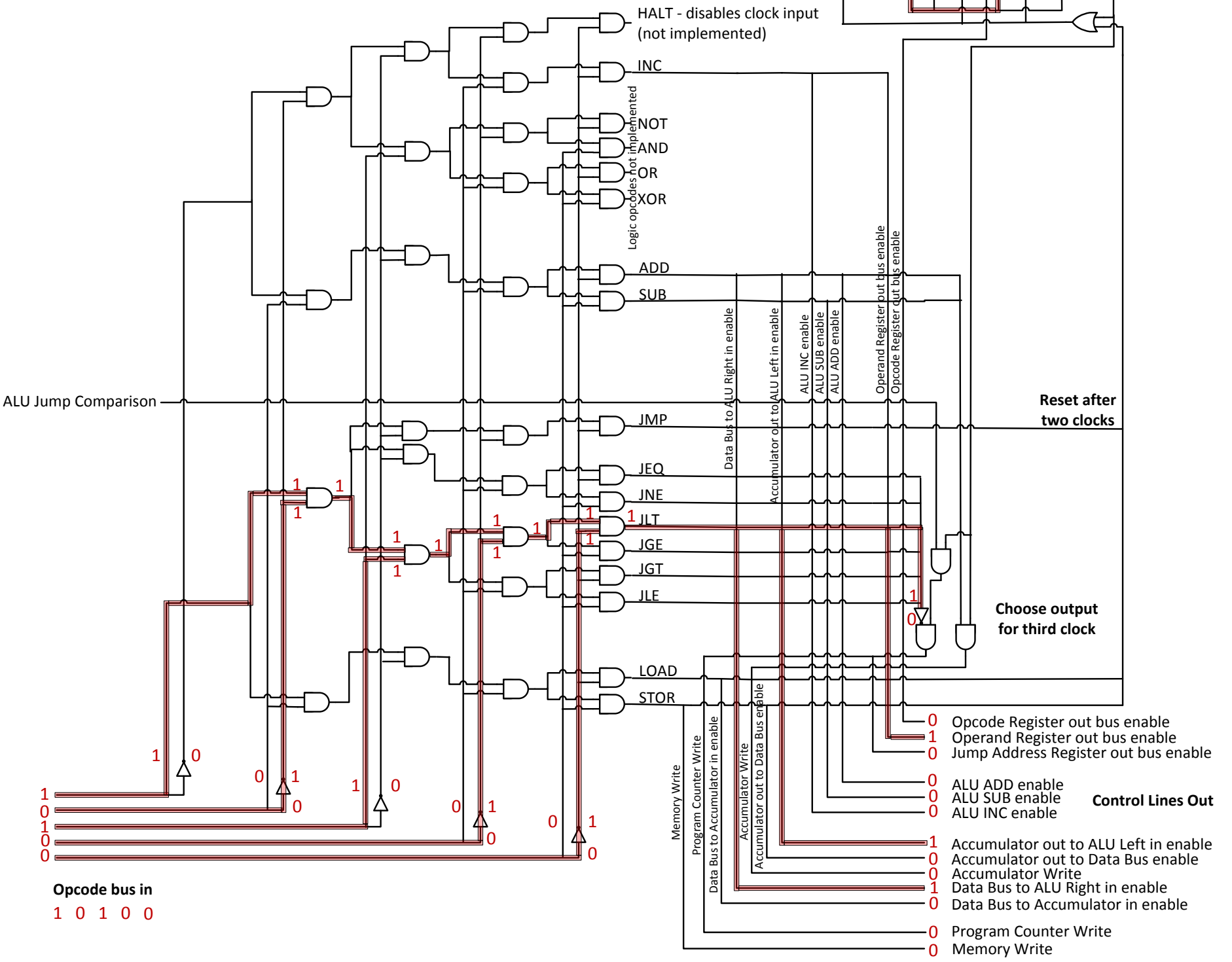
- Control Lines Out**
- 1 Opcode Register out bus enable
 - 0 Operand Register out bus enable
 - 0 Jump Address Register out bus enable
 - 0 ALU ADD enable
 - 0 ALU SUB enable
 - 0 ALU INC enable
 - 0 Accumulator out to ALU Left in enable
 - 0 Accumulator out to Data Bus enable
 - 0 Accumulator Write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

