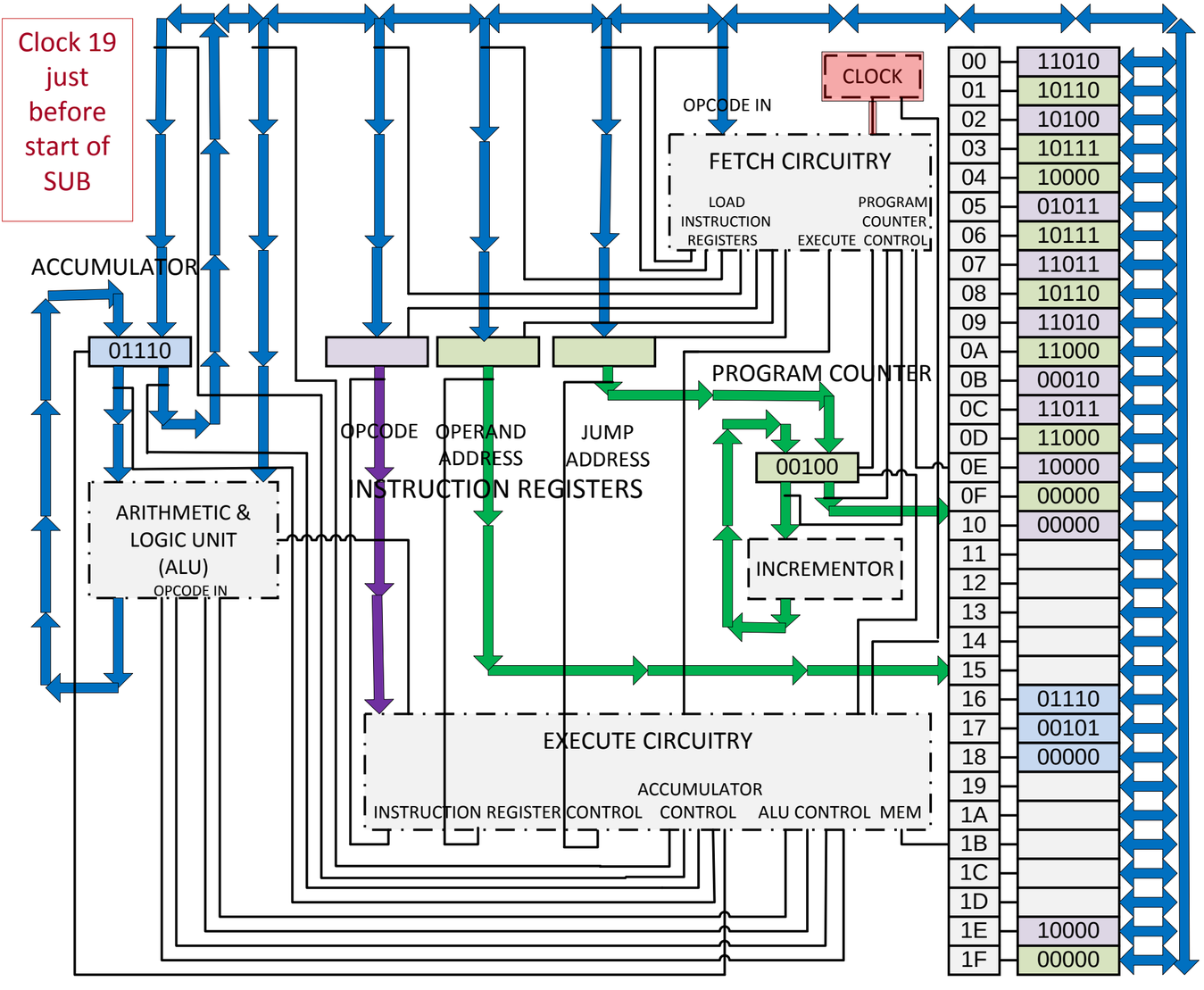
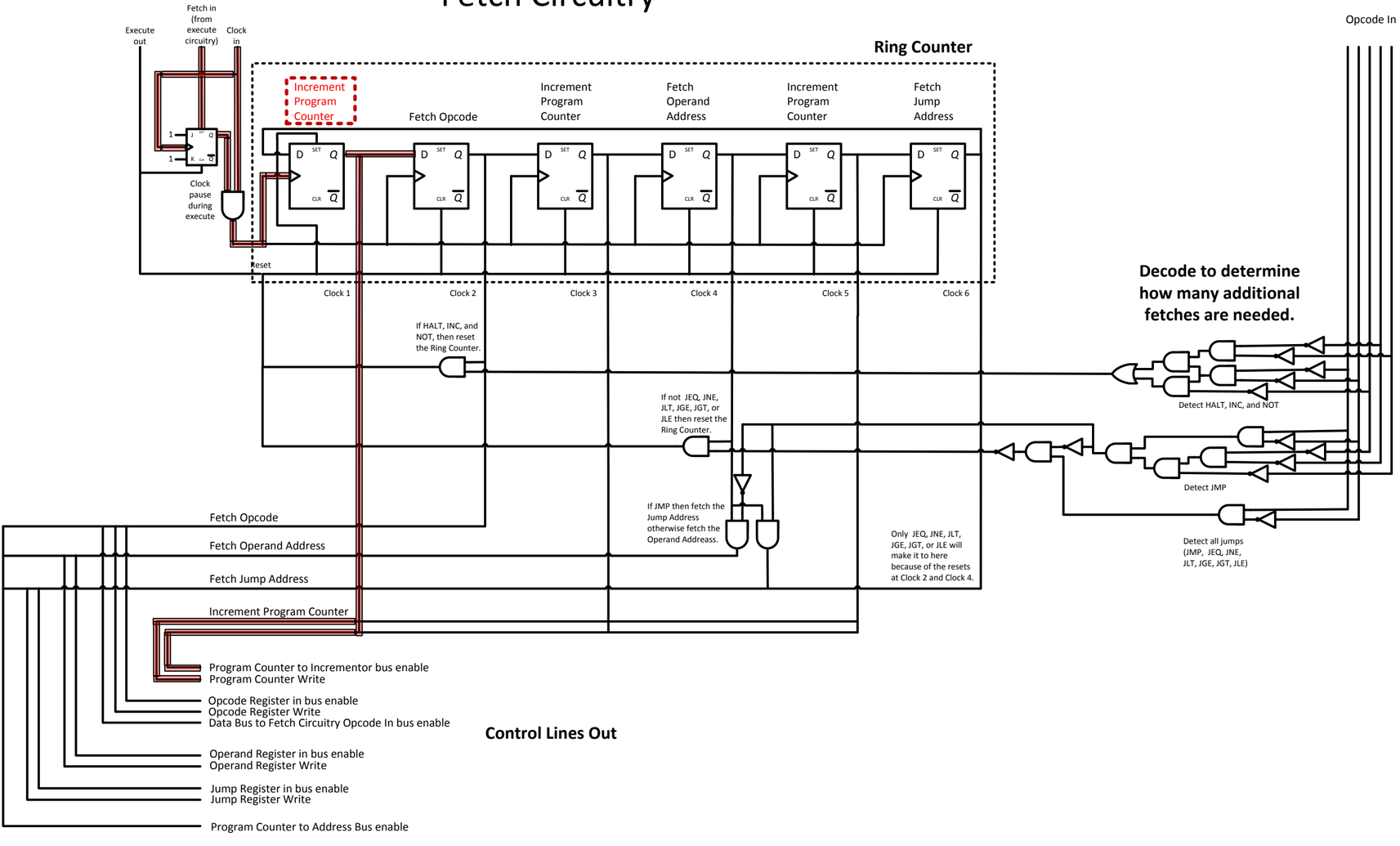
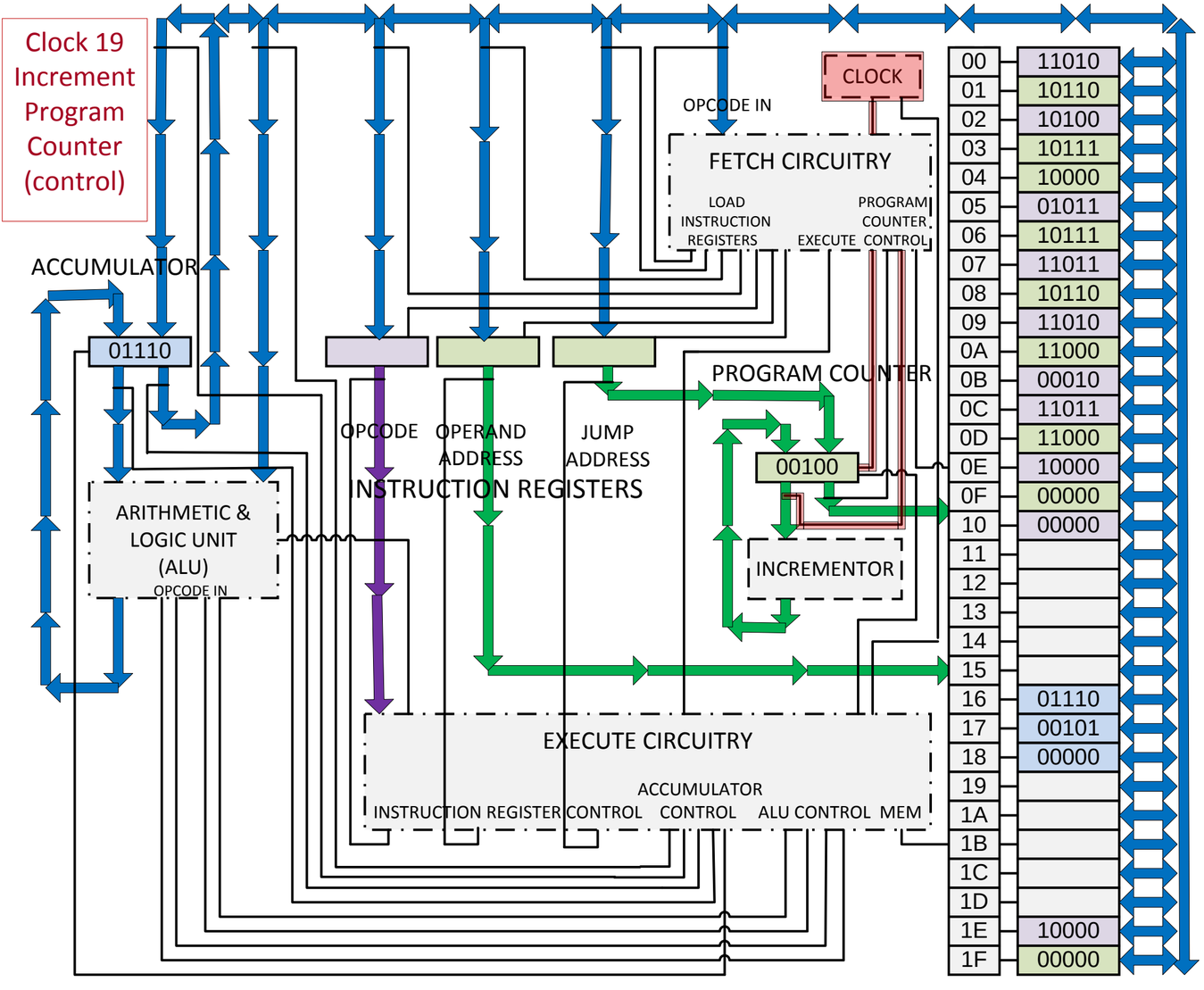


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

Fetch Circuitry

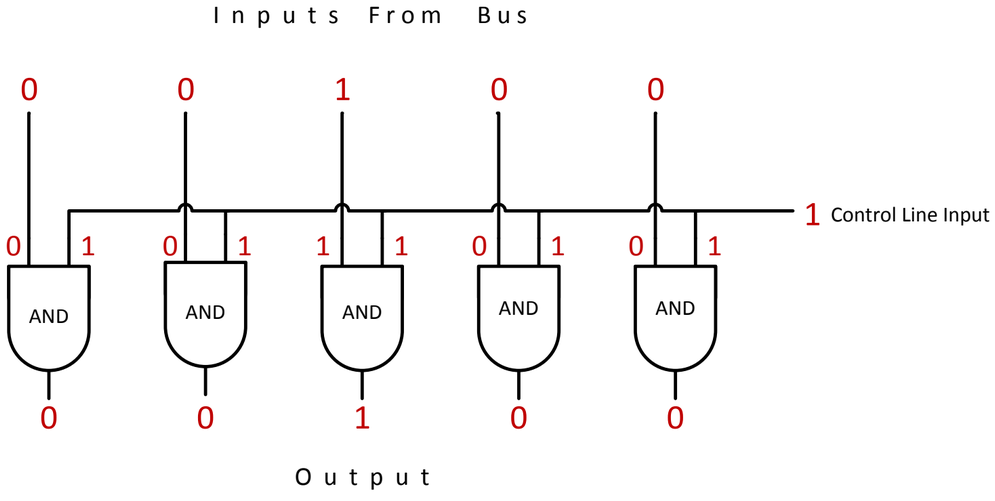




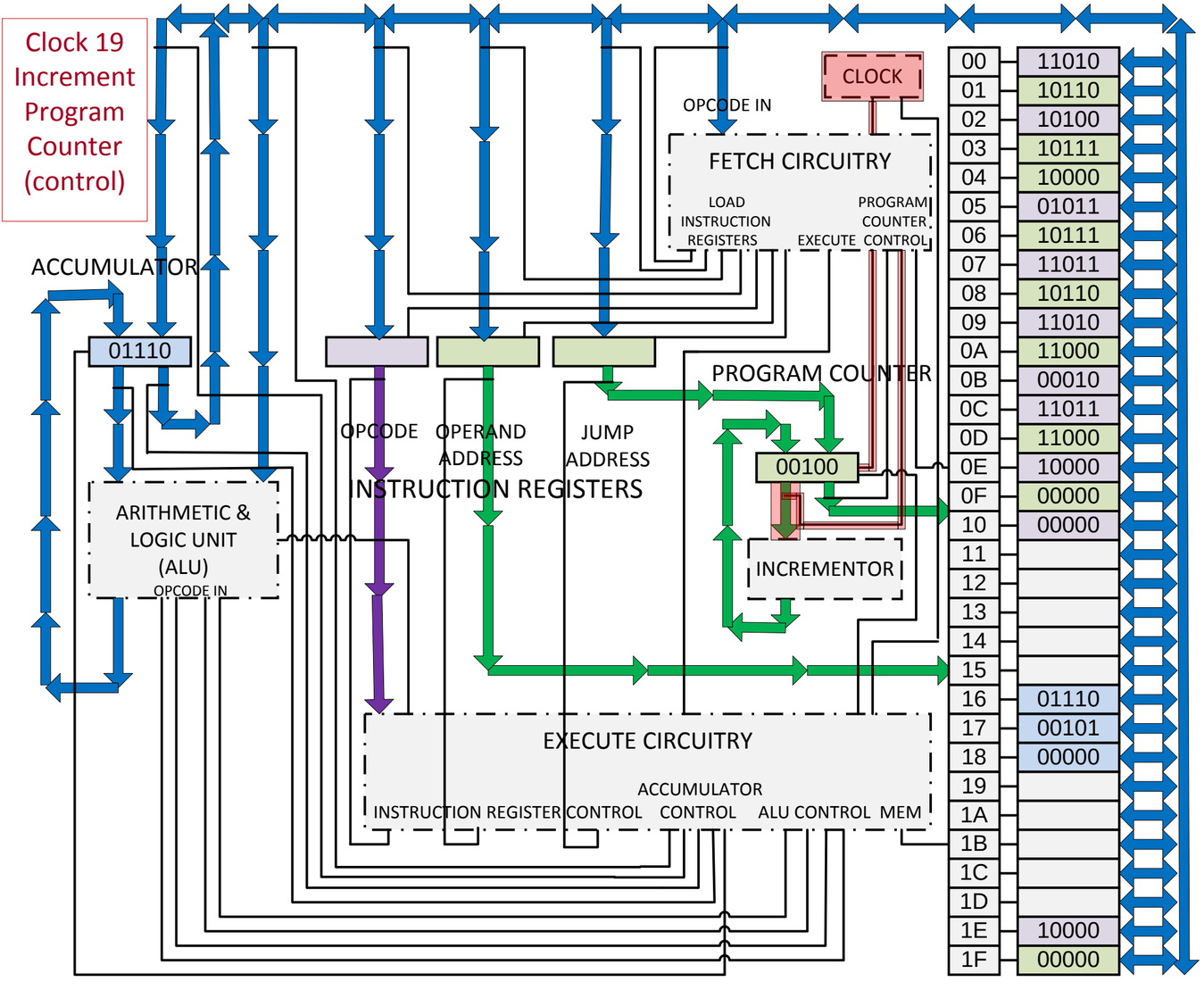
00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	01110
17	00101
18	00000
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

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

Bus Control



All output is 0 if Control Line is 0
Identical to Inputs From Bus if Control Line is 1



