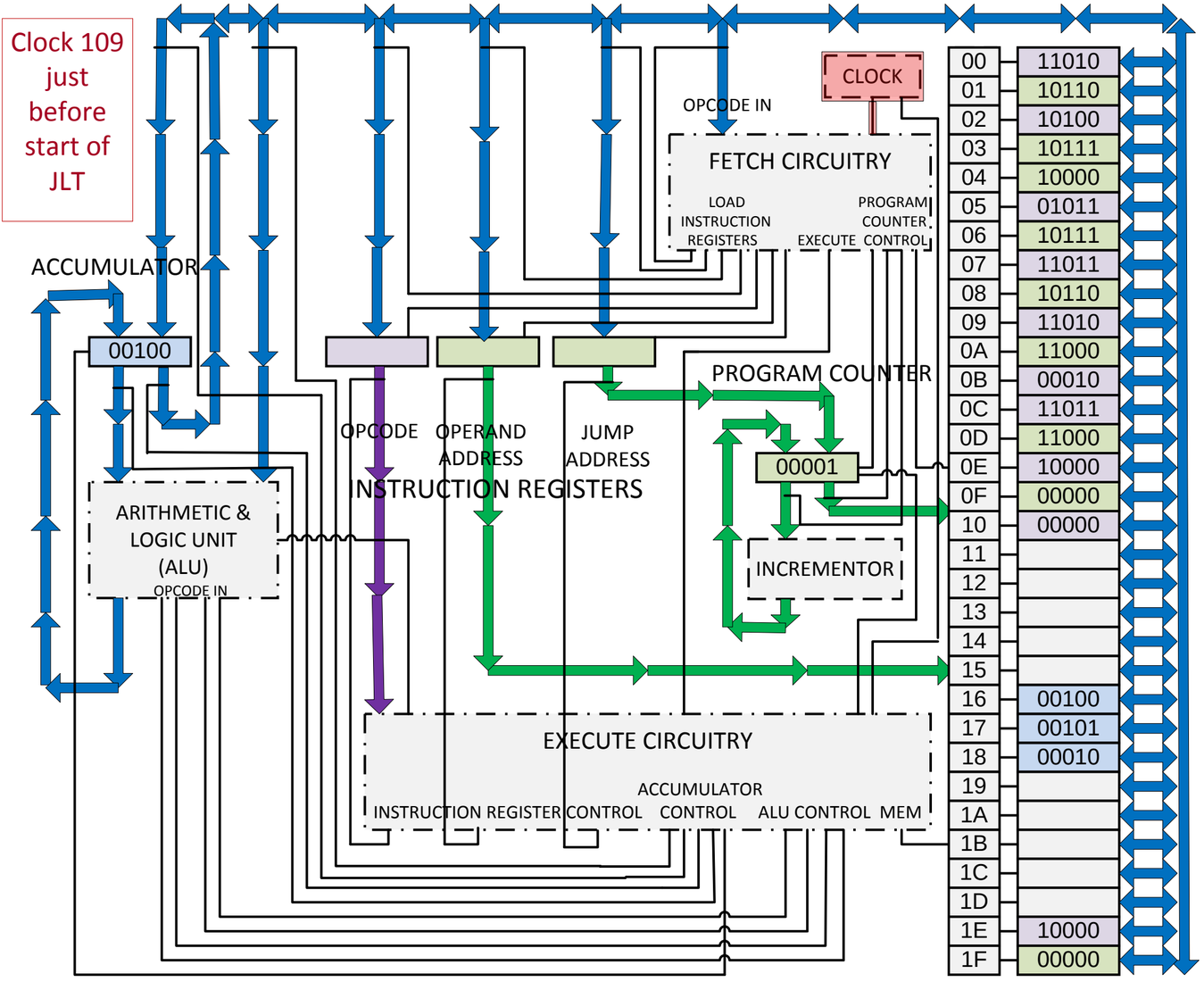
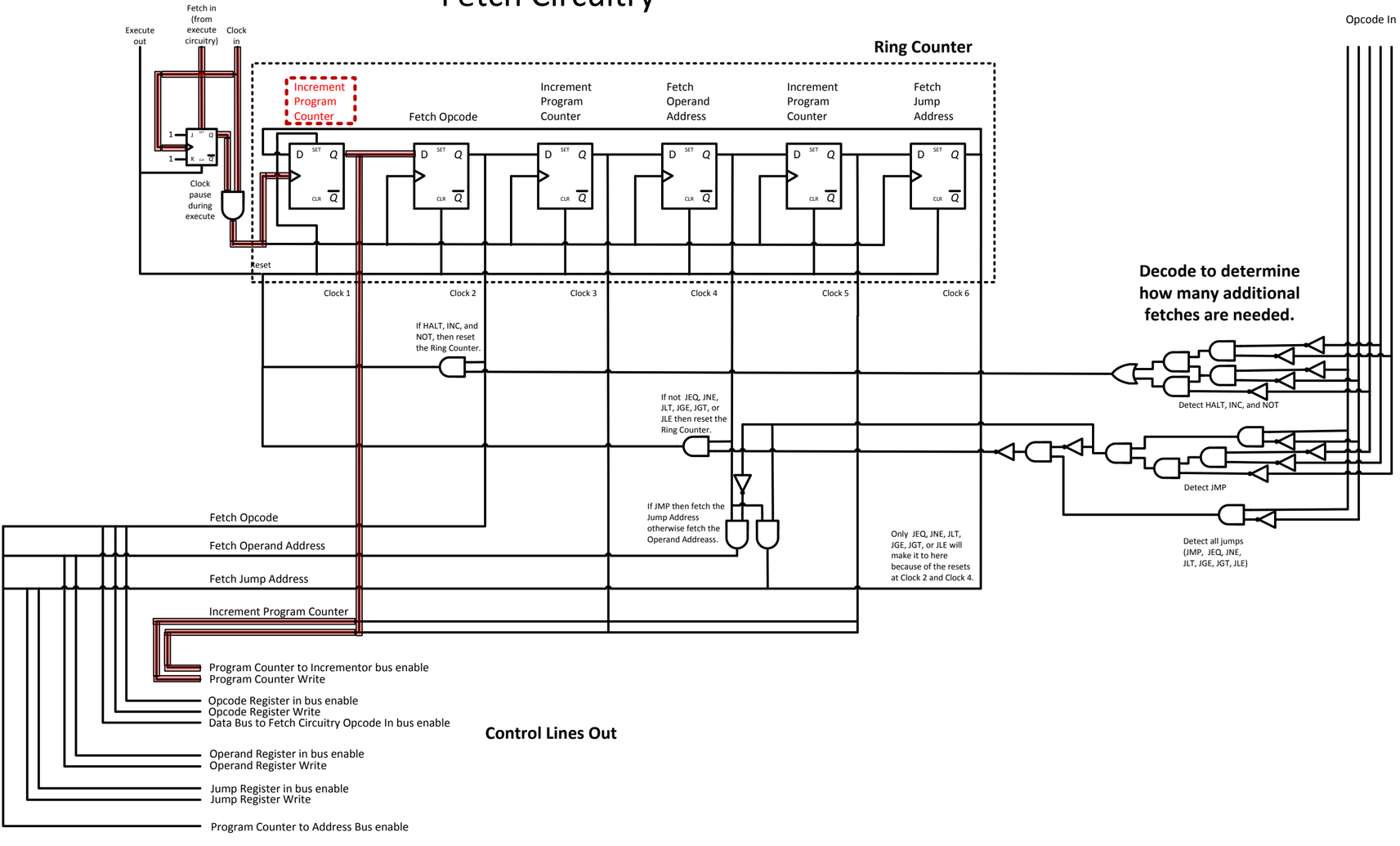
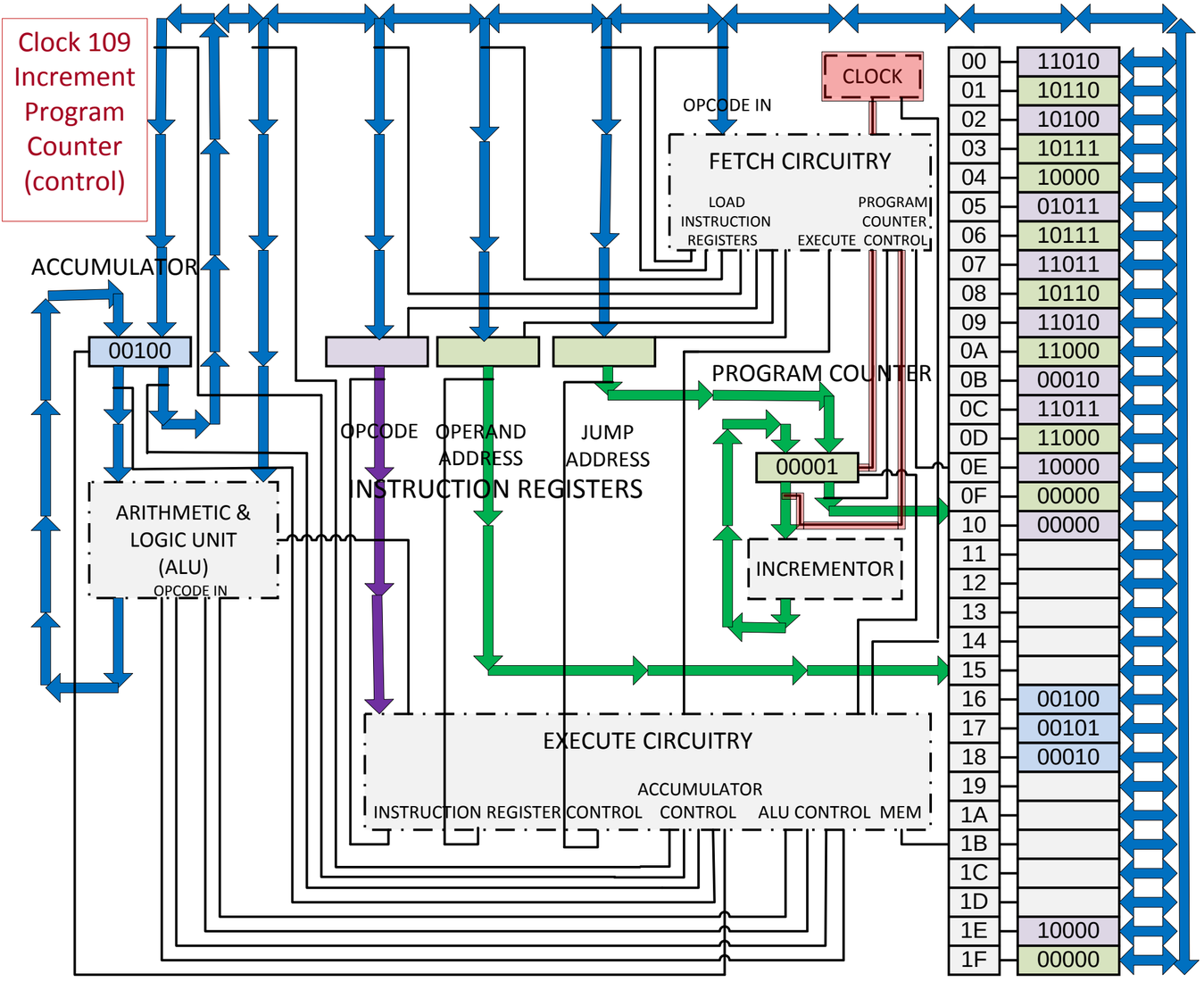