Execute Circuitry



JLT opcode

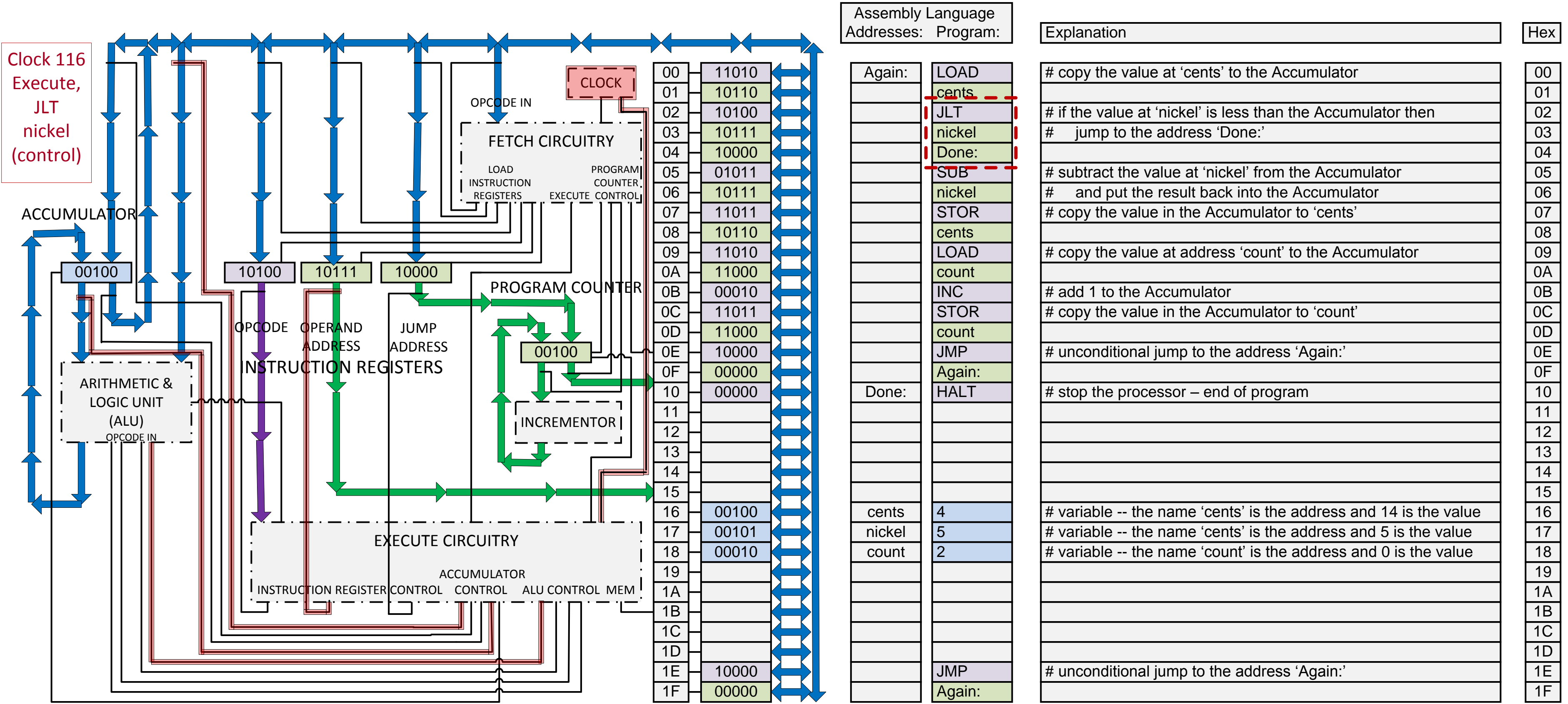
1 0 1 0 0

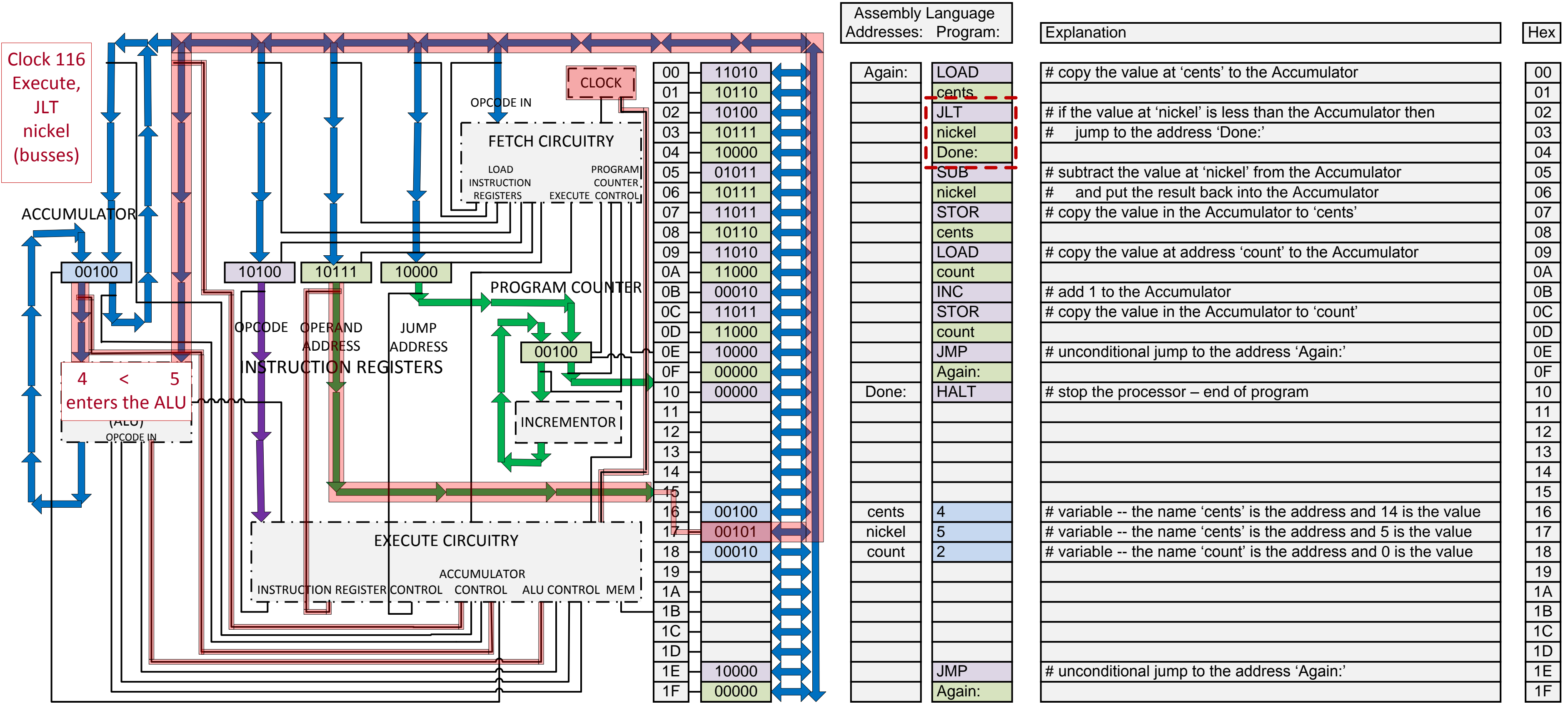


1
0
1
0
0