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

Incrementor

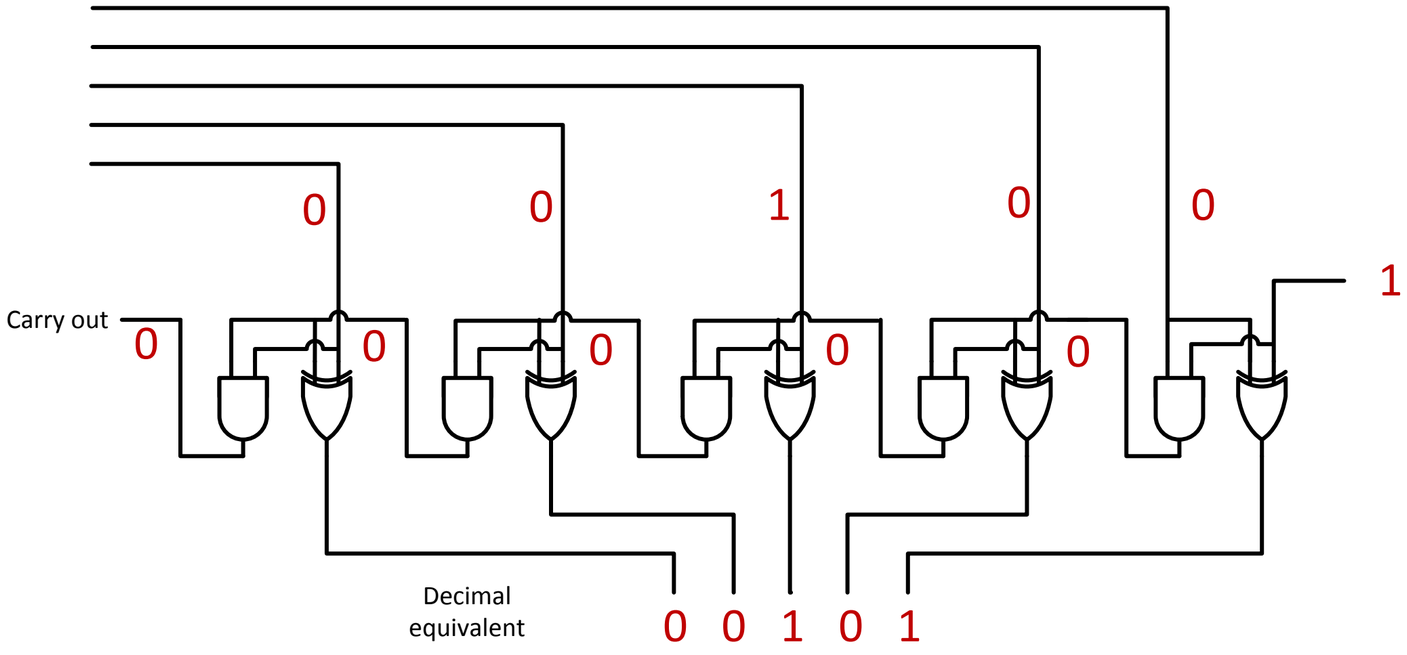
INC opcode

00100 + 00001

4 + 1

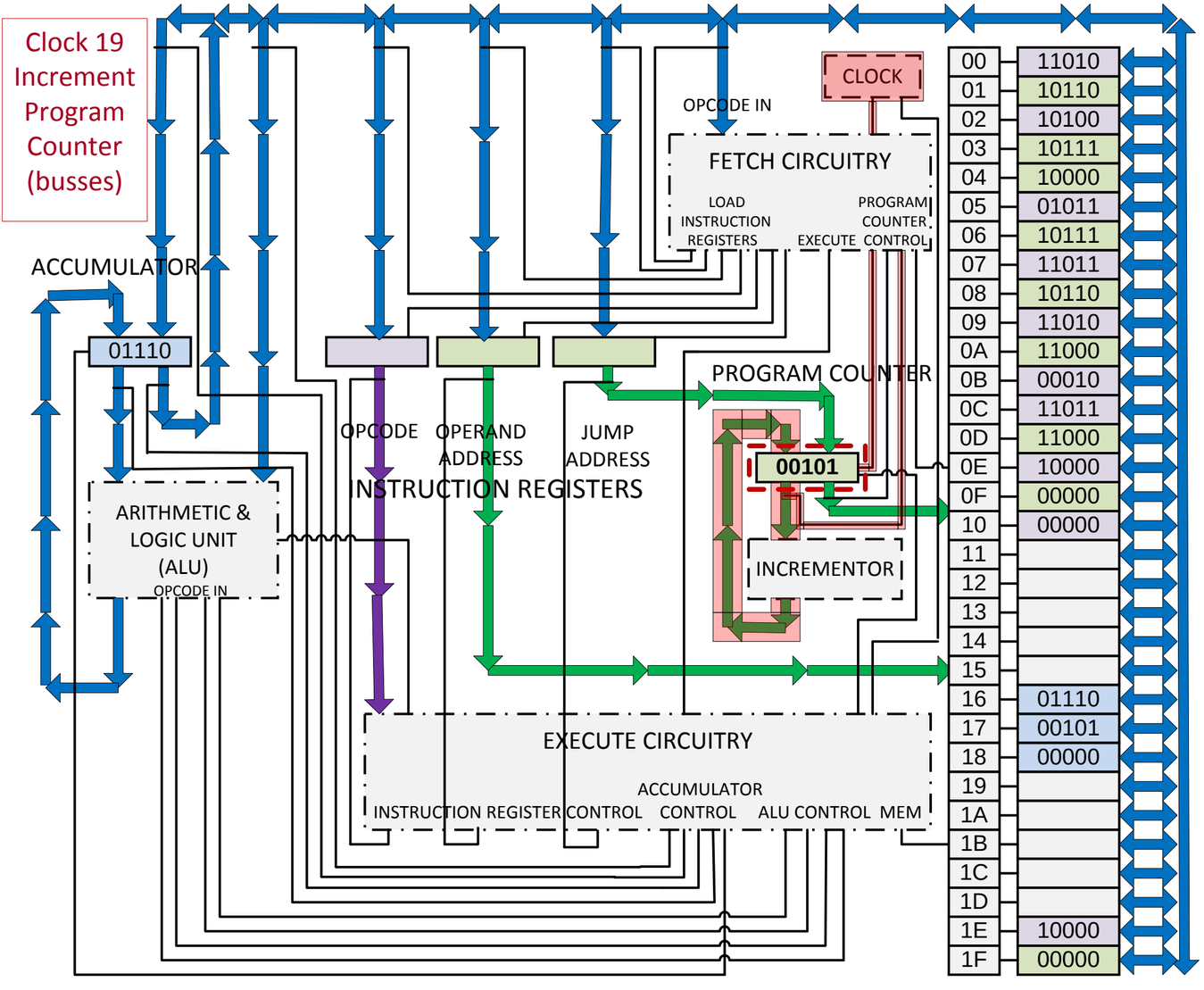
Left bus into ALU

0 0 1 0 0



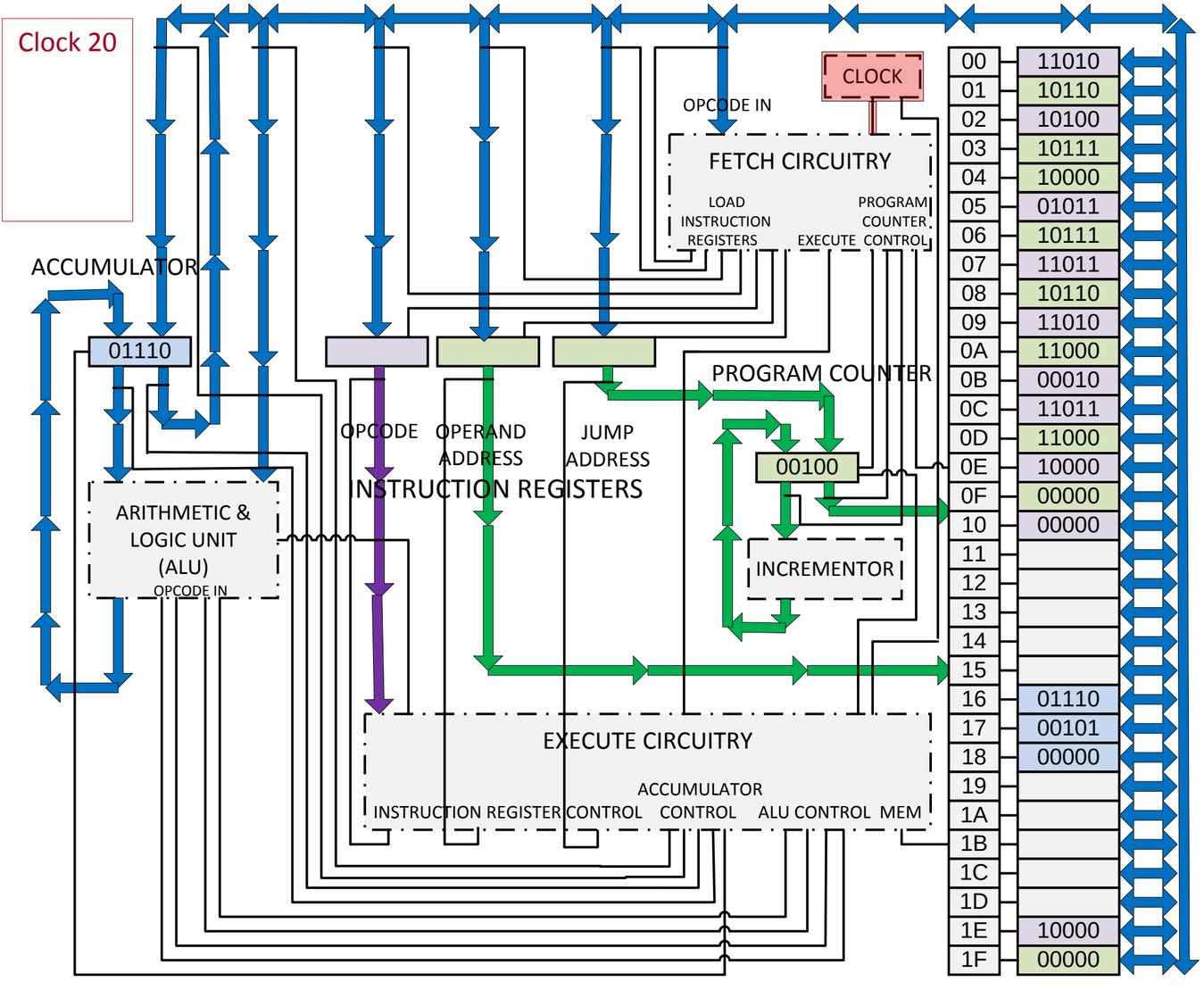
Decimal equivalent

```
  0 0 1 0 0        4
+ 0 0 0 0 1    +   1
-----------    -----
  0 0 1 0 1        5
```



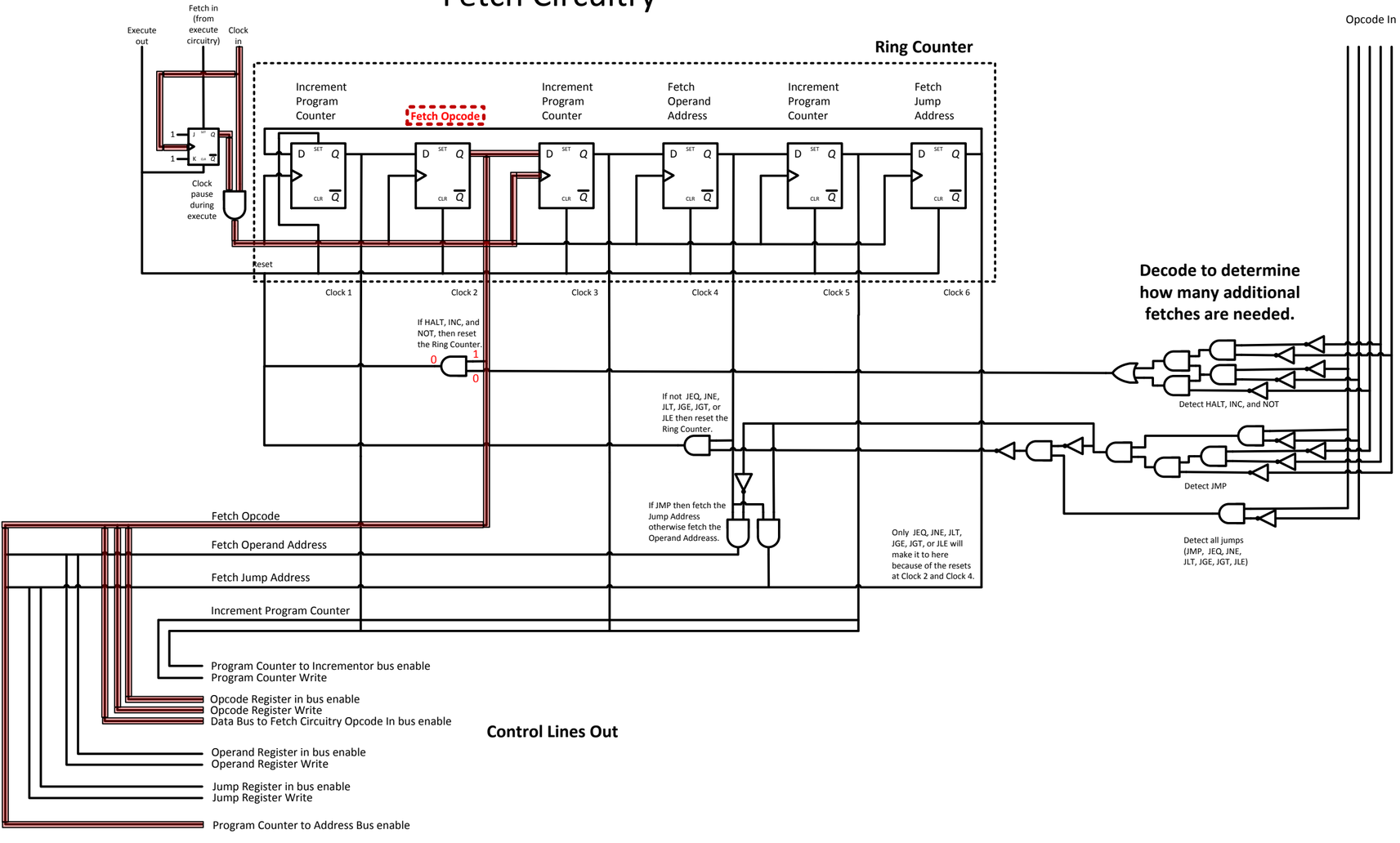
00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	01110
17	00101
18	00000
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

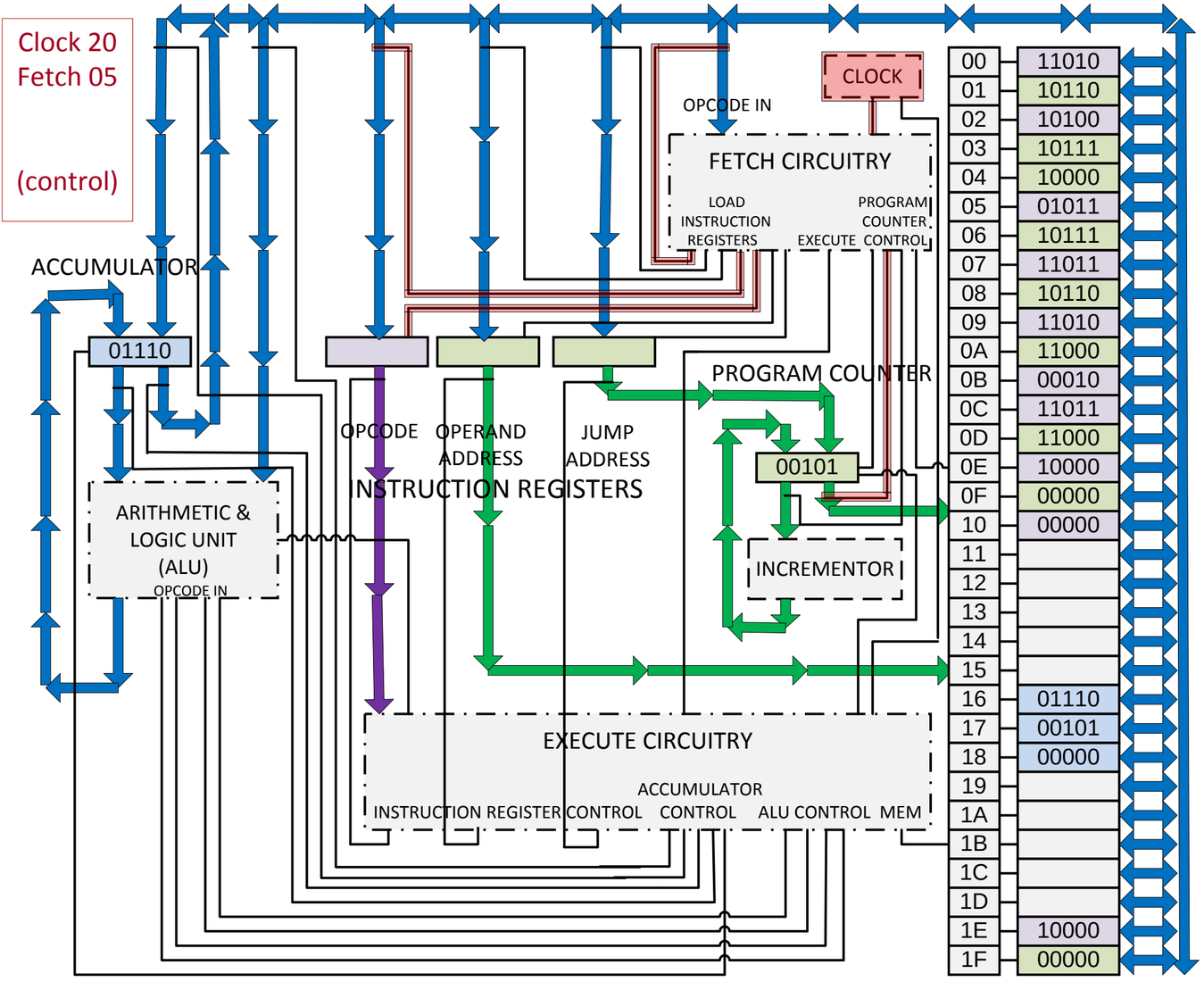
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