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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

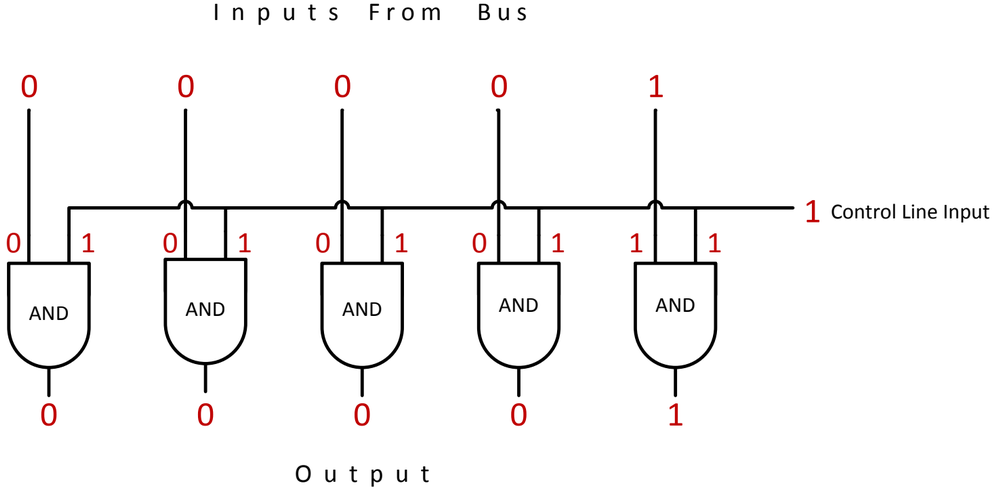
Fetch Circuitry





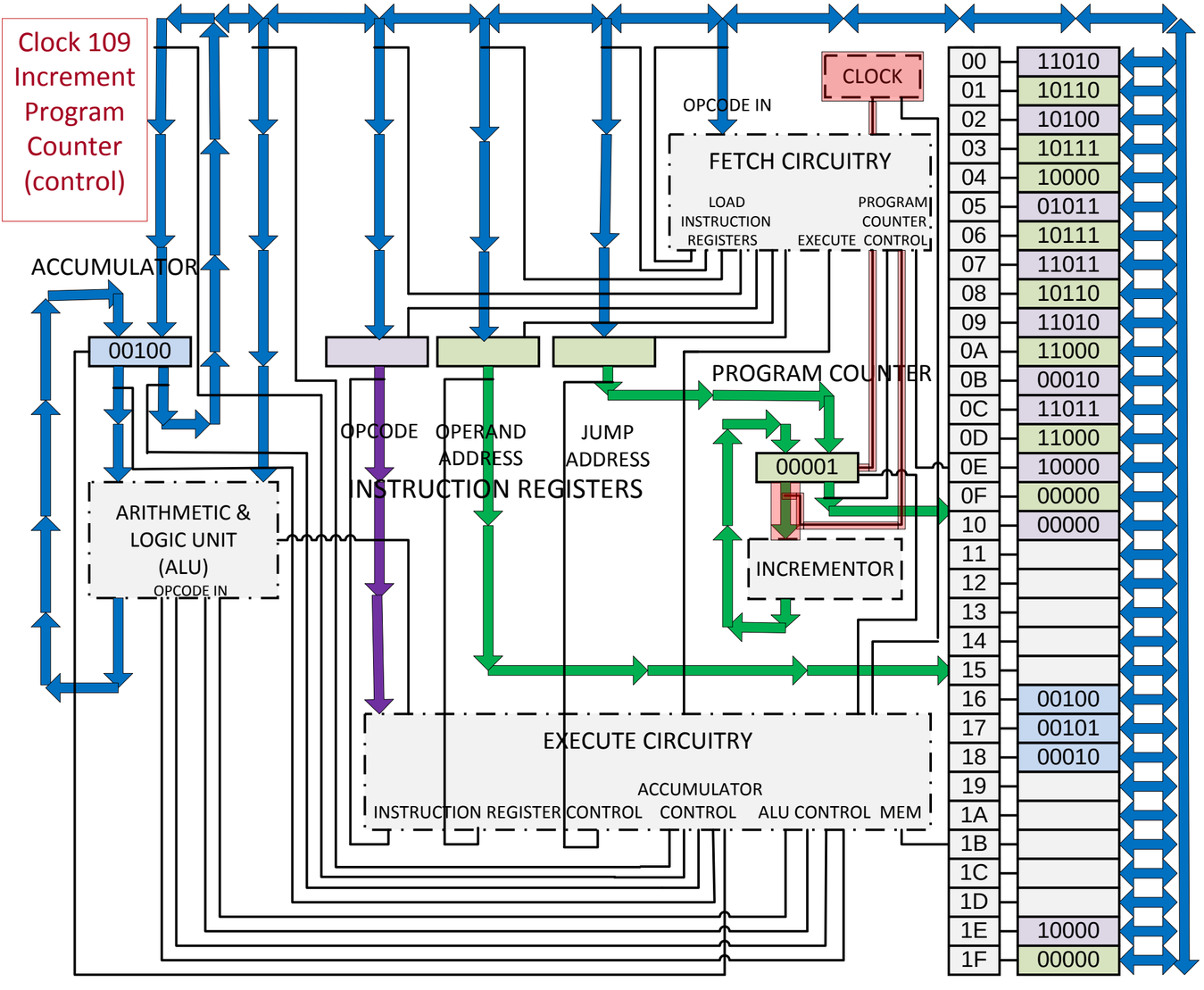