Opcode bus in

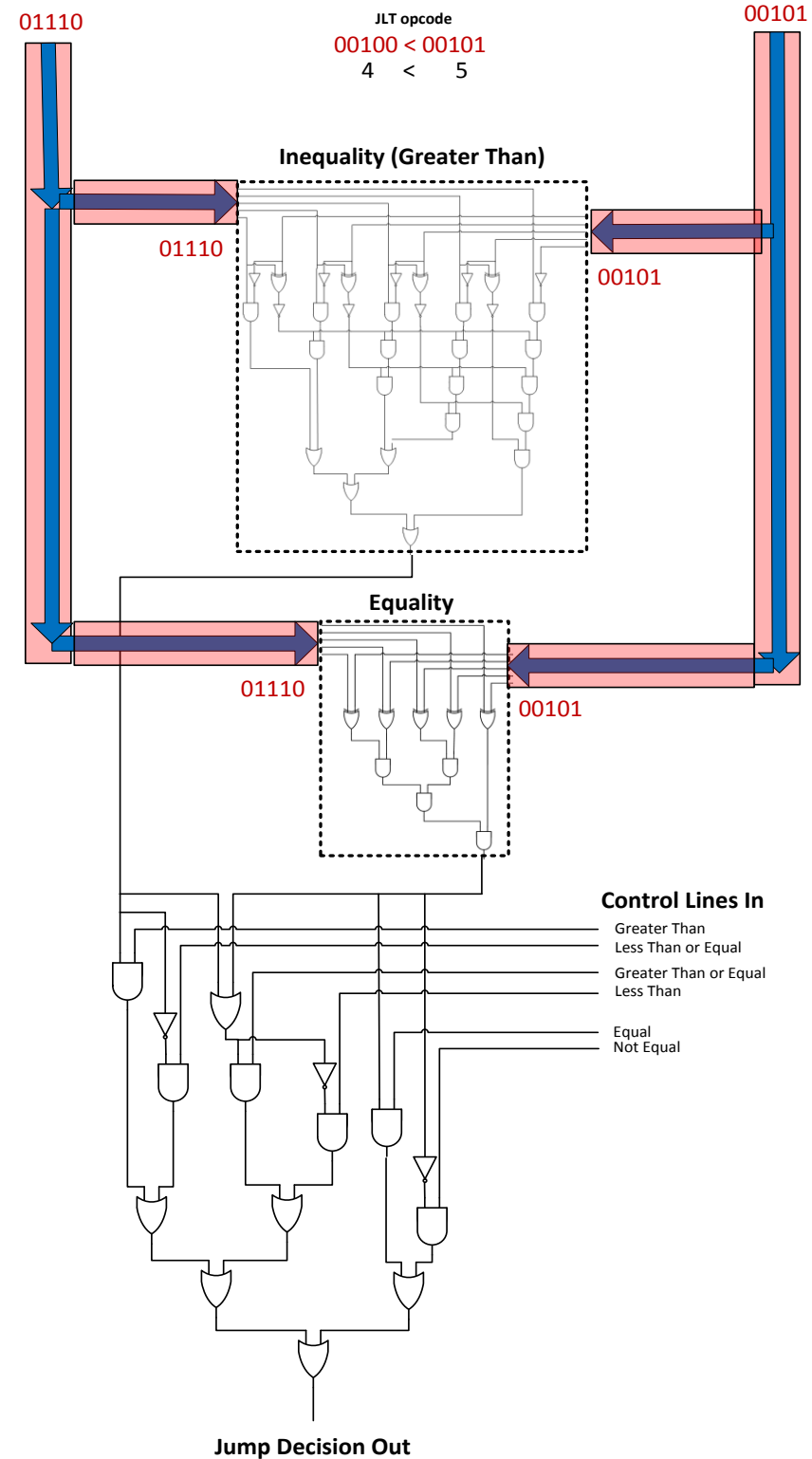
1 0 1 0 0



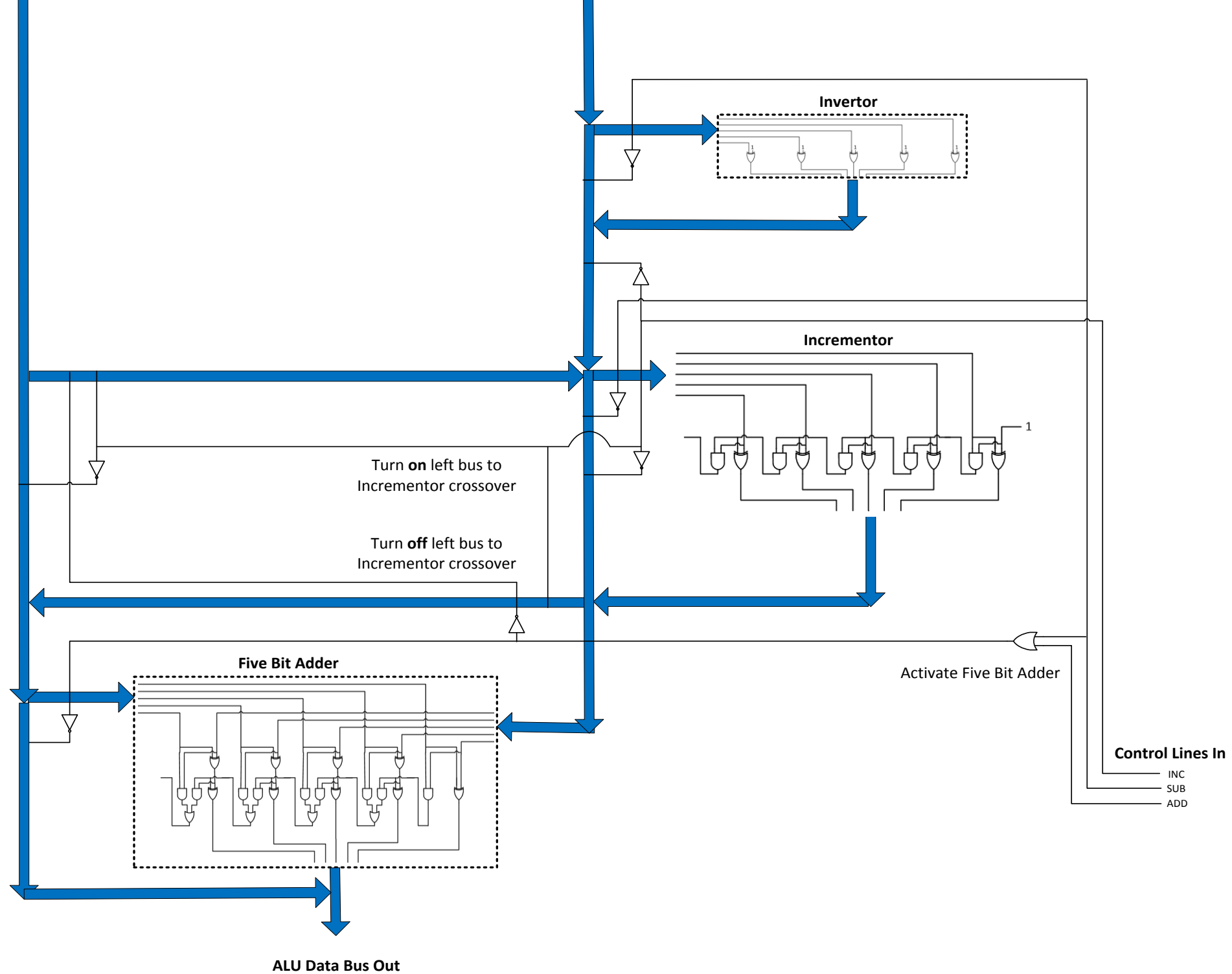


ALU (Arithmetic and Logic Unit)