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

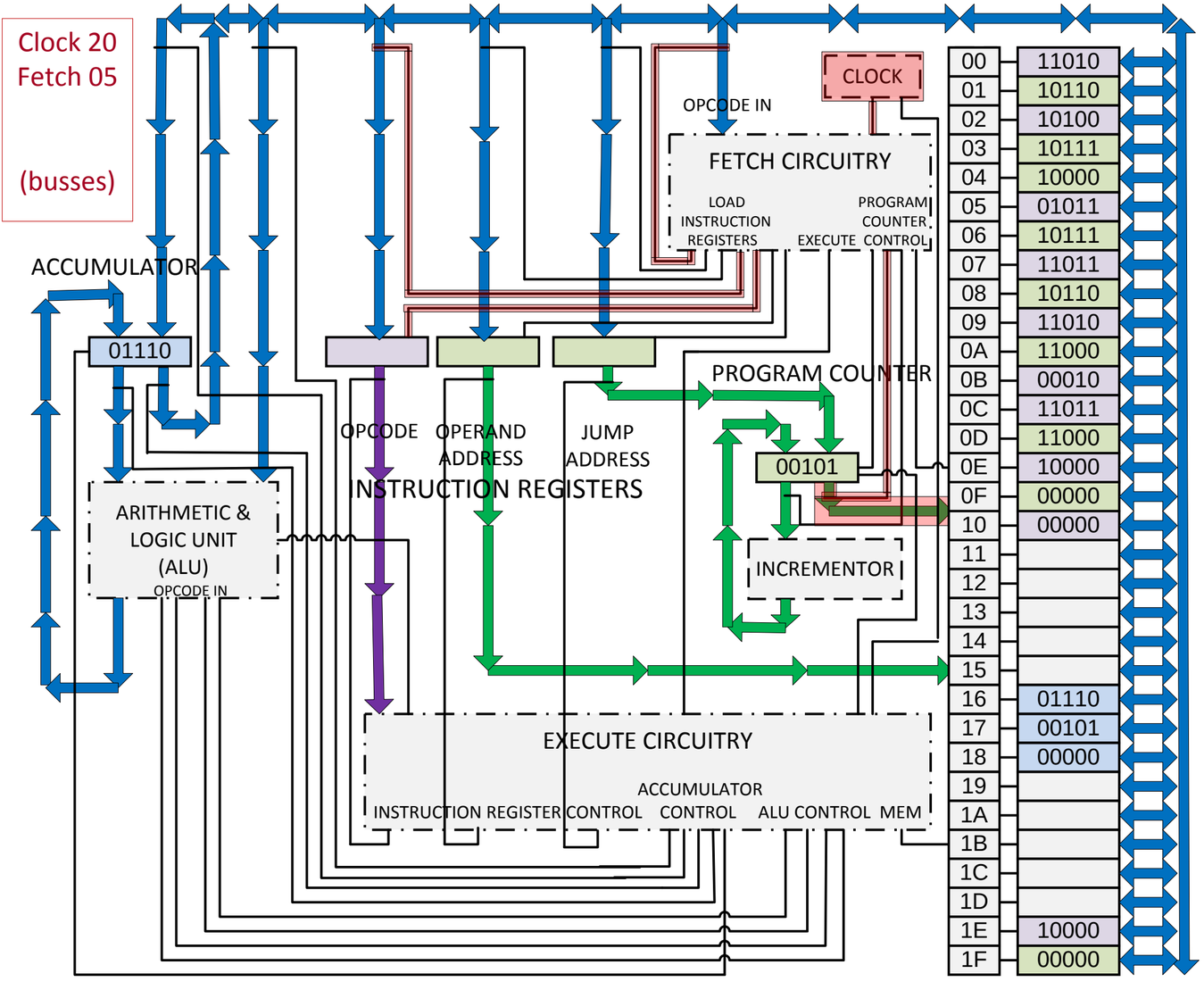
Fetch Circuitry





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	01110	
17	00101	
18	00000	
19		
1A		
1B		
1C		
1D		
1E	10000	
1F	00000	

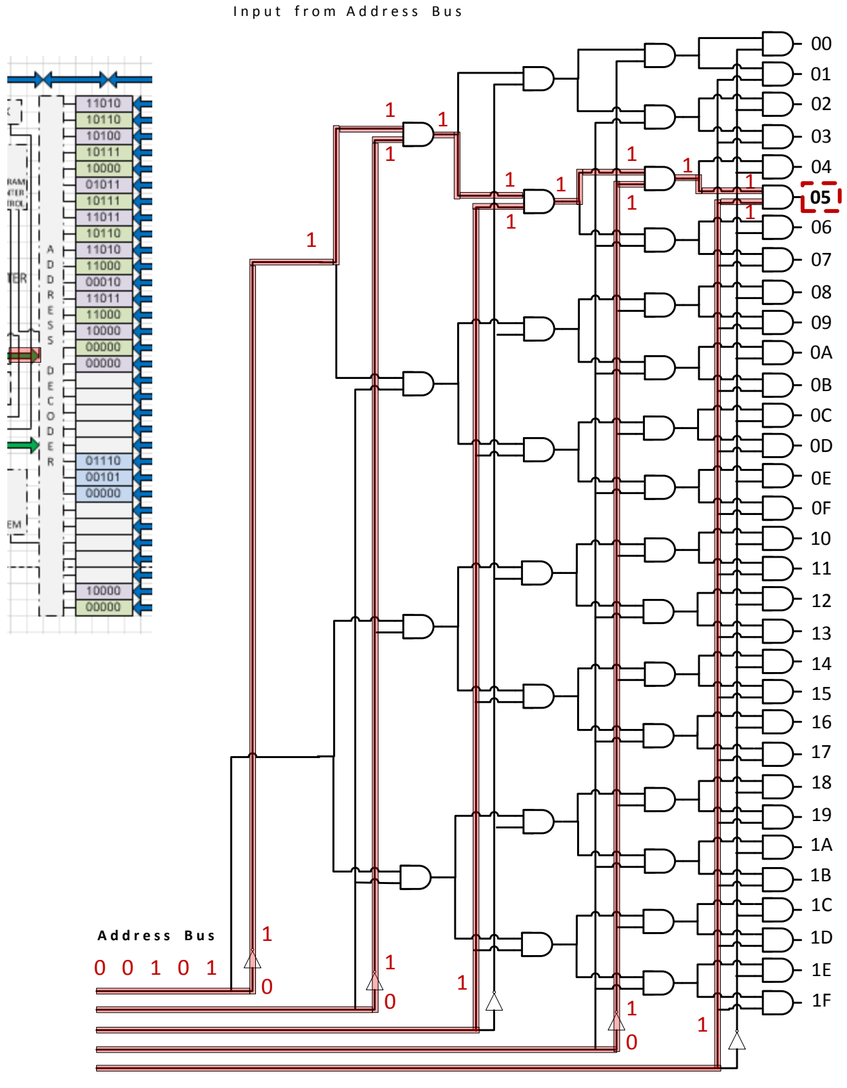
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

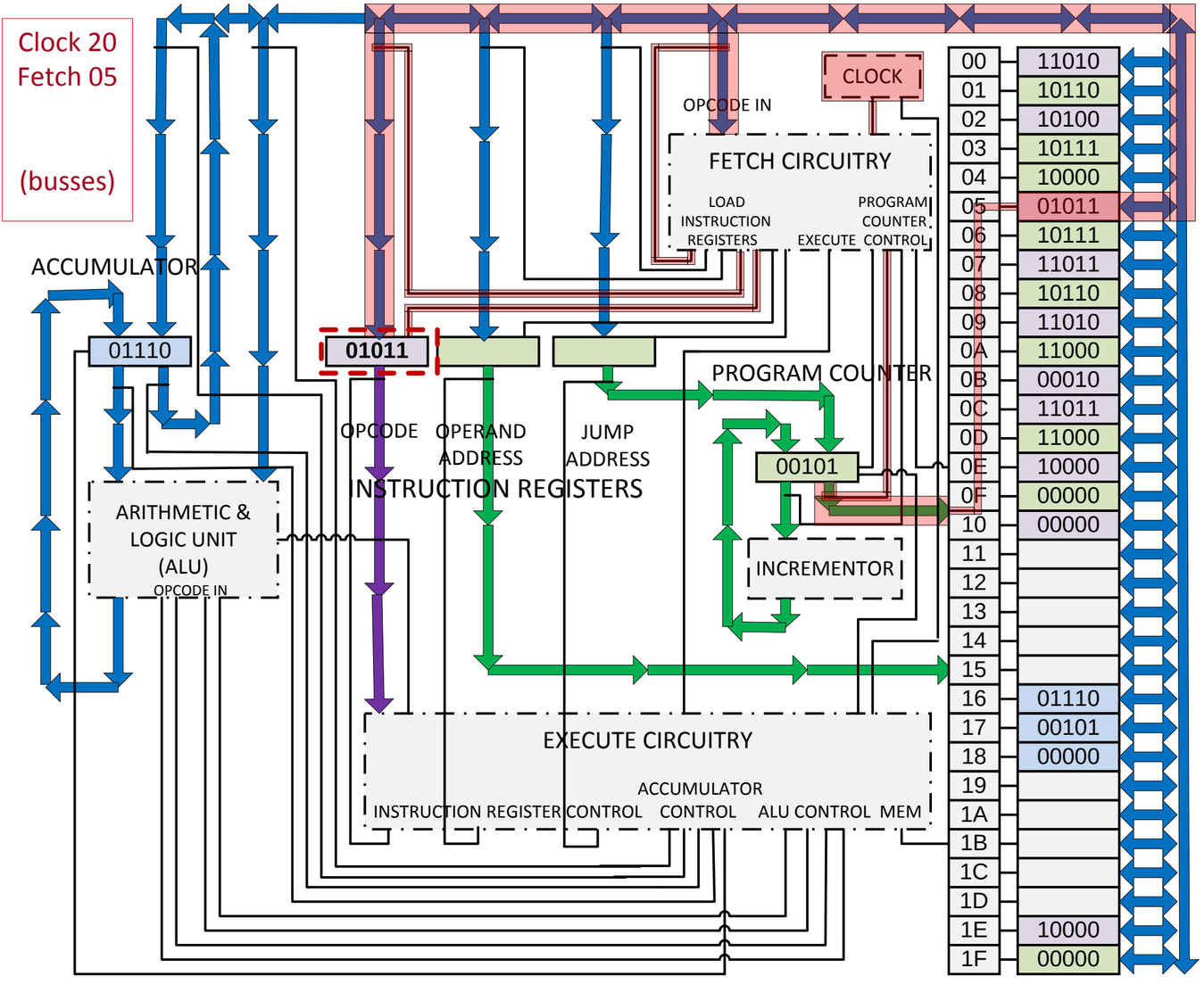


00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	01110
17	00101
18	00000
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

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

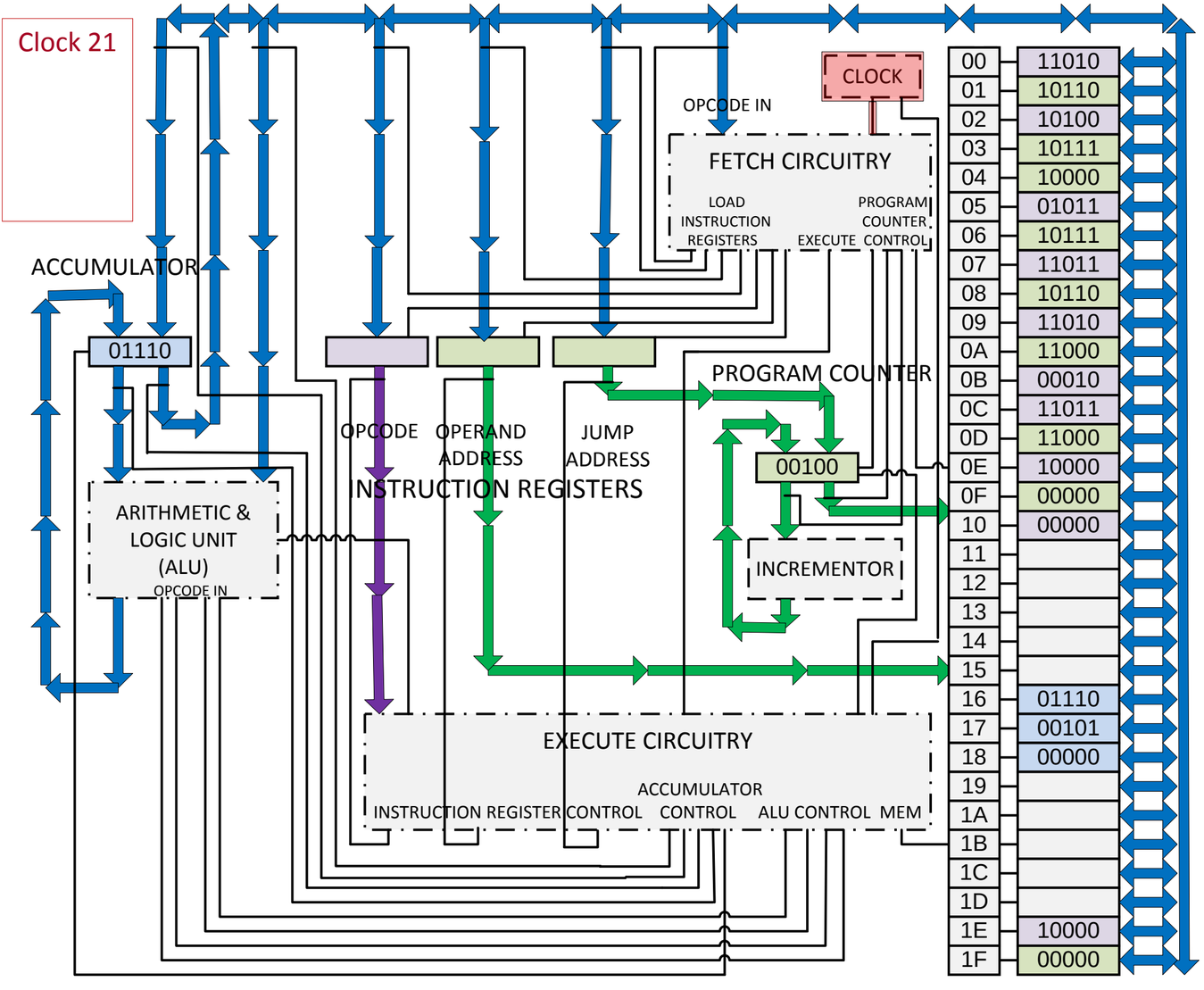
Address Decoder





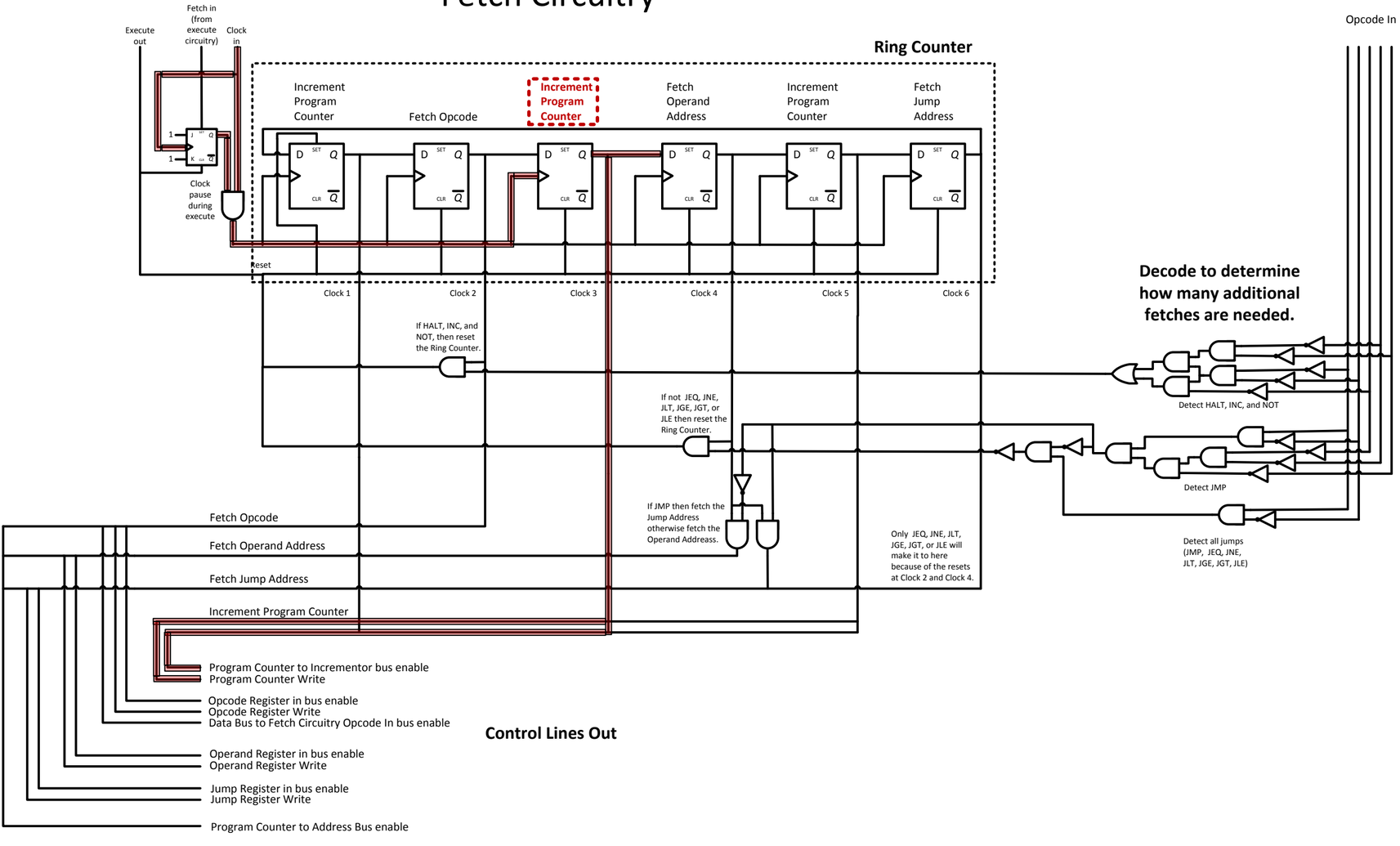
00	11010	
01	10110	
02	10100	
03	10111	
04	10000	
05	01011	
06	10111	
07	11011	
08	10110	
09	11010	
0A	11000	
0B	00010	
0C	11011	
0D	11000	
0E	10000	
0F	00000	
10	00000	
11		
12		
13		
14		
15		
16	01110	
17	00101	
18	00000	
19		
1A		
1B		
1C		
1D		
1E	10000	
1F	00000	