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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	4	# 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	2	# 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

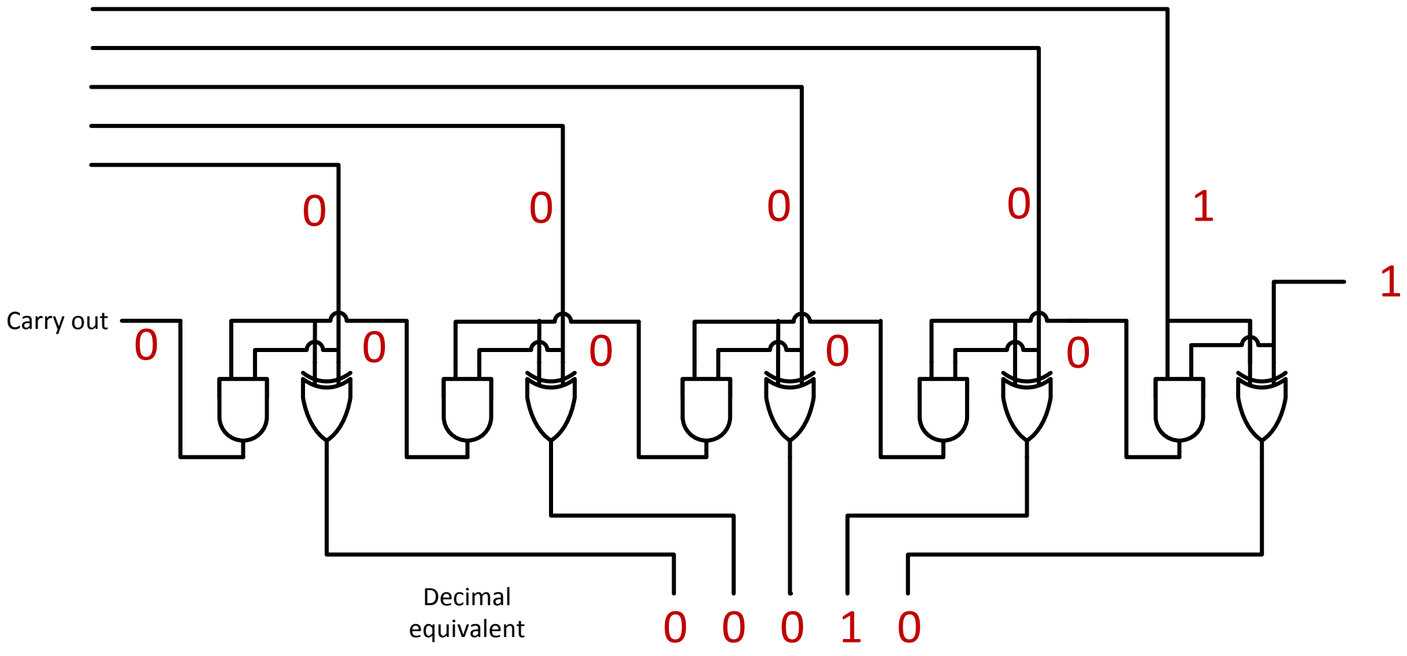