Left bus into ALU **Jump Logic** Right bus into ALU

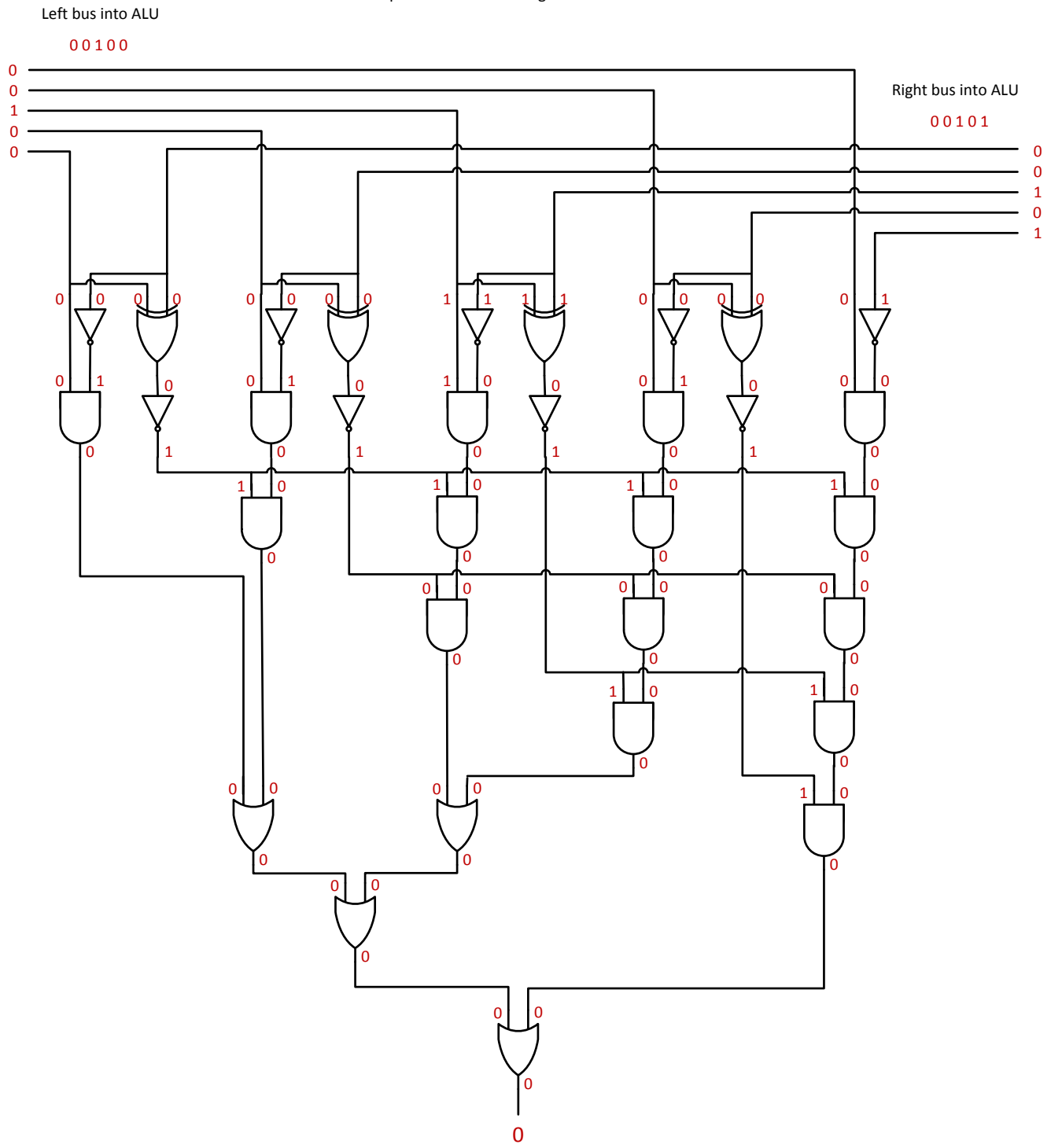


Left bus into ALU **Arithmetic** Right bus into ALU



Inequality – Left > Right

00100 > 00101
4 > 5



Output
1 if Left > Right, 0 otherwise

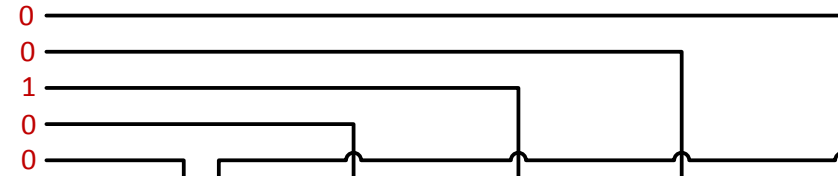
Equality

00100 = 00101
4 = 5

Inputs from Left and Right Buses into the ALU