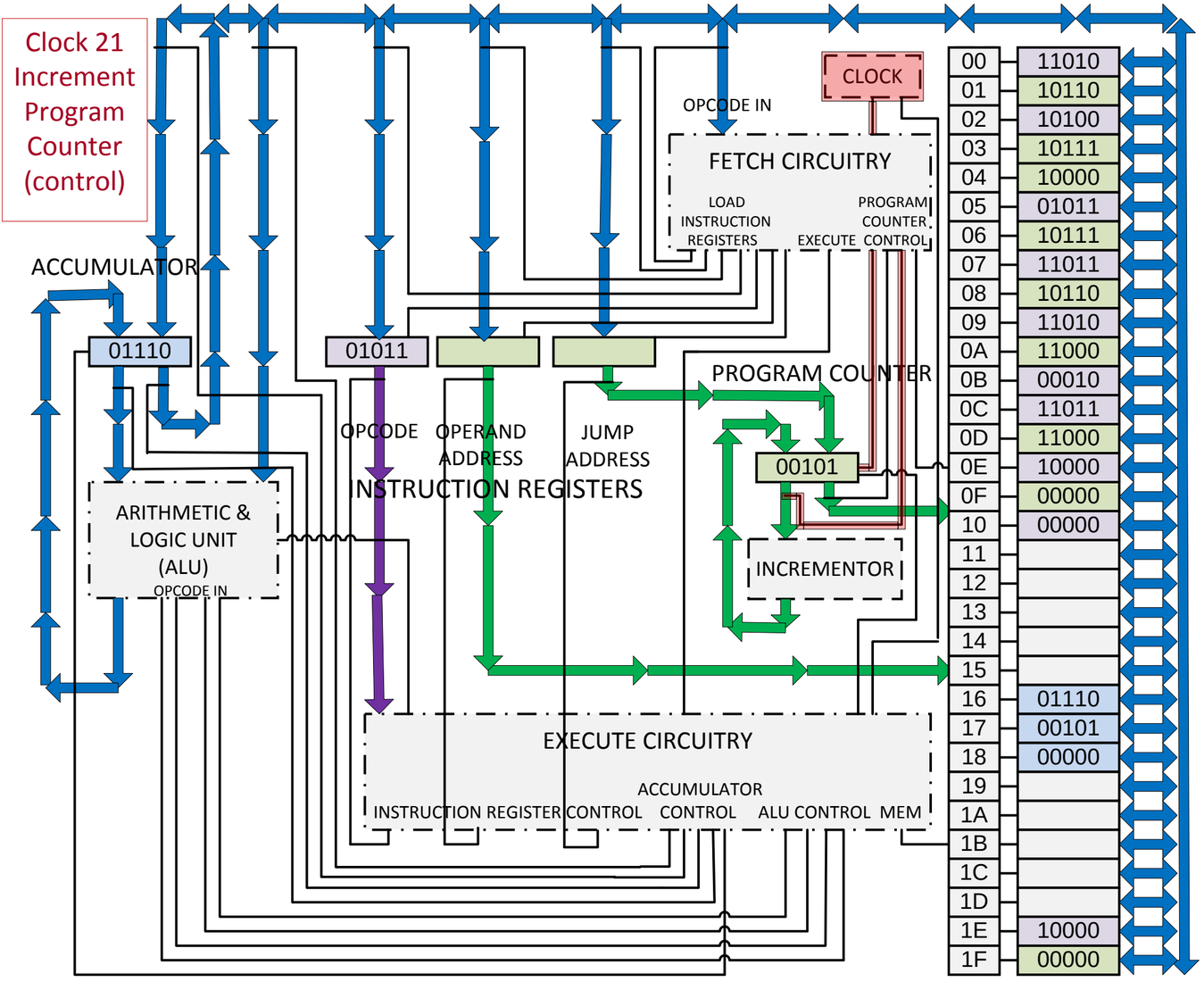
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



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

Fetch Circuitry





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

Incrementor

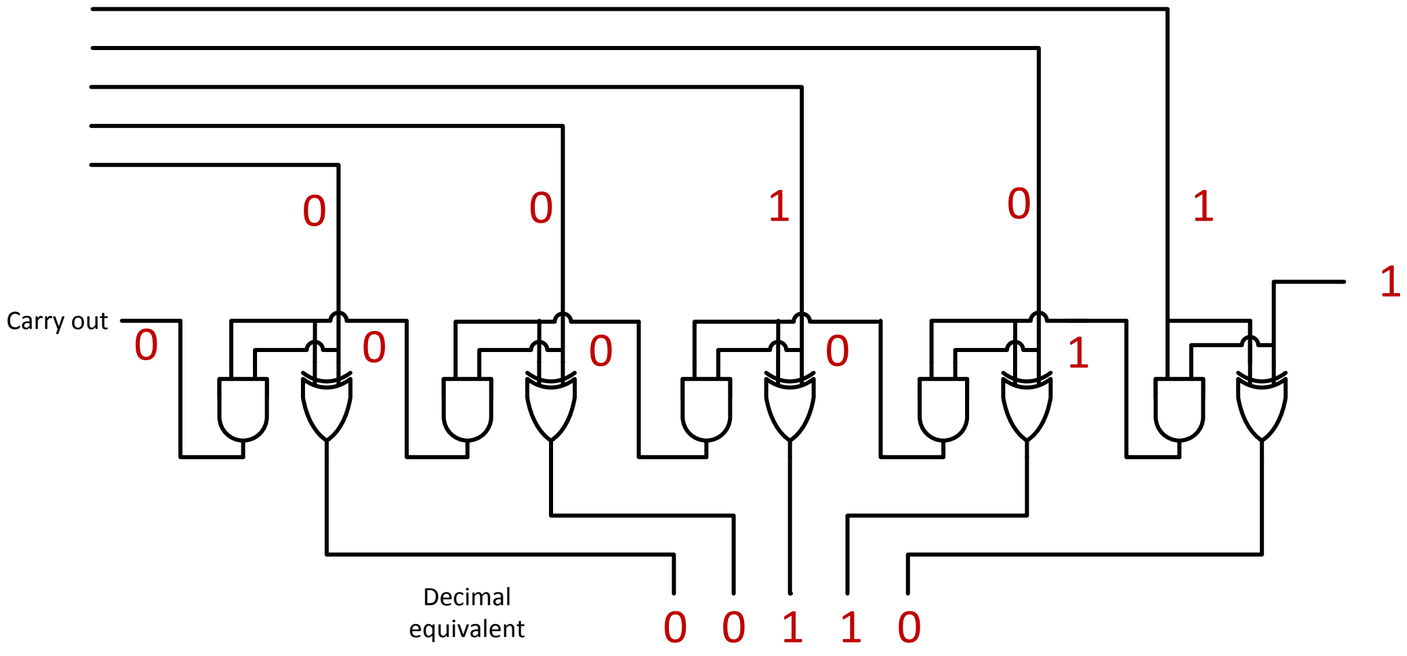
INC opcode

00101 + 00001

5 + 1

Left bus into ALU

0 0 1 0 1

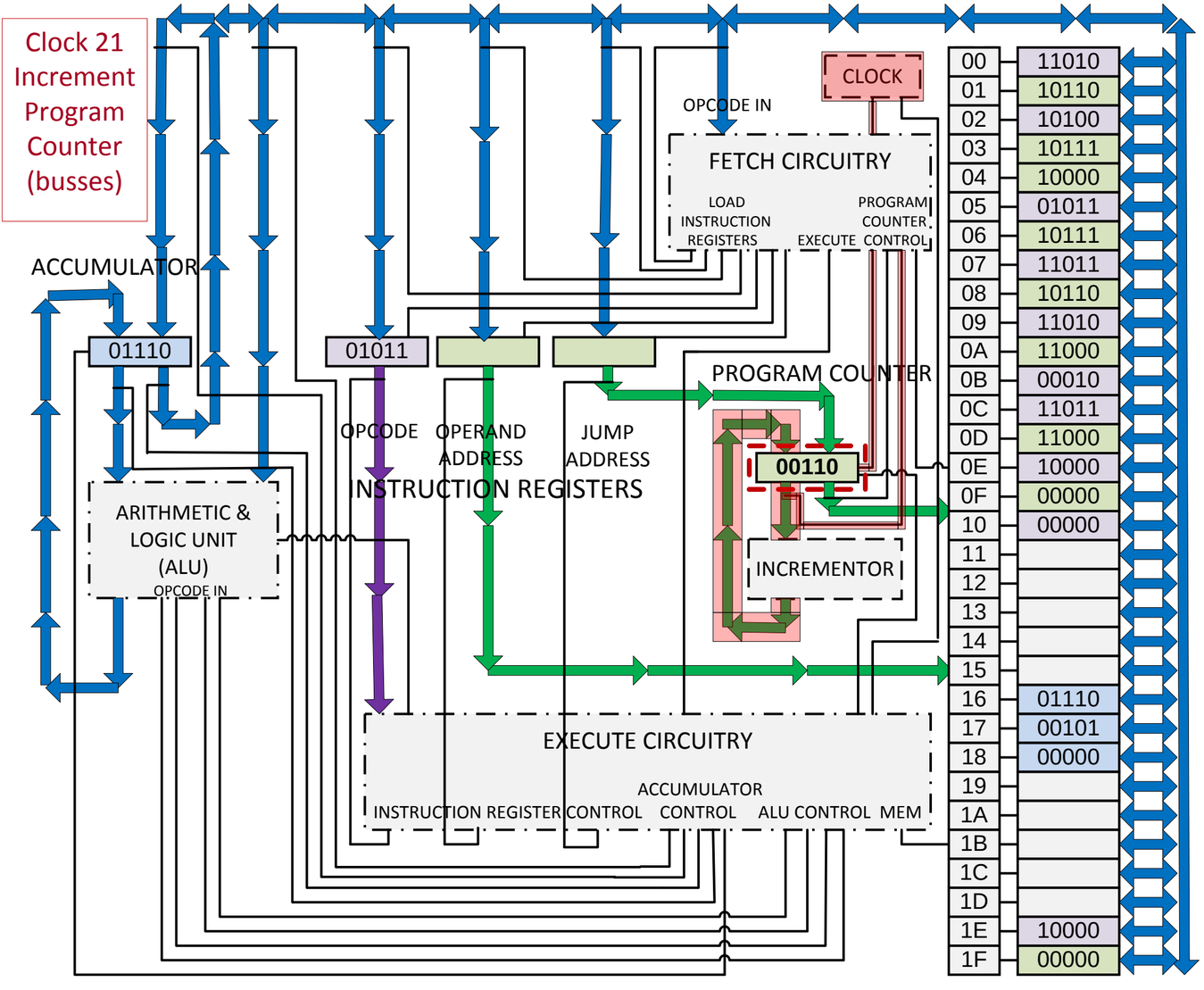


Decimal equivalent

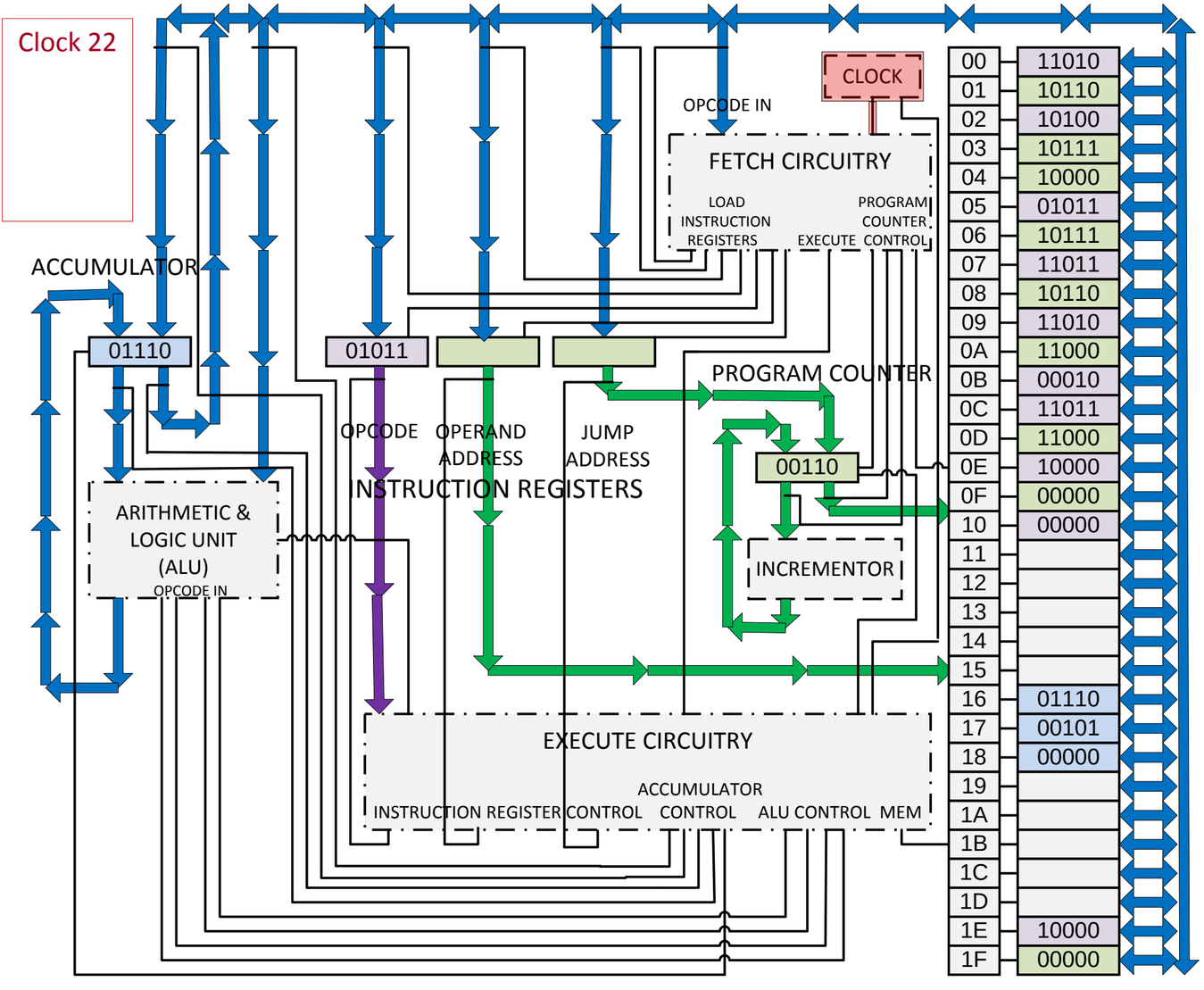
Output

Input number + 1

```
  0 0 1 0 1          5
+ 0 0 0 0 0      +   1
-----------      -----
  0 0 1 1 0          6
```