INC opcode

00001 + 00001

1 + 1

Left bus into ALU

0 0 0 0 1

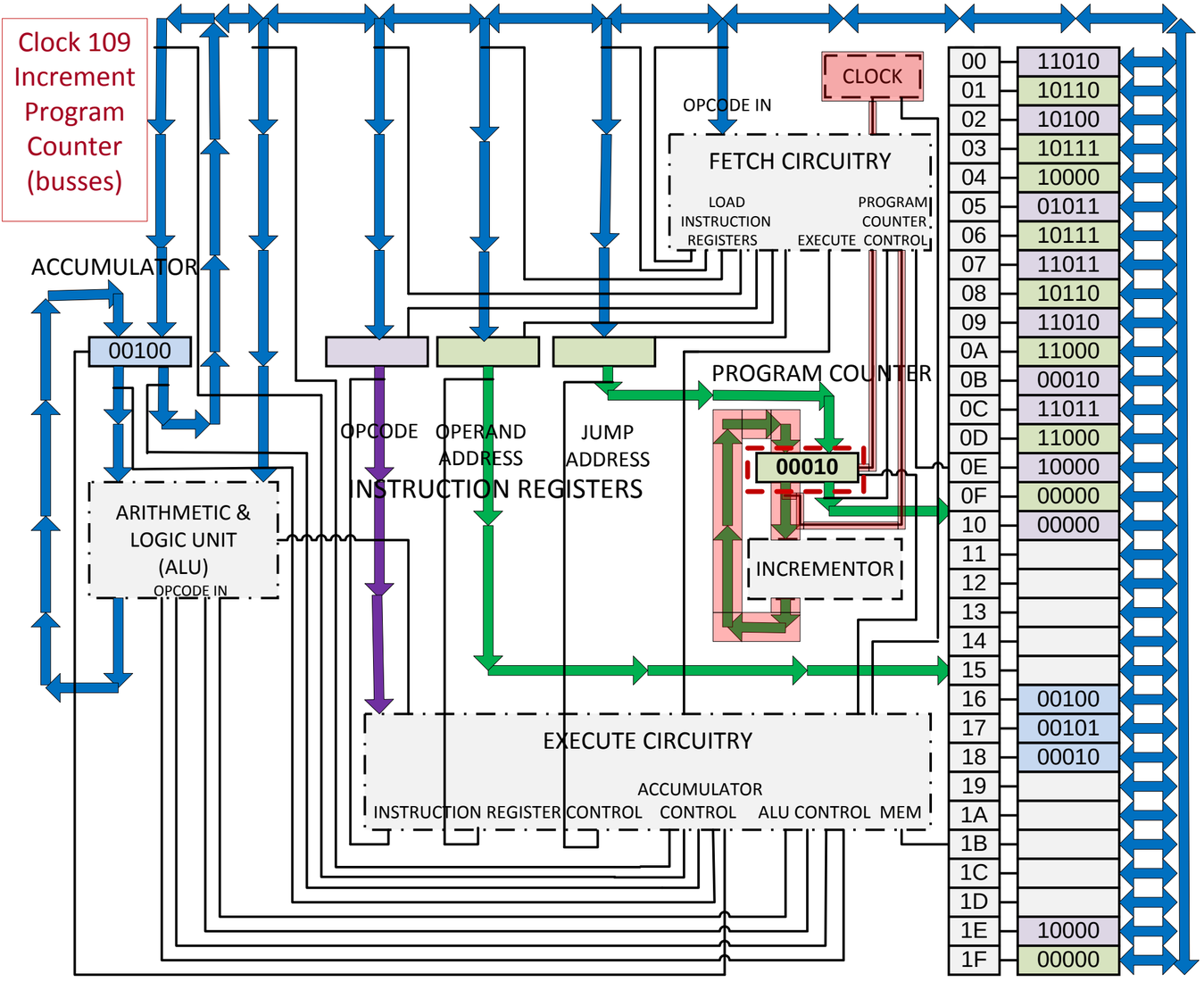