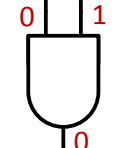
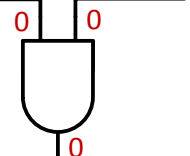
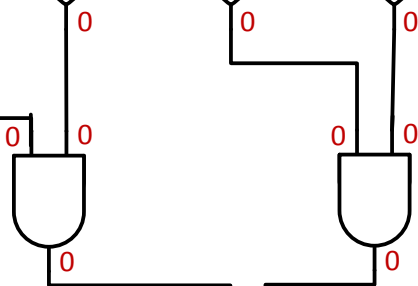
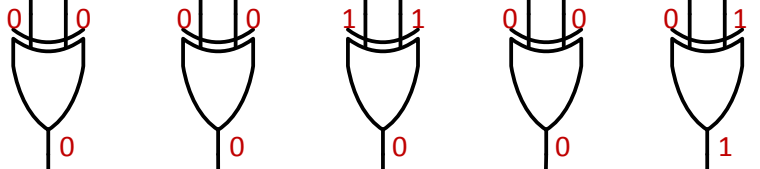
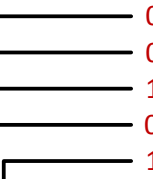
Left bus into ALU

00100



Right bus into ALU

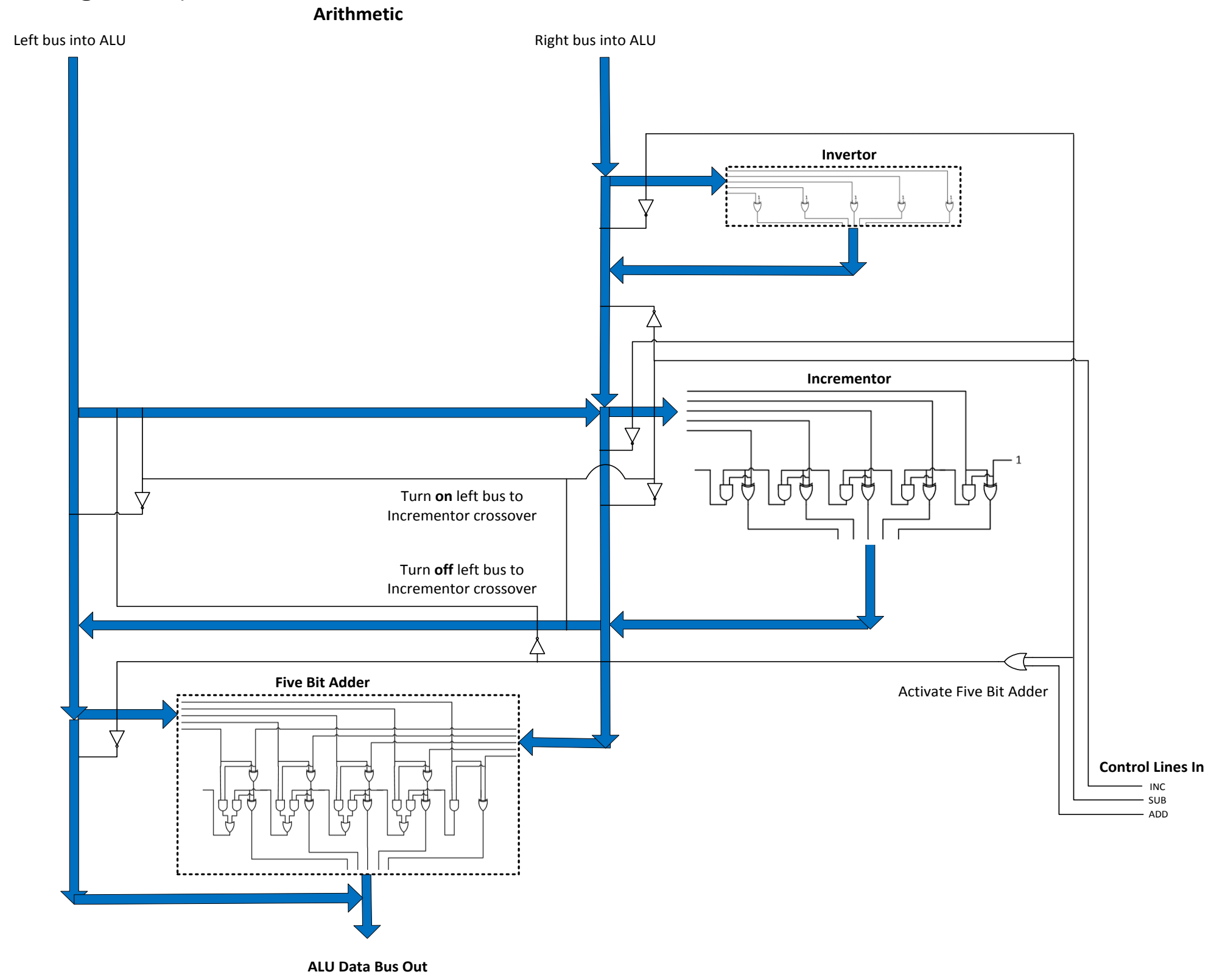
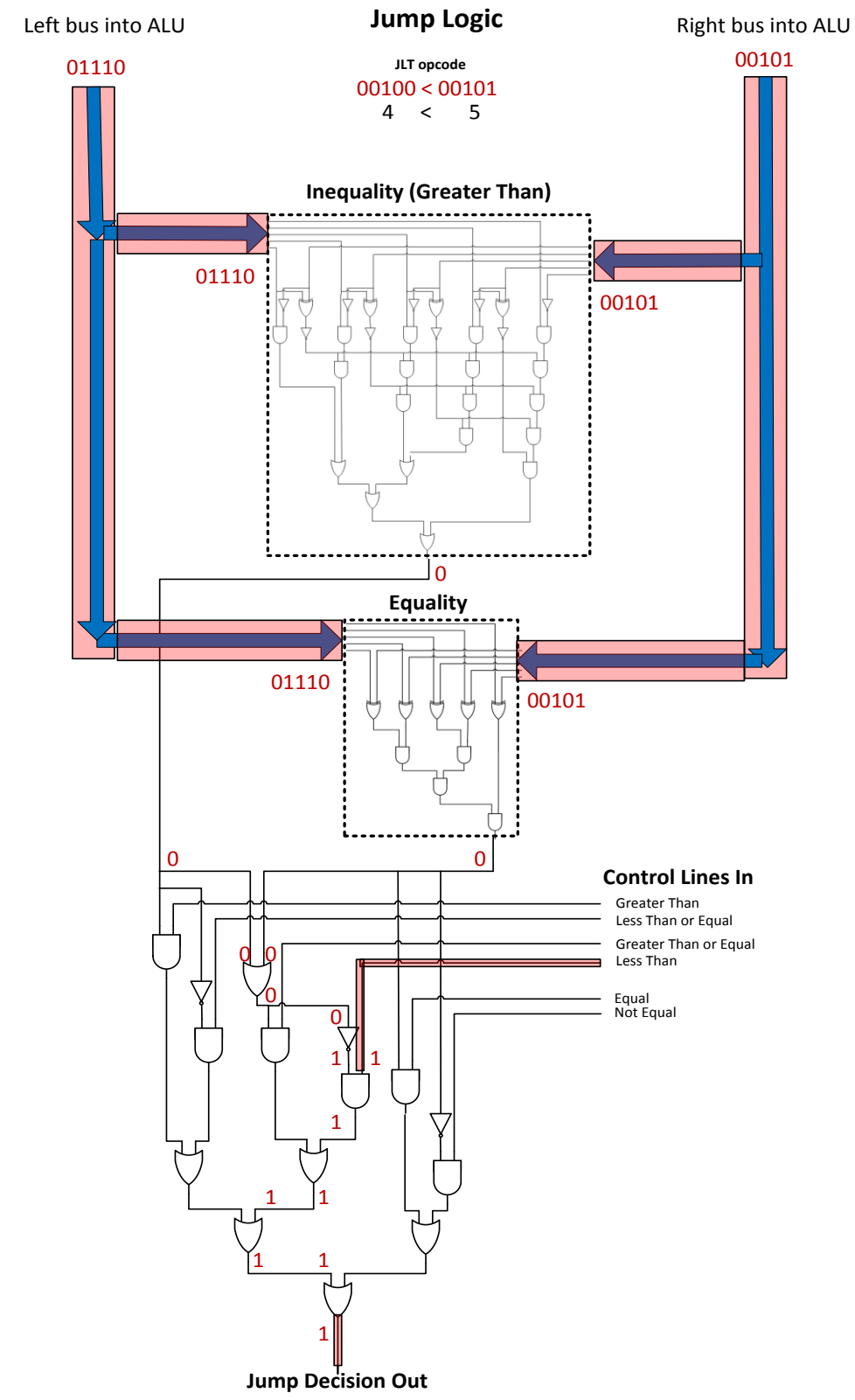
00101