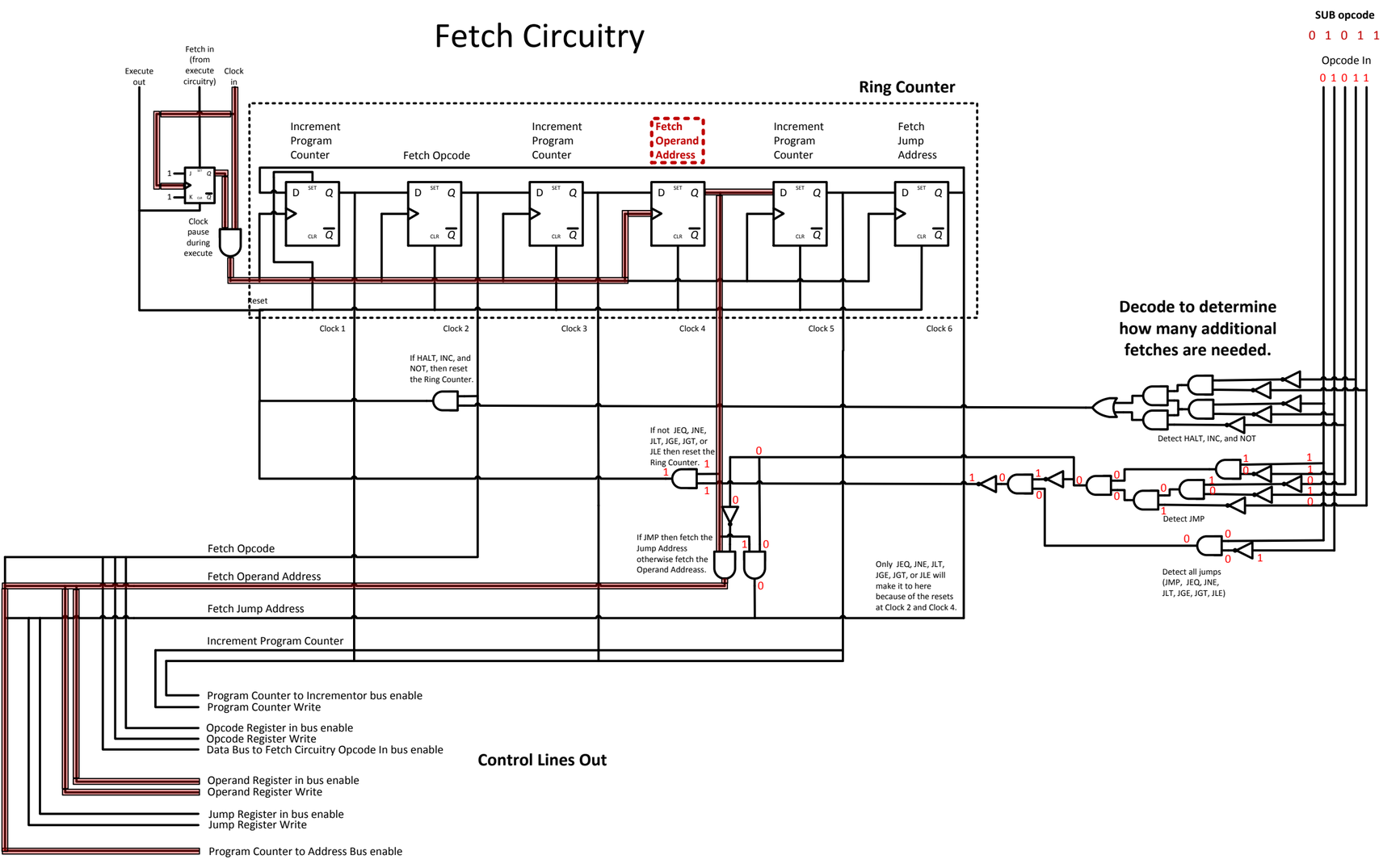


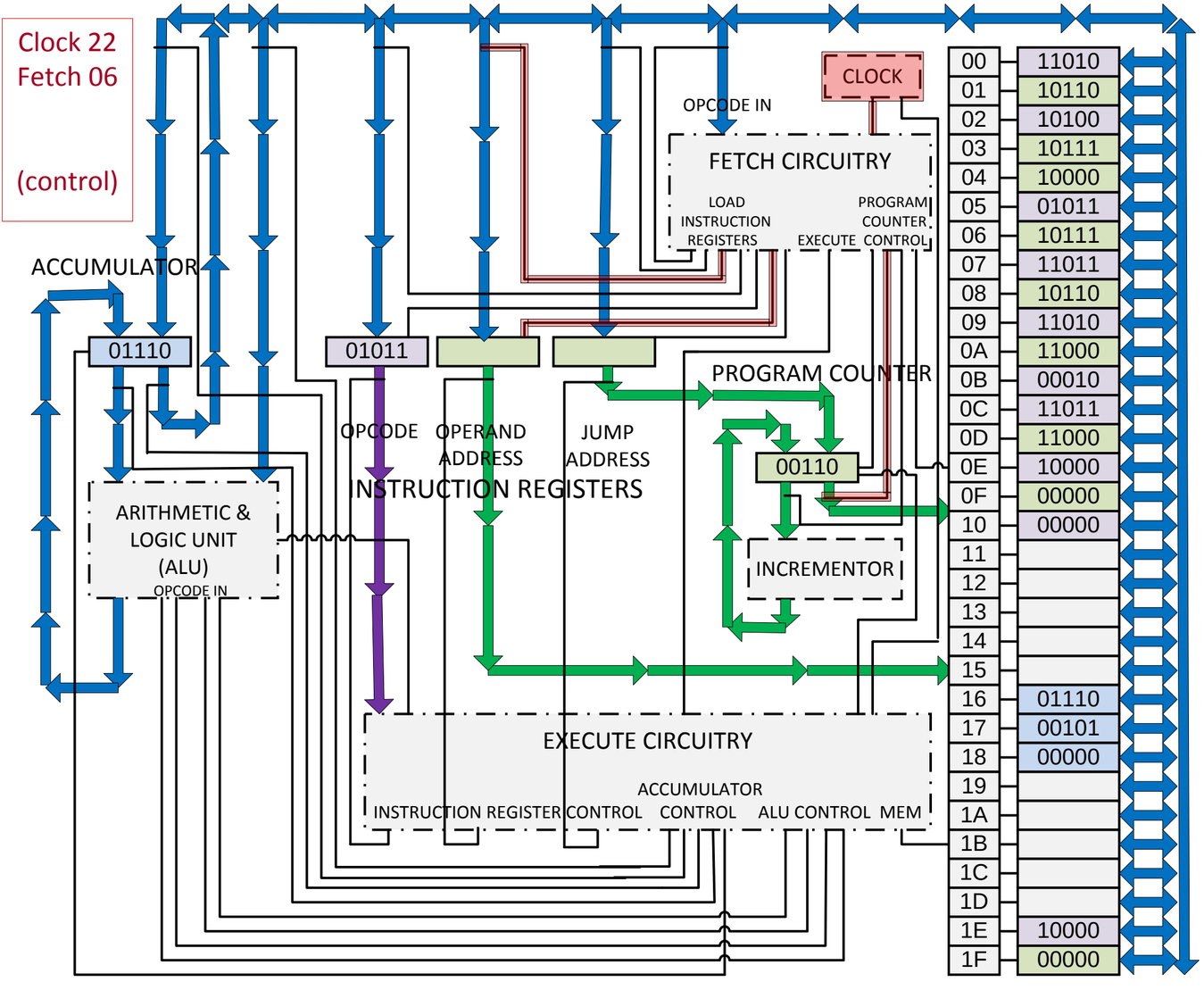
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



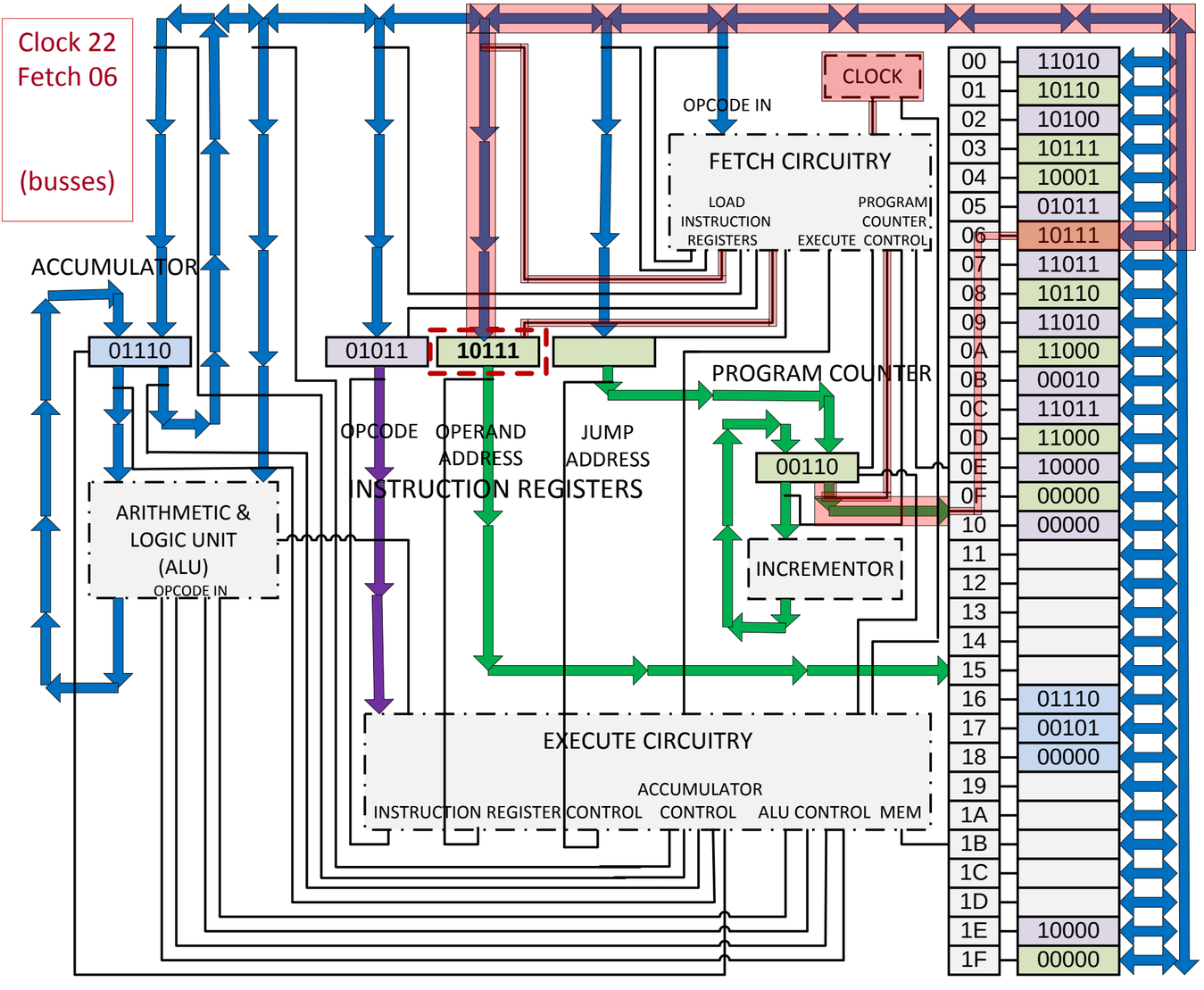
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

Fetch Circuitry



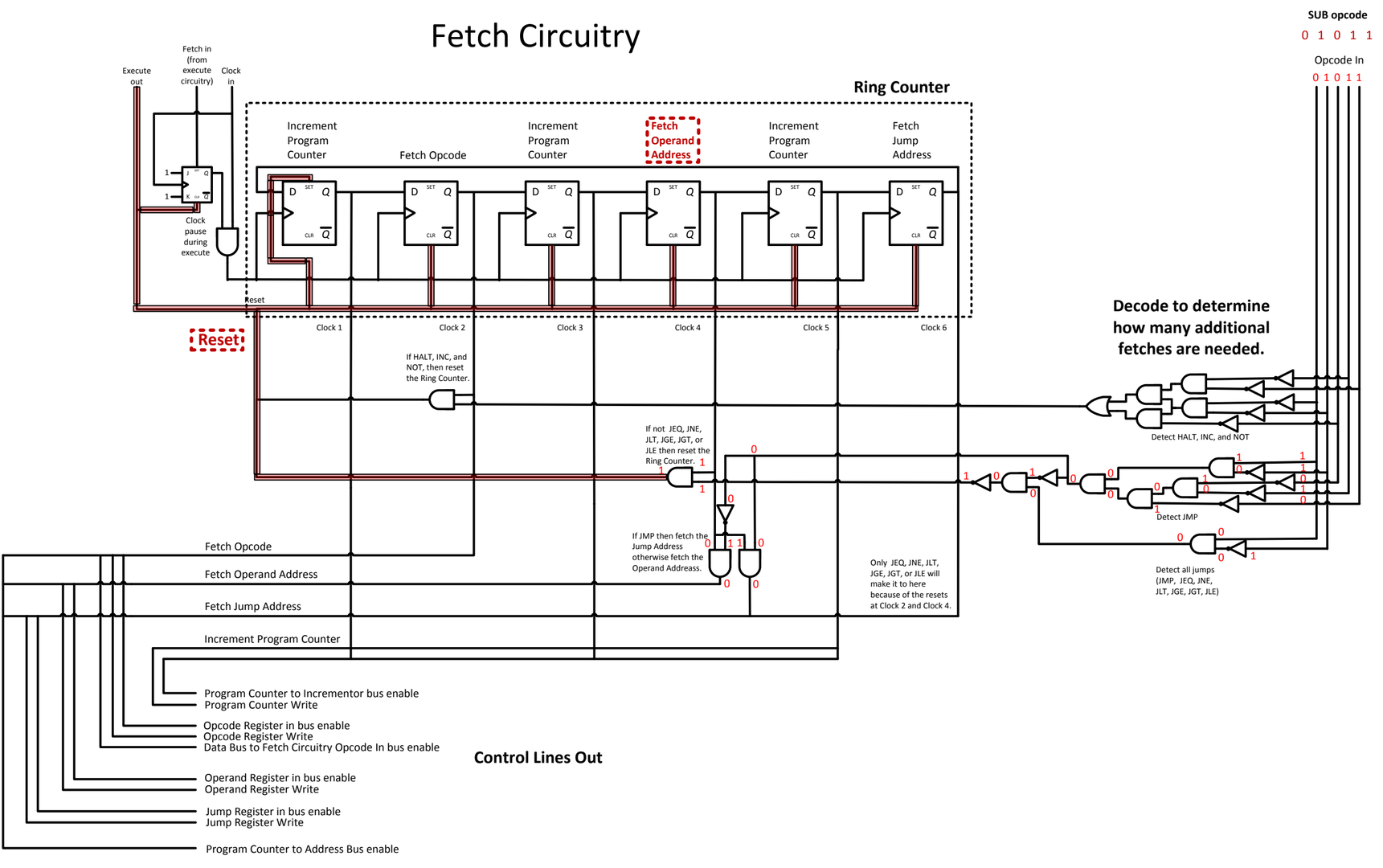


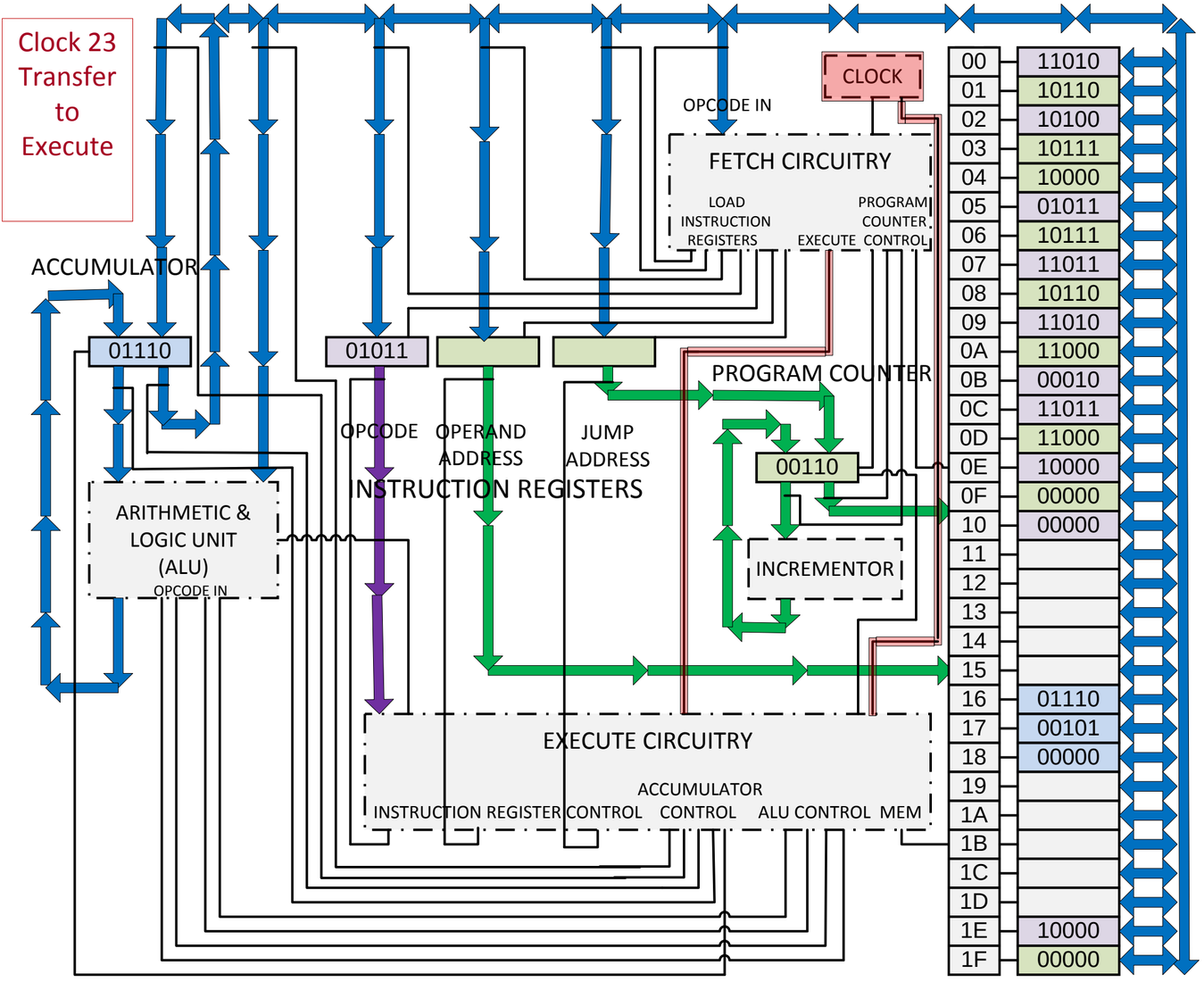
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