Decimal equivalent

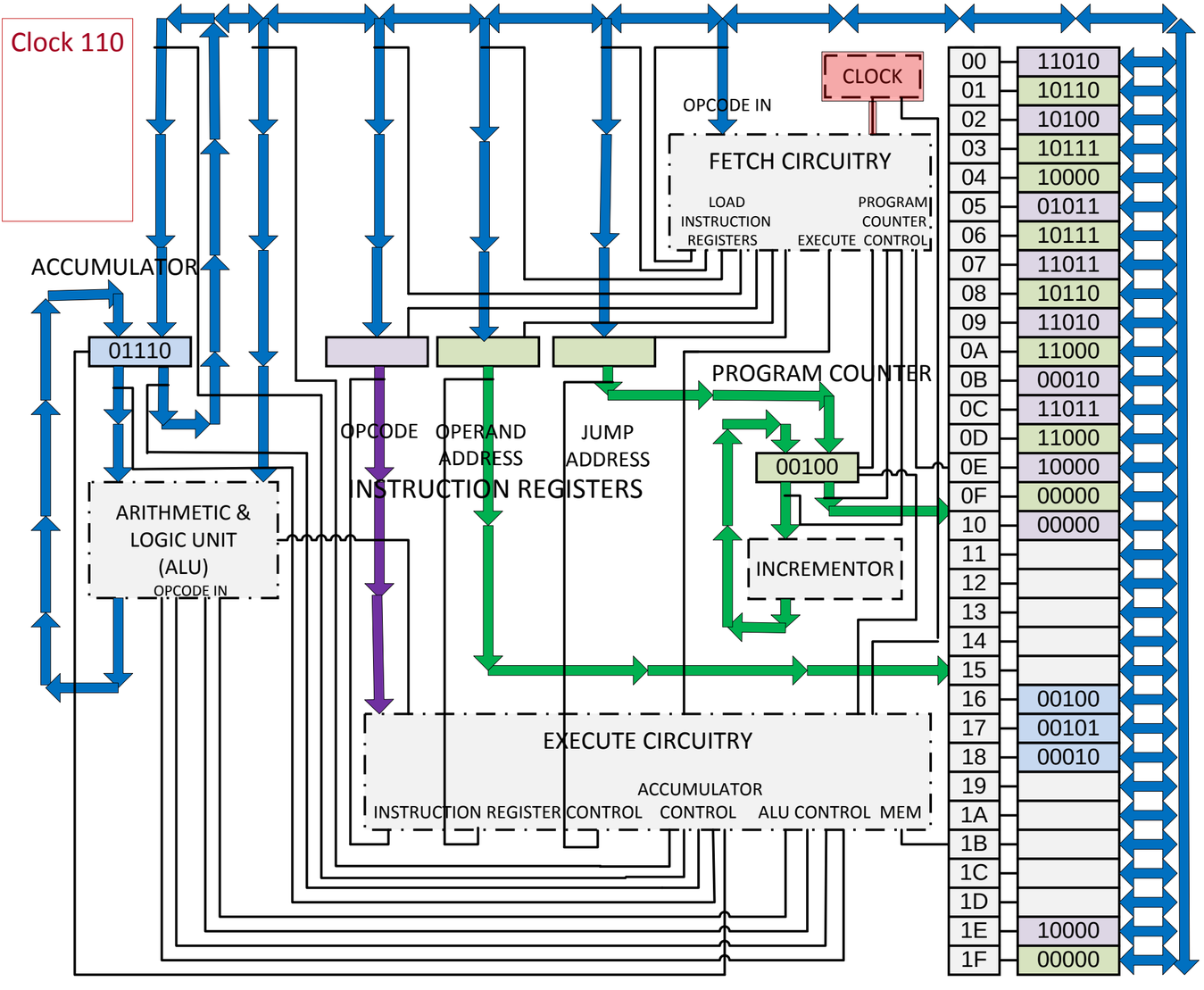
Output

Input number + 1

```
  0 0 0 0 1        1