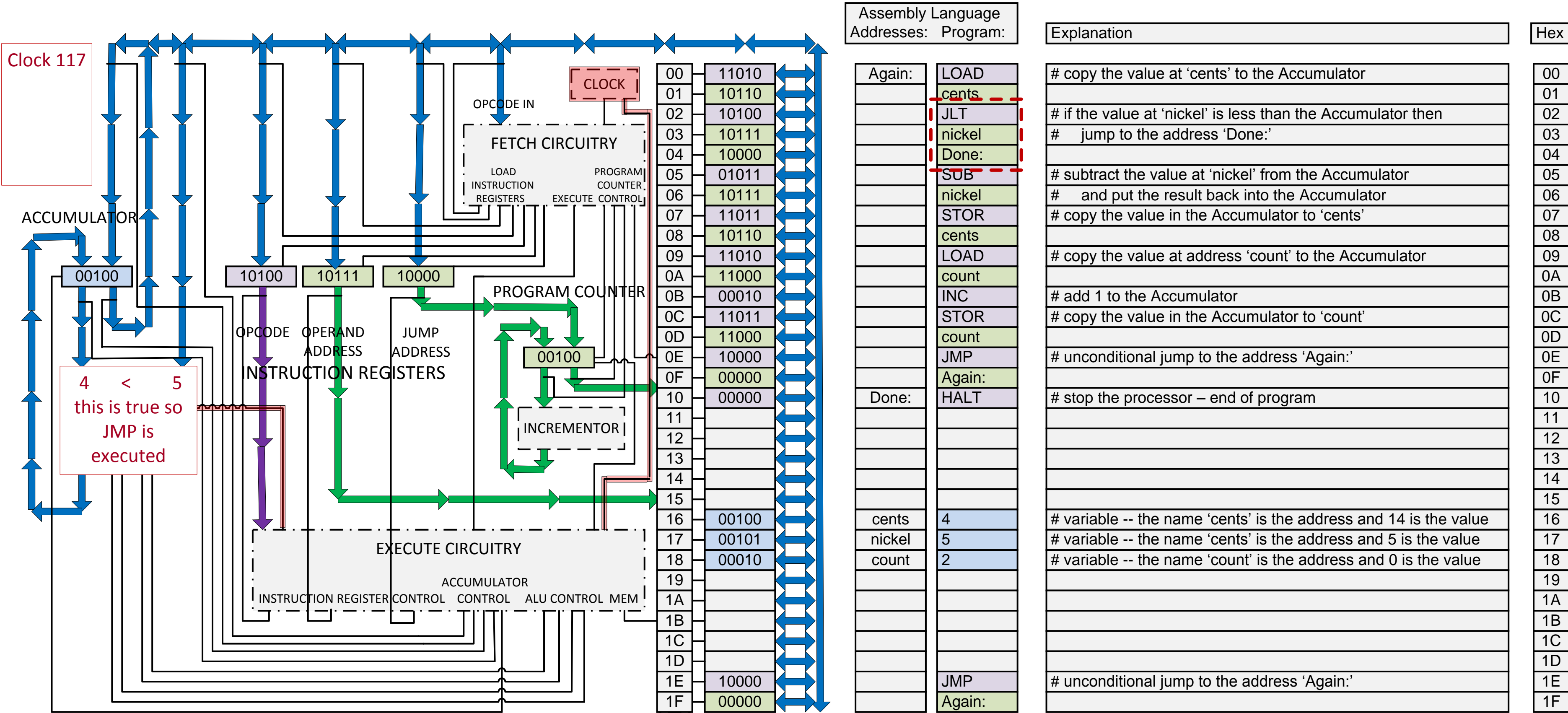


O u t p u t

1 if Left = Right, 0 otherwise

ALU (Arithmetic and Logic Unit)