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

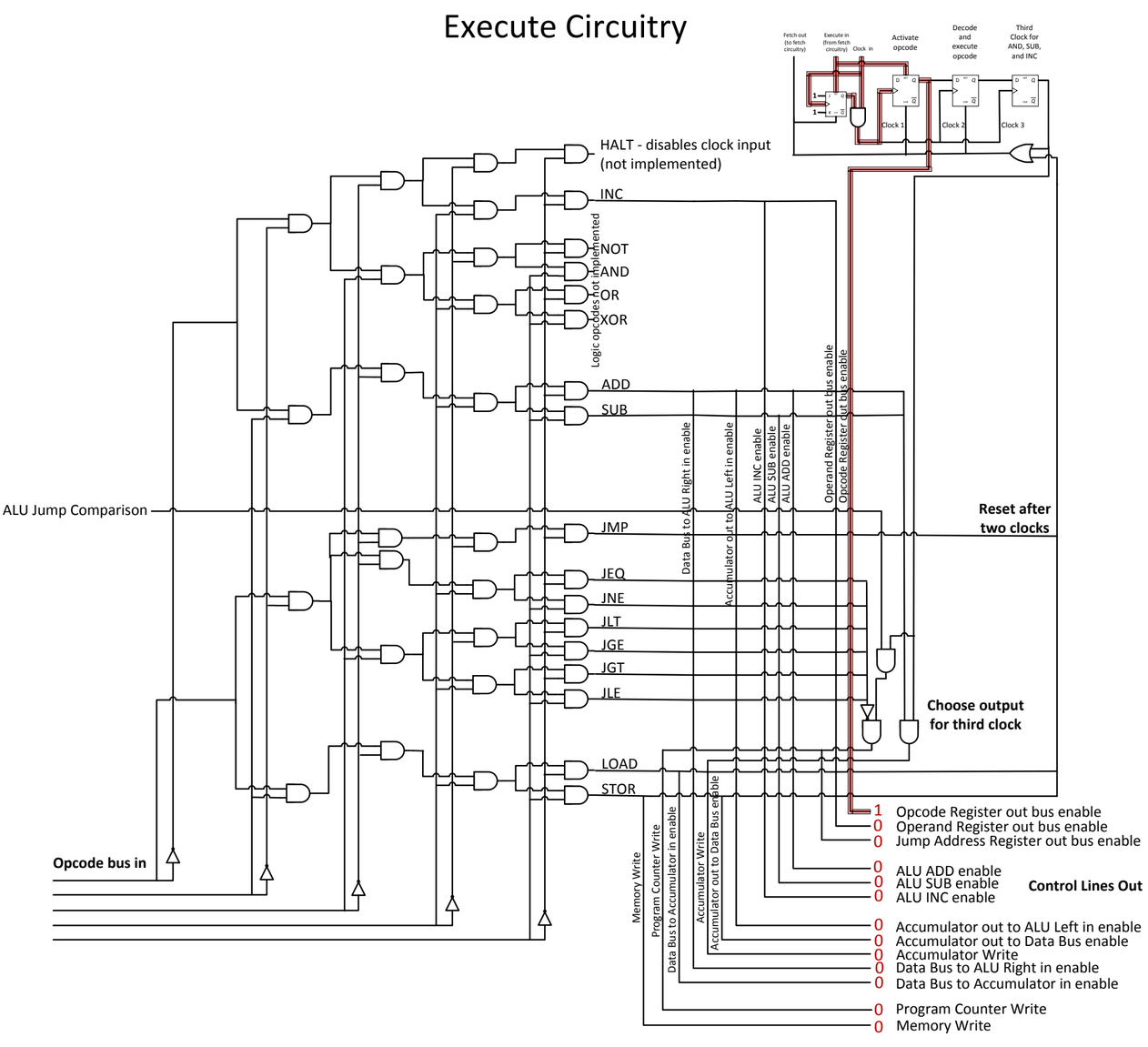
Fetch Circuitry

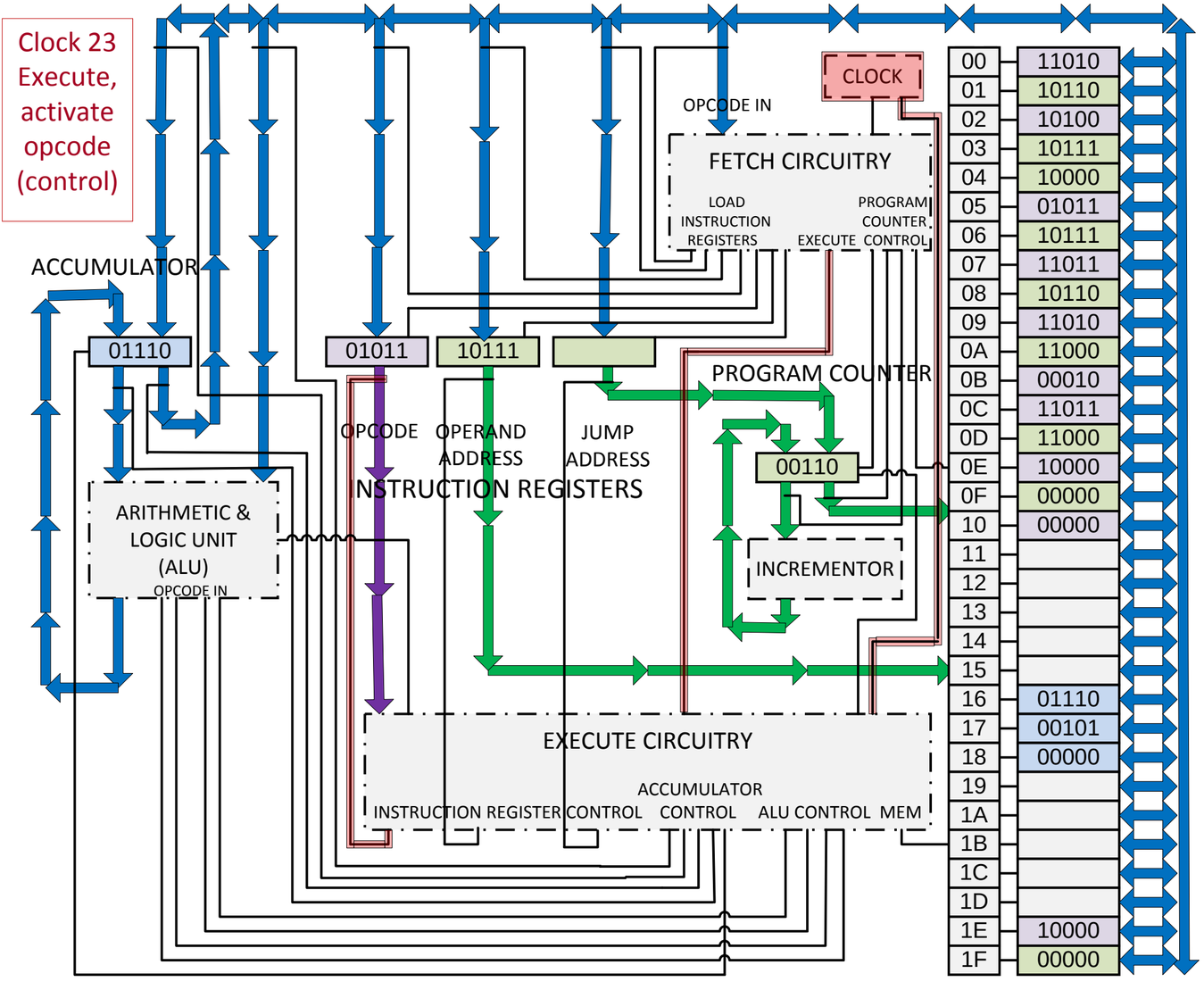




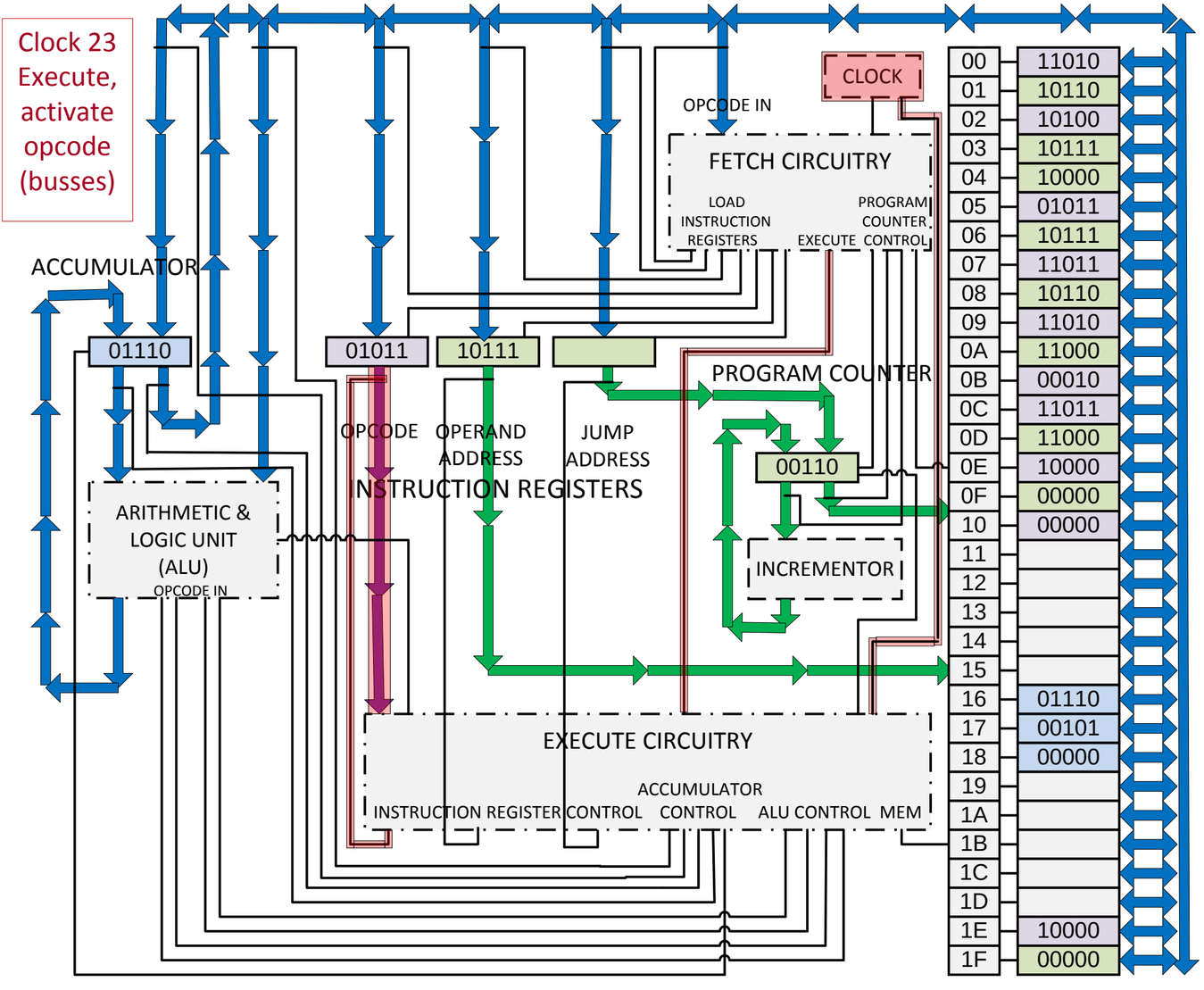
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

Execute Circuitry



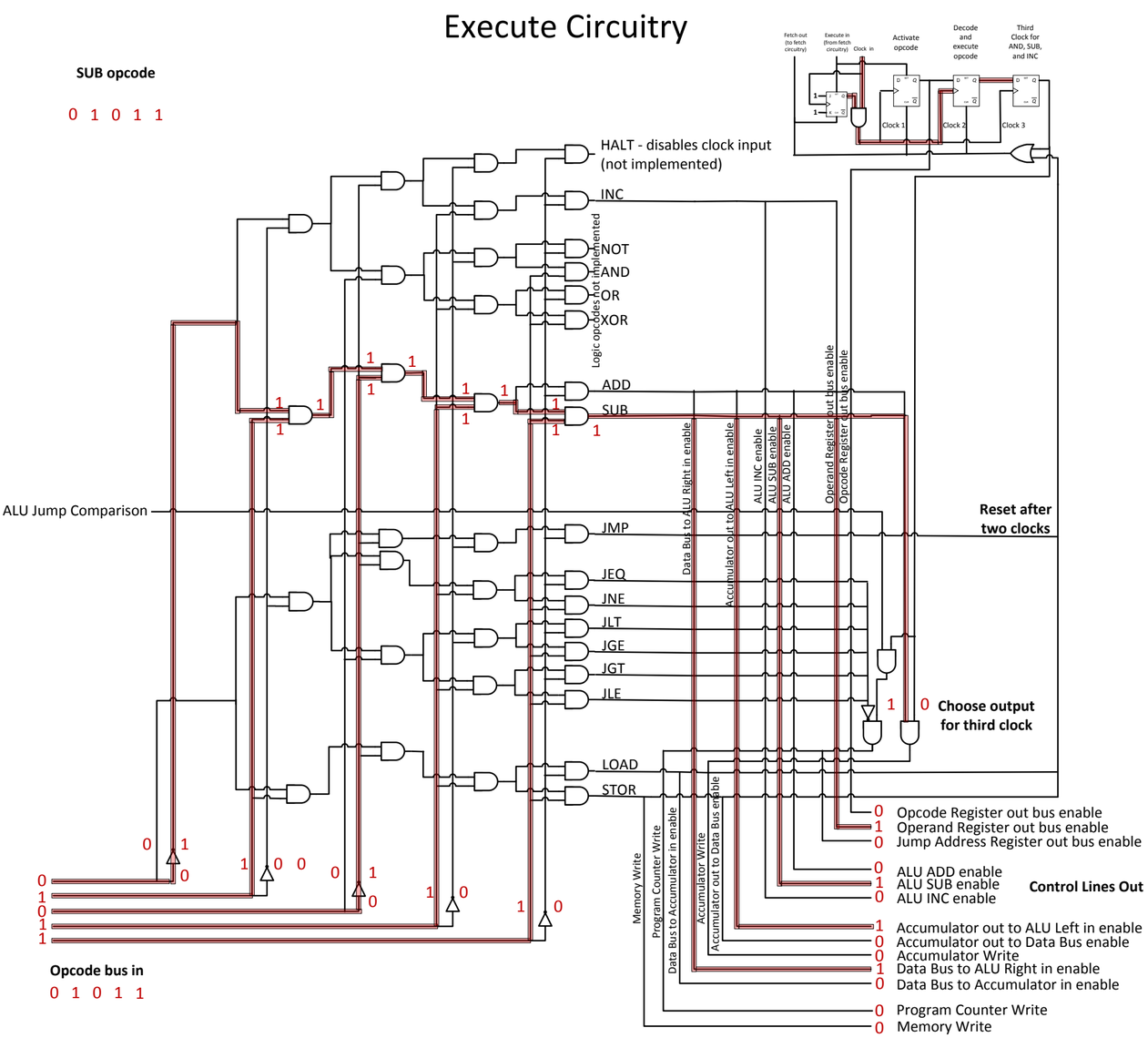


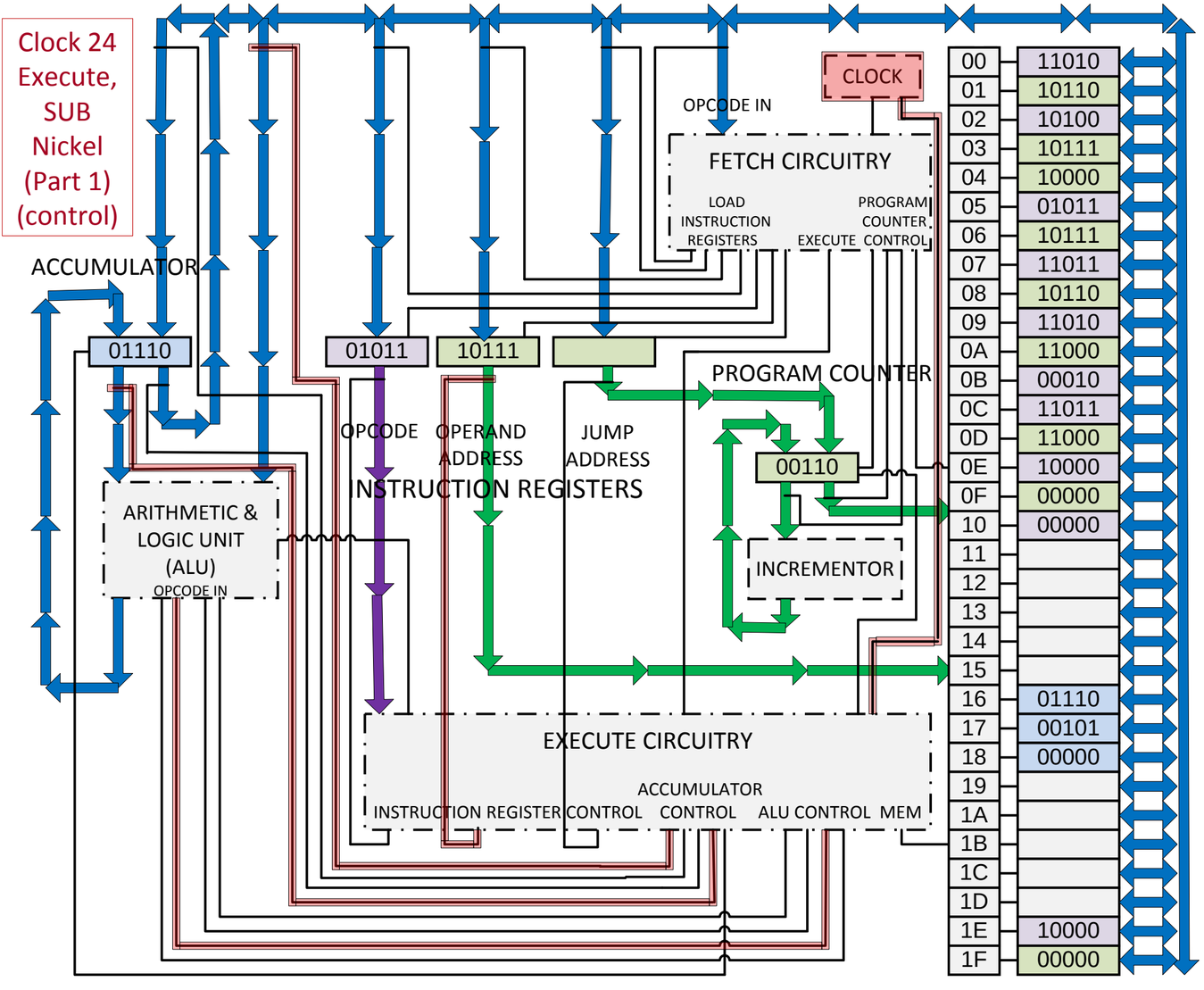
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



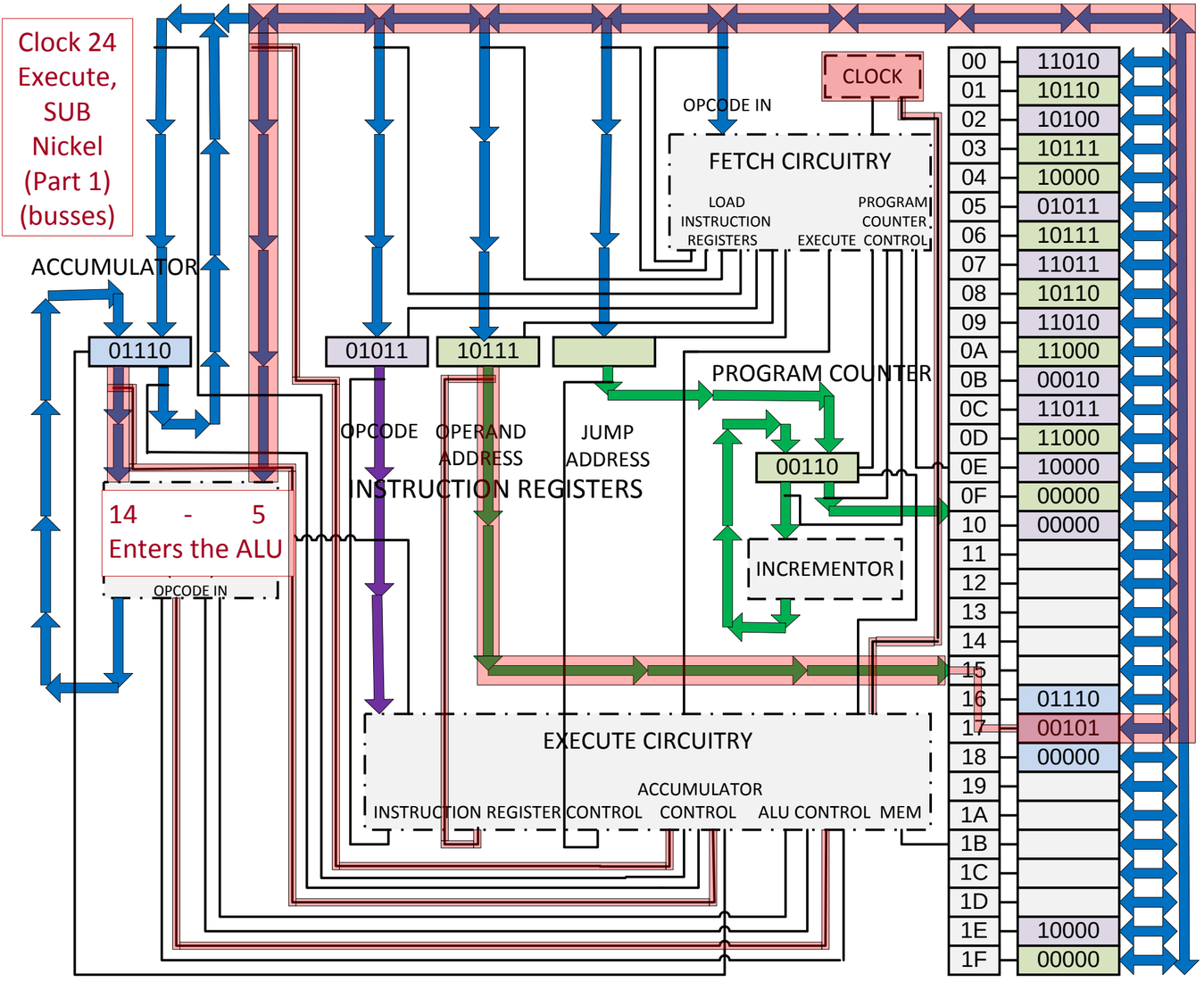
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