4 < 5
this is true so
JMP is
executed

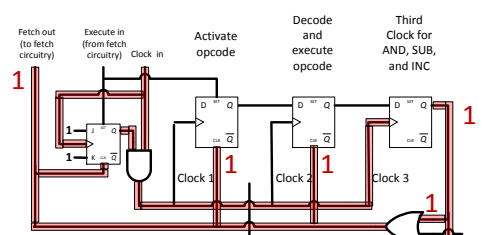
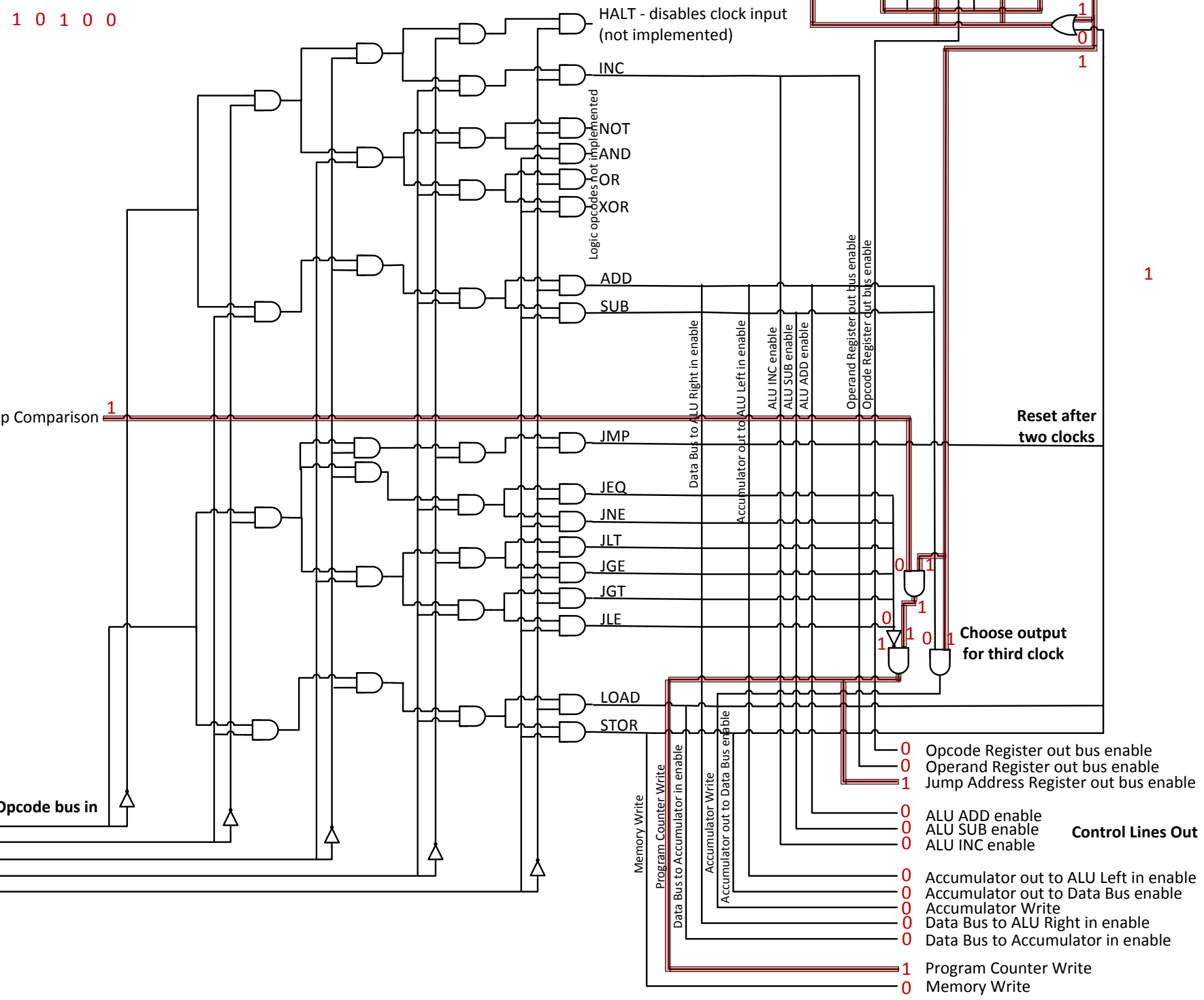
Execute Circuitry

JLT opcode

1 0 1 0 0

ALU Jump Comparison 1

Opcode bus in



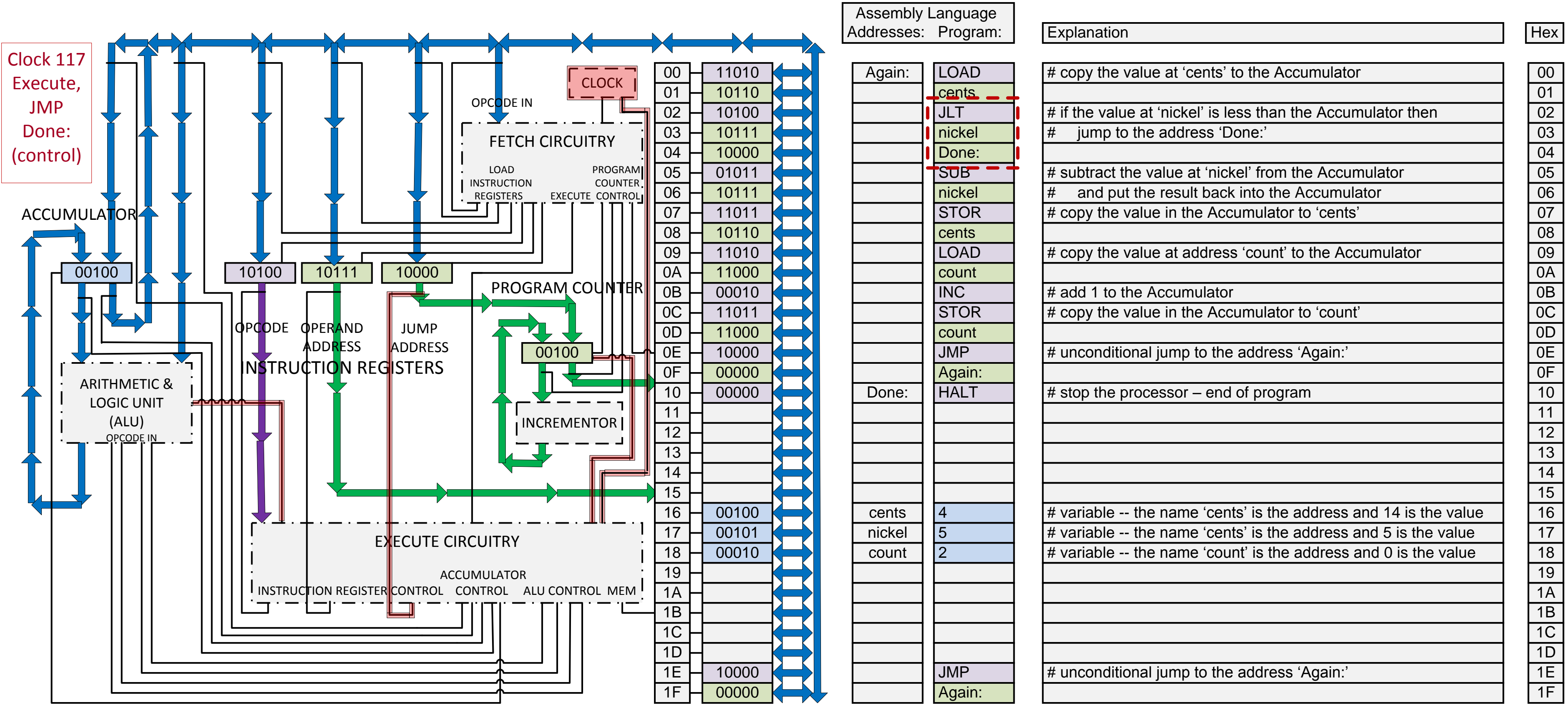
1

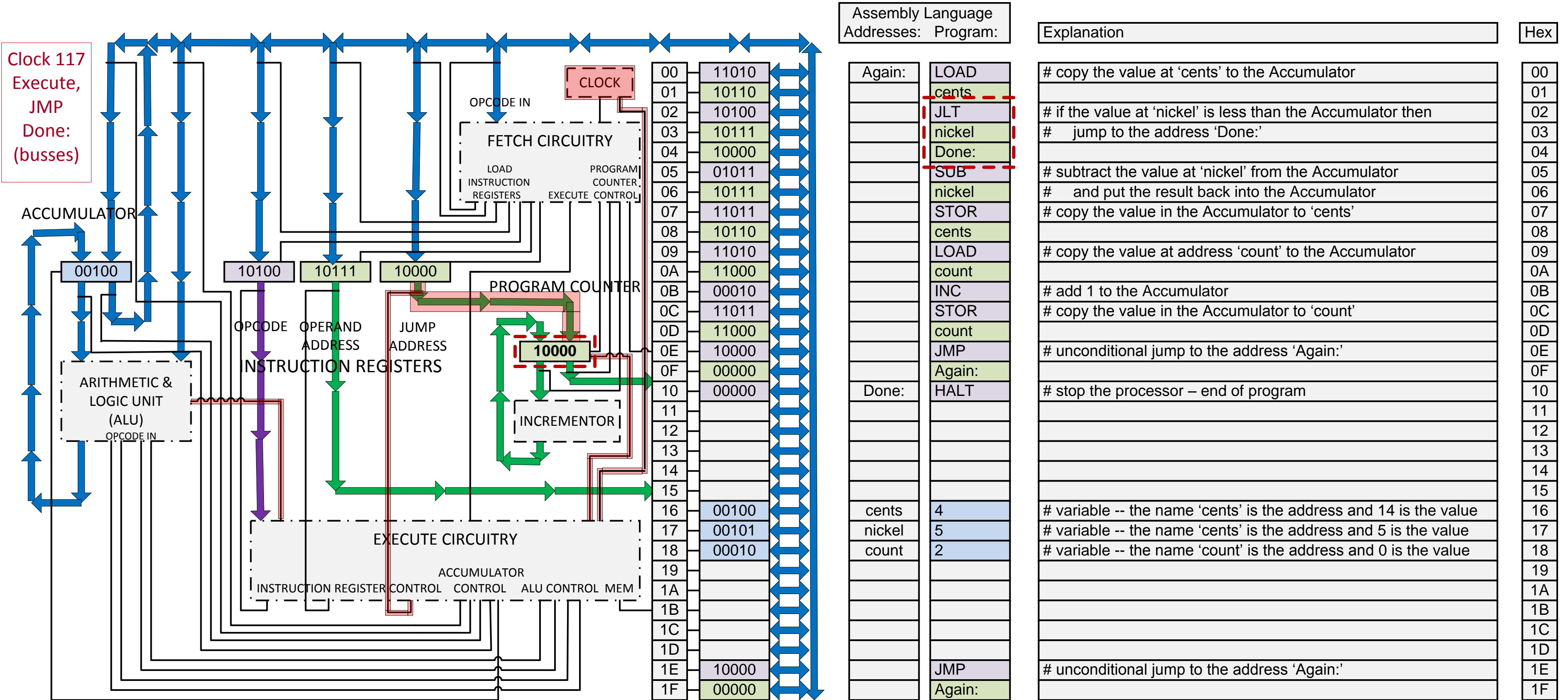
Reset after two clocks

Choose output for third clock

Control Lines Out

- 0 Opcode Register out bus enable
- 0 Operand Register out bus enable
- 1 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
- 1 Program Counter Write
- 0 Memory Write





Optional Next Presentation:
Execute each clock cycle with individual control line and bus slides

End of Presentation