Execute Circuitry



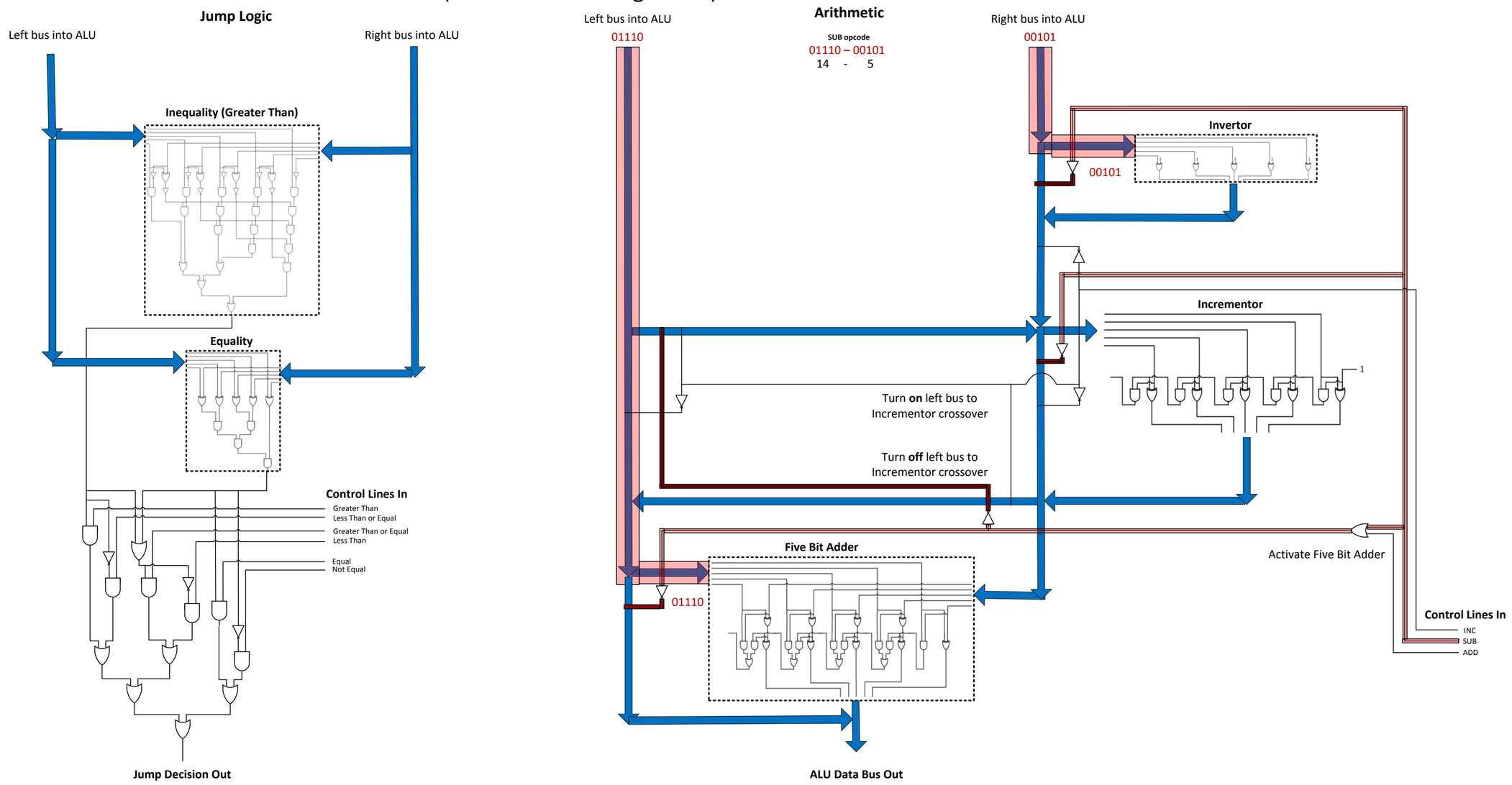


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



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

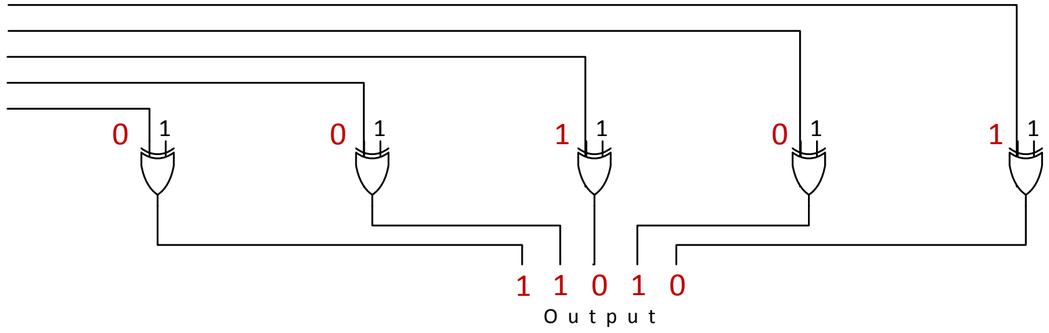
A L U
(Arithmetic and Logic Unit)



Invertor

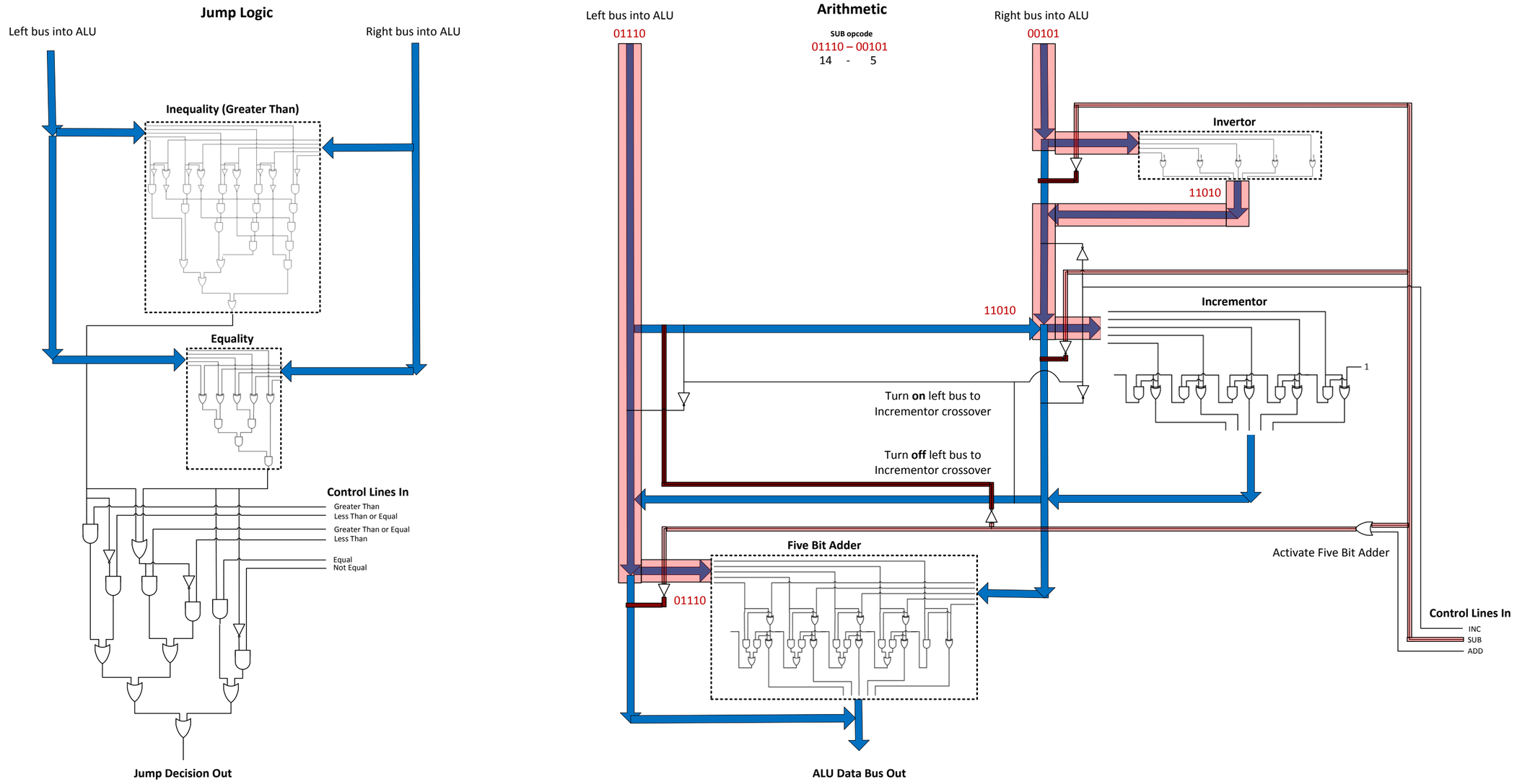
Bus input

0 0 1 0 1



twos complement of the input bus

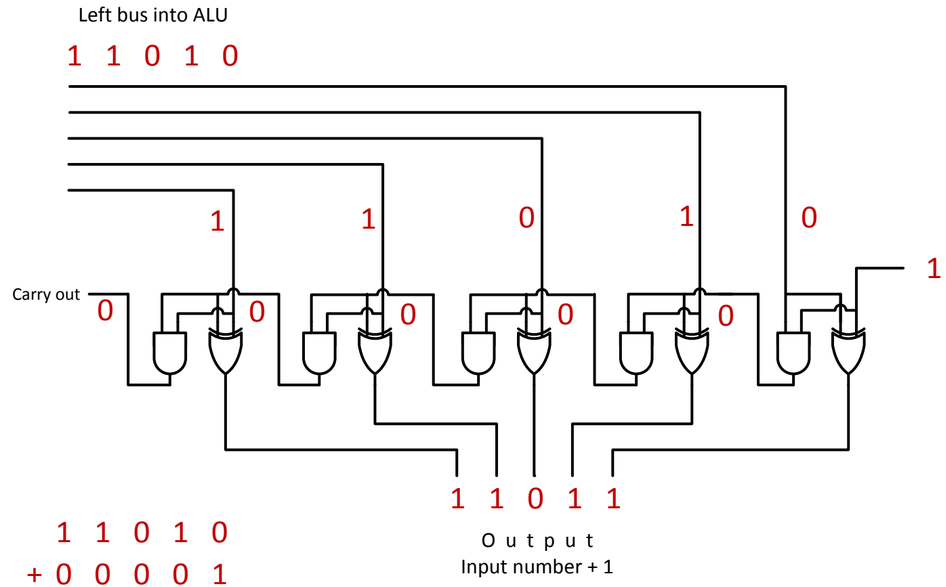
A L U
(Arithmetic and Logic Unit)



Incrementor

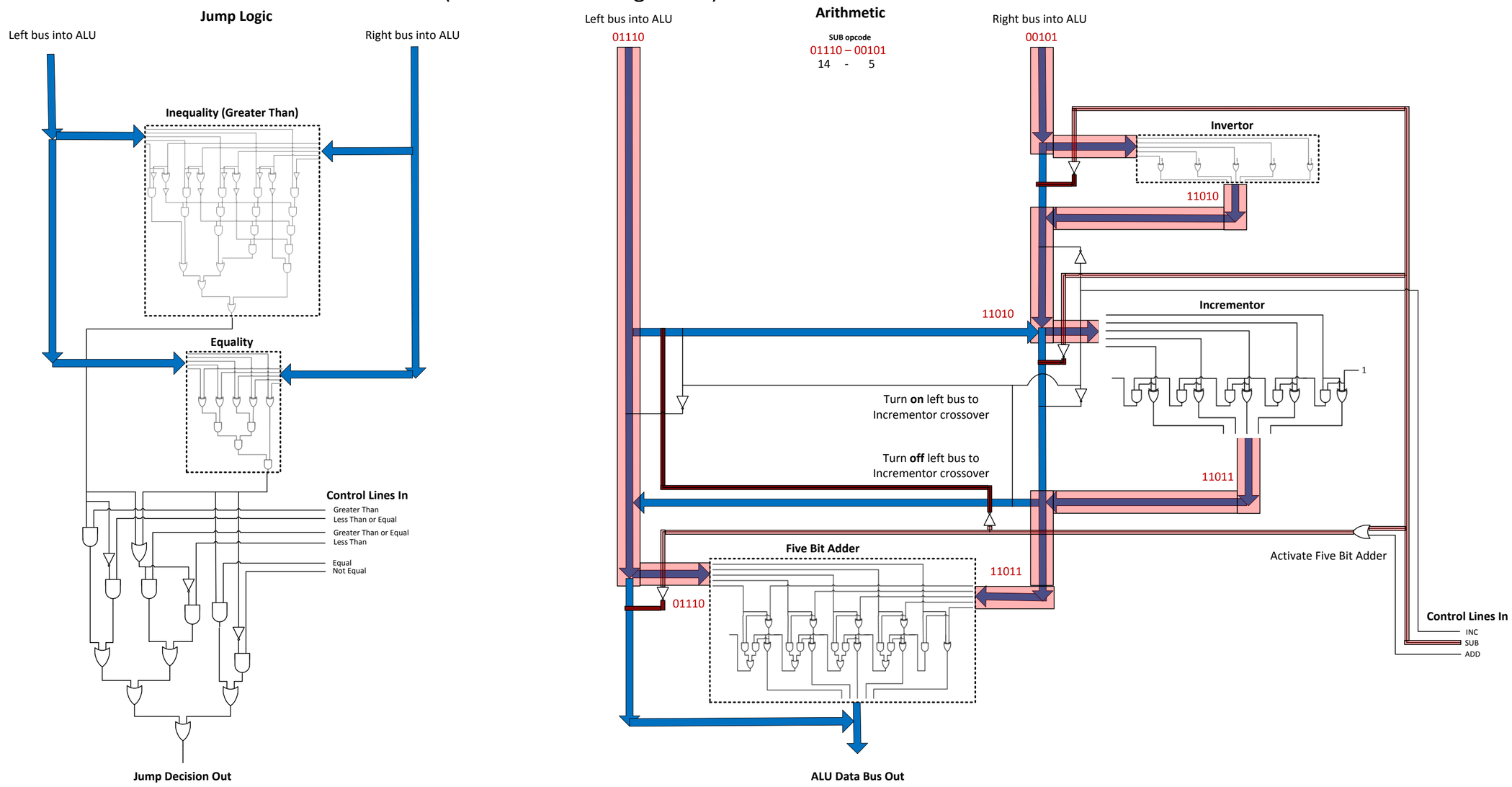
INC opcode

11010 + 00001




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

A L U
(Arithmetic and Logic Unit)



Five Bit Adder

SUB opcode

01110 – 00101

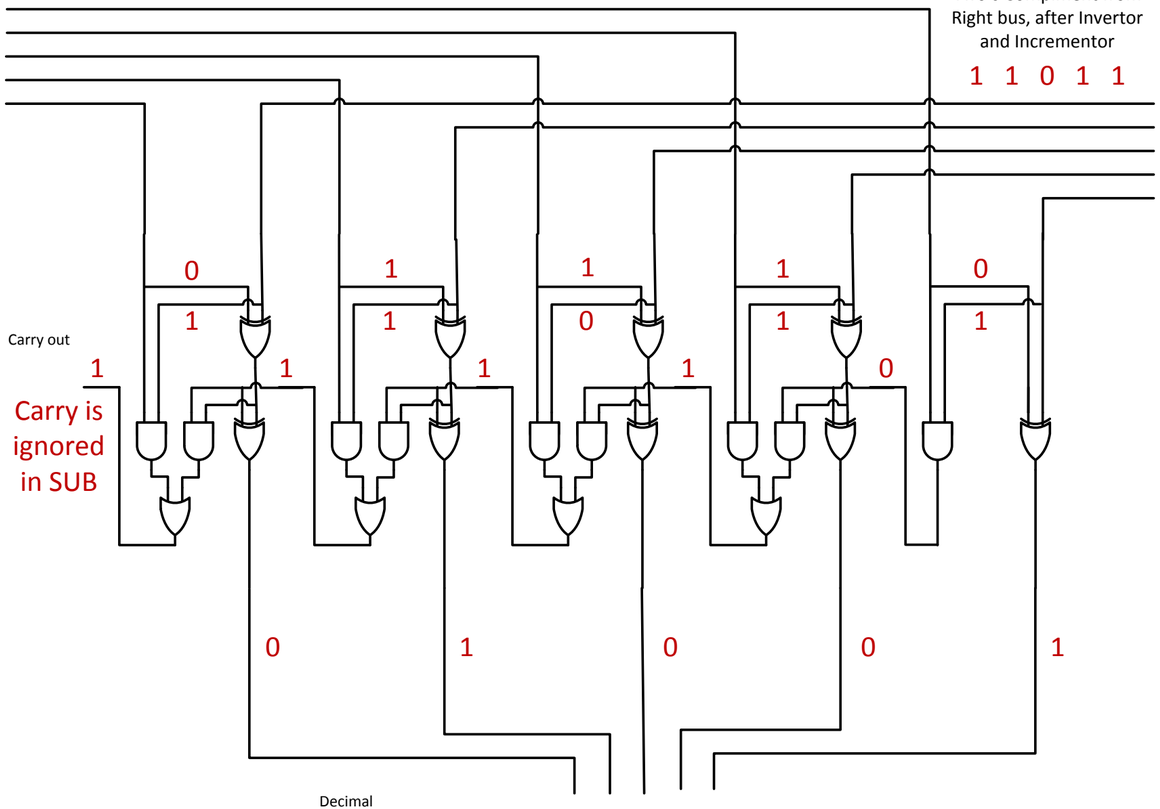
14 - 5

Left bus from ALU

0 1 1 1 0

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

1 1 0 1 1



Decimal equivalent

14

+ (-5)

9

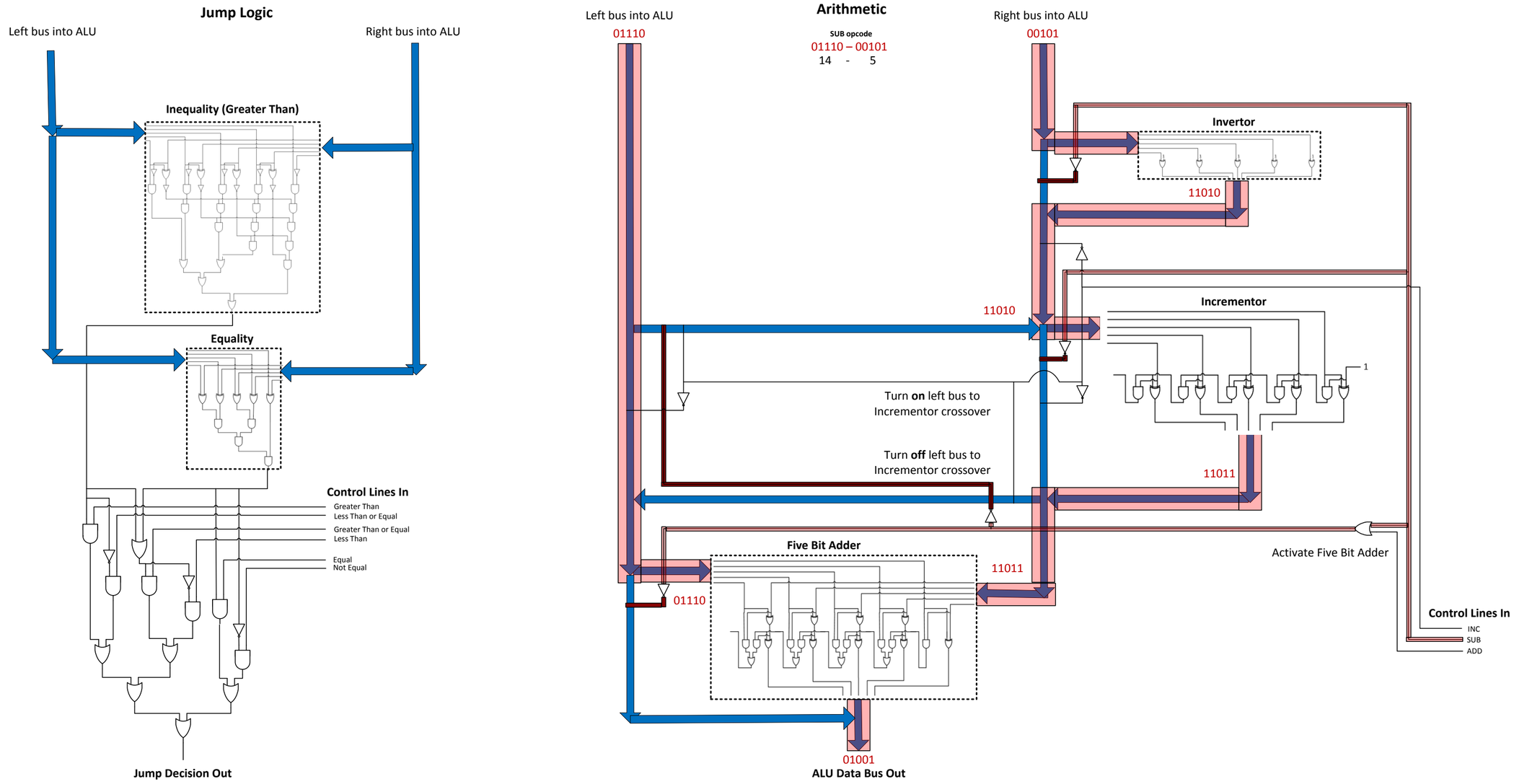
0 1 1 1 0

+ 1 1 0 1 1

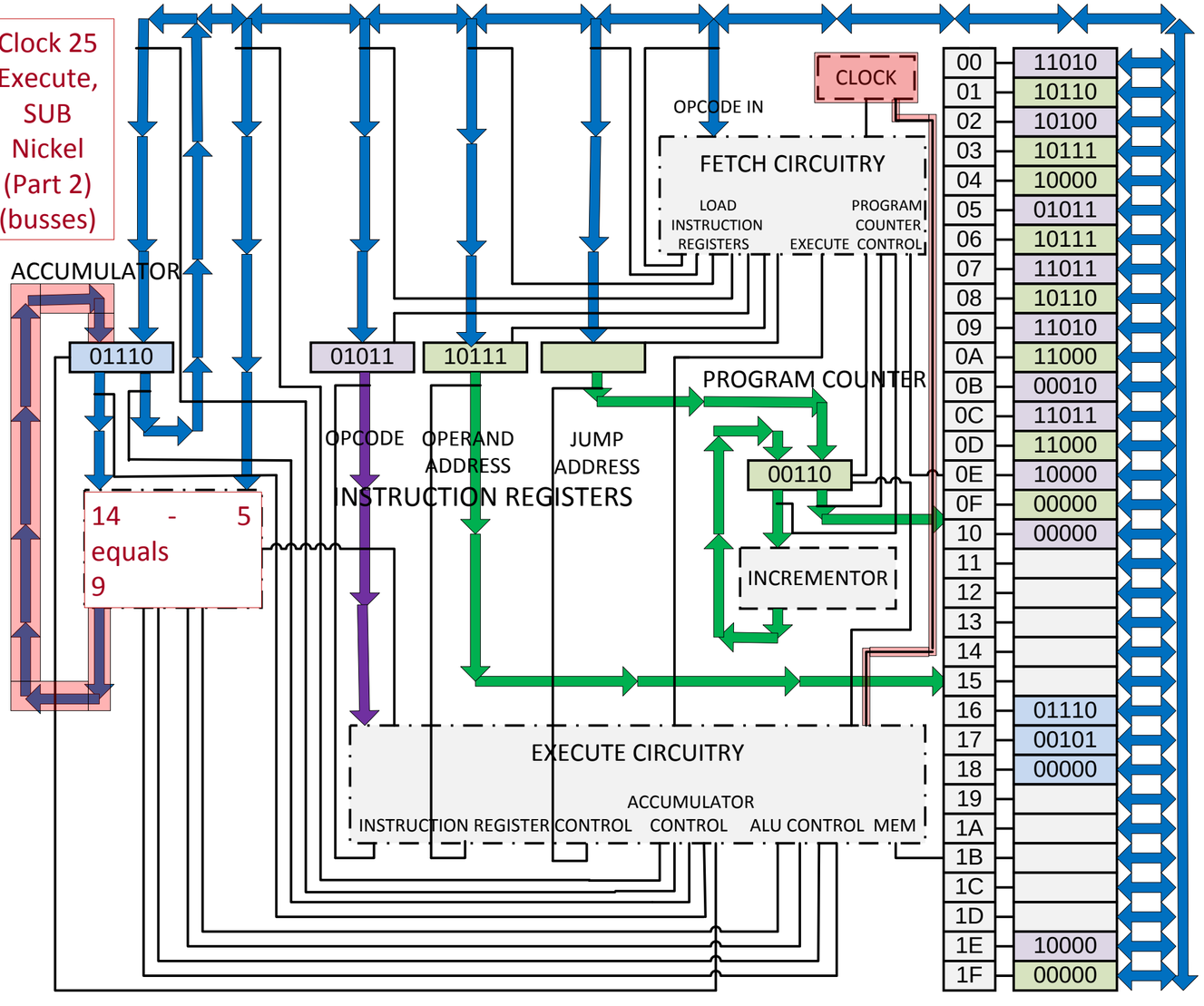
0 1 0 0 1

Accurate addition of two 5-bit binary numbers.

A L U
(Arithmetic and Logic Unit)



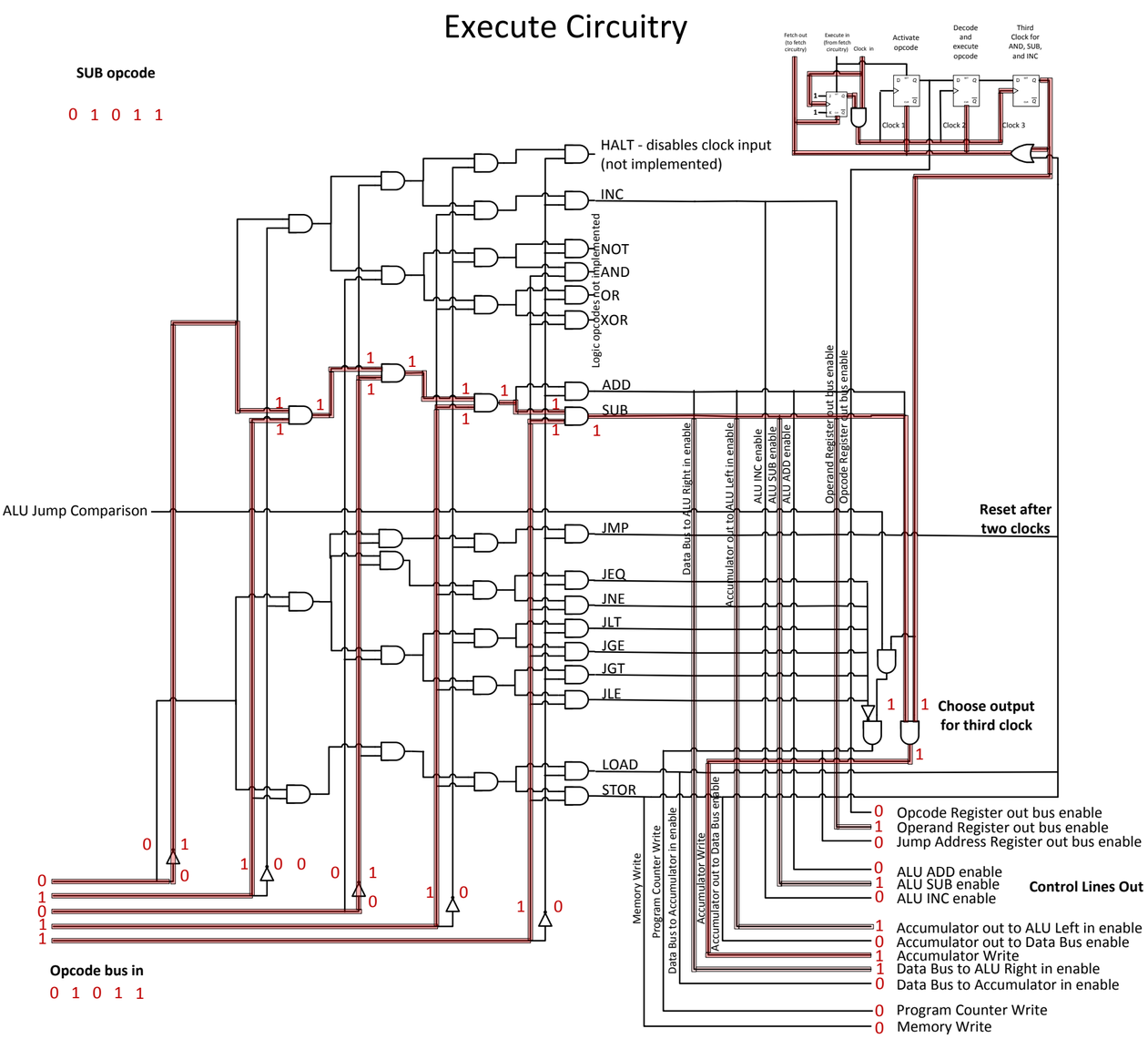
Clock 25
Execute,
SUB
Nickel
(Part 2)
(busses)



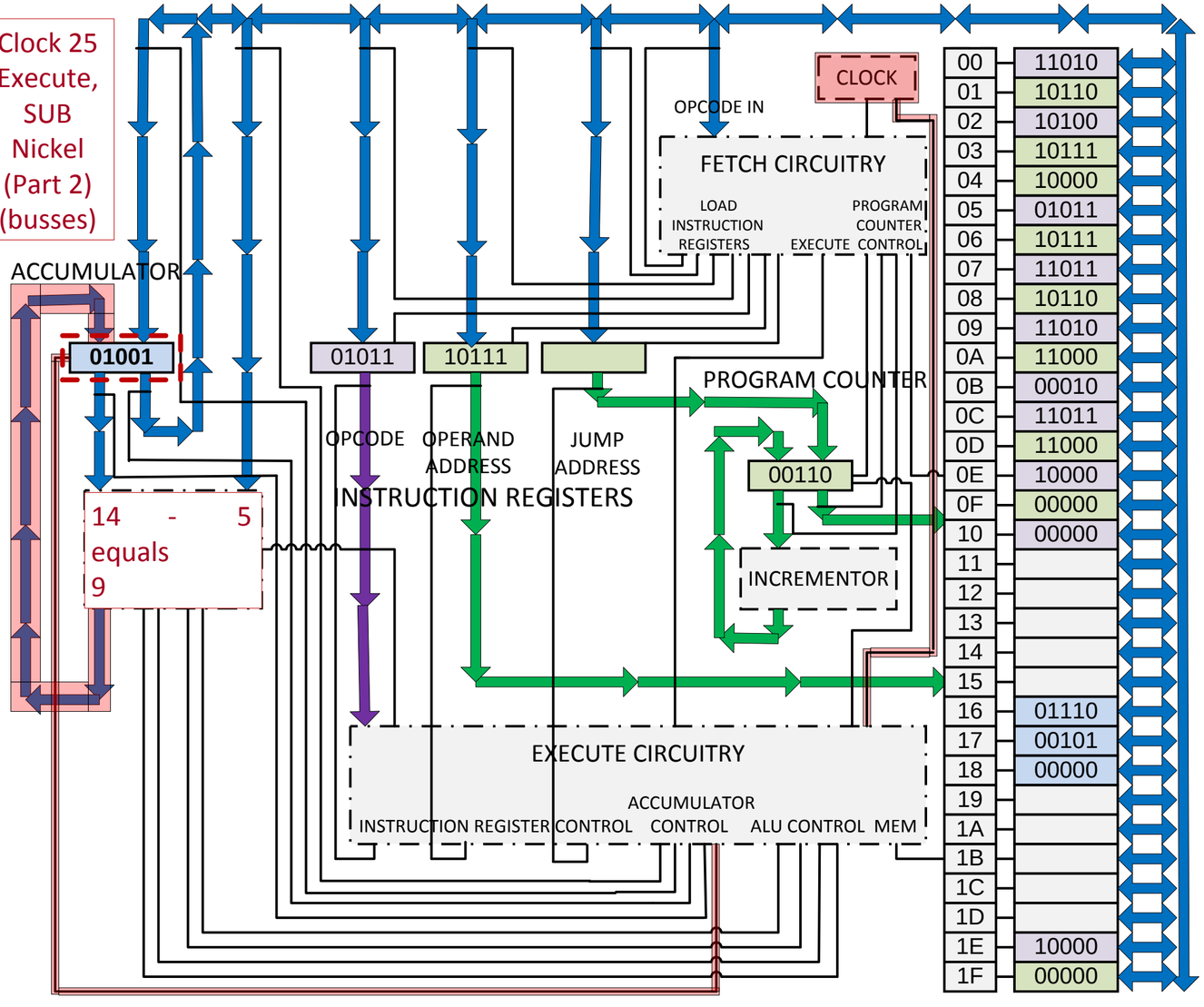
00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	01110
17	00101
18	00000
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

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

Execute Circuitry



Clock 25
Execute,
SUB
Nickel
(Part 2)
(busses)



		Assembly Language Addresses:	Program:	Explanation
00	11010	Again:	LOAD	# copy the value at ‘cents’ to the Accumulator
01	10110		cents	
02	10100		JLT	# if the value at ‘nickel’ is less than the Accumulator then
03	10111		nickel	# jump to the address ‘Done:’
04	10000		Done:	
05	01011		SUB	# subtract the value at ‘nickel’ from the Accumulator
06	10111		nickel	# and put the result back into the Accumulator
07	11011		STOR	# copy the value in the Accumulator to ‘cents’
08	10110		cents	
09	11010		LOAD	# copy the value at address ‘count’ to the Accumulator
0A	11000		count	
0B	00010		INC	# add 1 to the Accumulator
0C	11011		STOR	# copy the value in the Accumulator to ‘count’
0D	11000		count	
0E	10000		JMP	# unconditional jump to the address ‘Again:’
0F	00000		Again:	
10	00000	Done:	HALT	# stop the processor – end of program
11				
12				
13				
14				
15				
16	01110	cents	14	# variable -- the name ‘cents’ is the address and 14 is the value
17	00101	nickel	5	# variable -- the name ‘cents’ is the address and 5 is the value
18	00000	count	0	# variable -- the name ‘count’ is the address and 0 is the value
19				
1A				
1B				
1C				
1D				
1E	10000		JMP	# unconditional jump to the address ‘Again:’
1F	00000		Again:	

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