+ 0 0 0 0 1    +   1
-----------    -----
  0 0 0 1 0        2
```

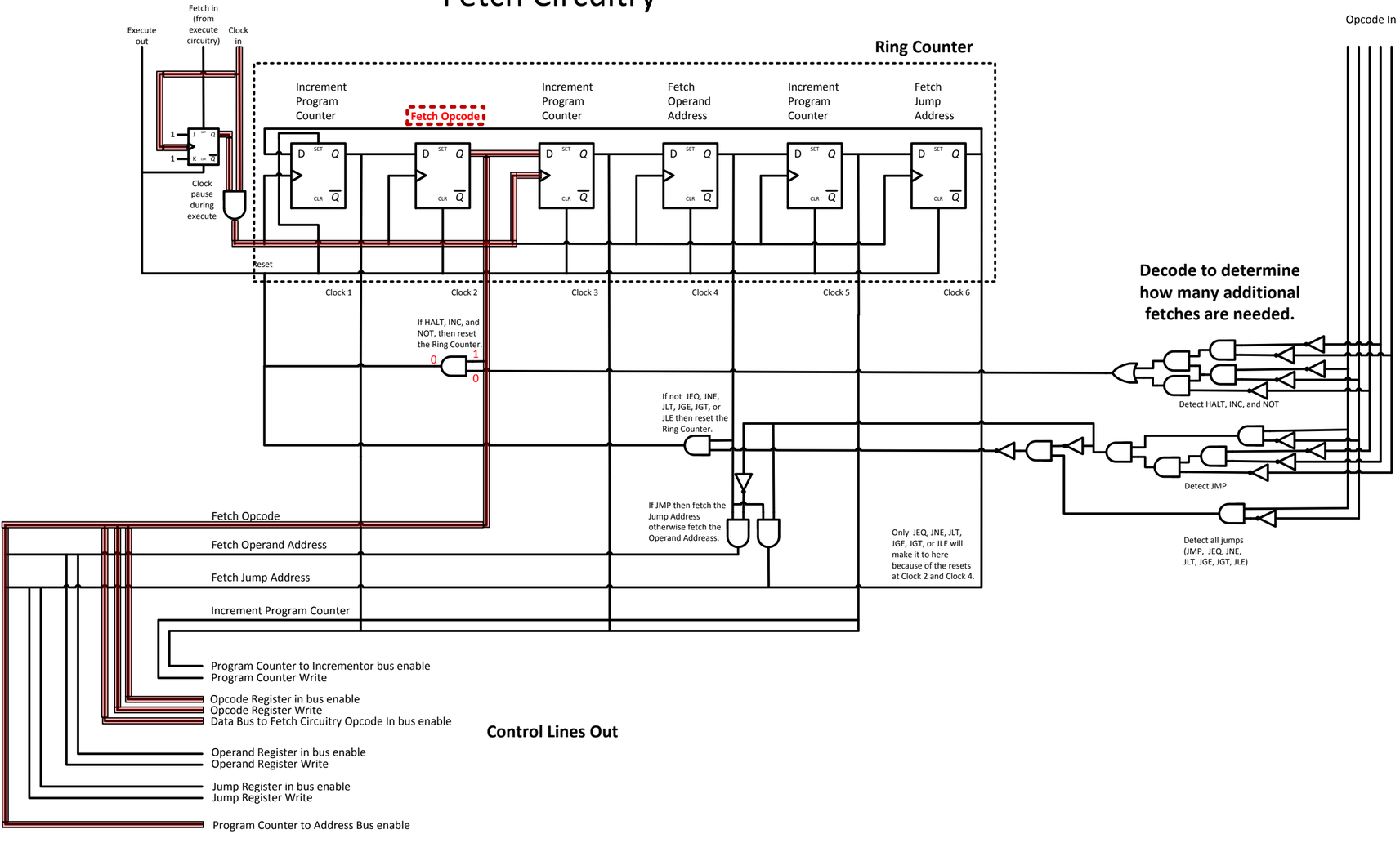


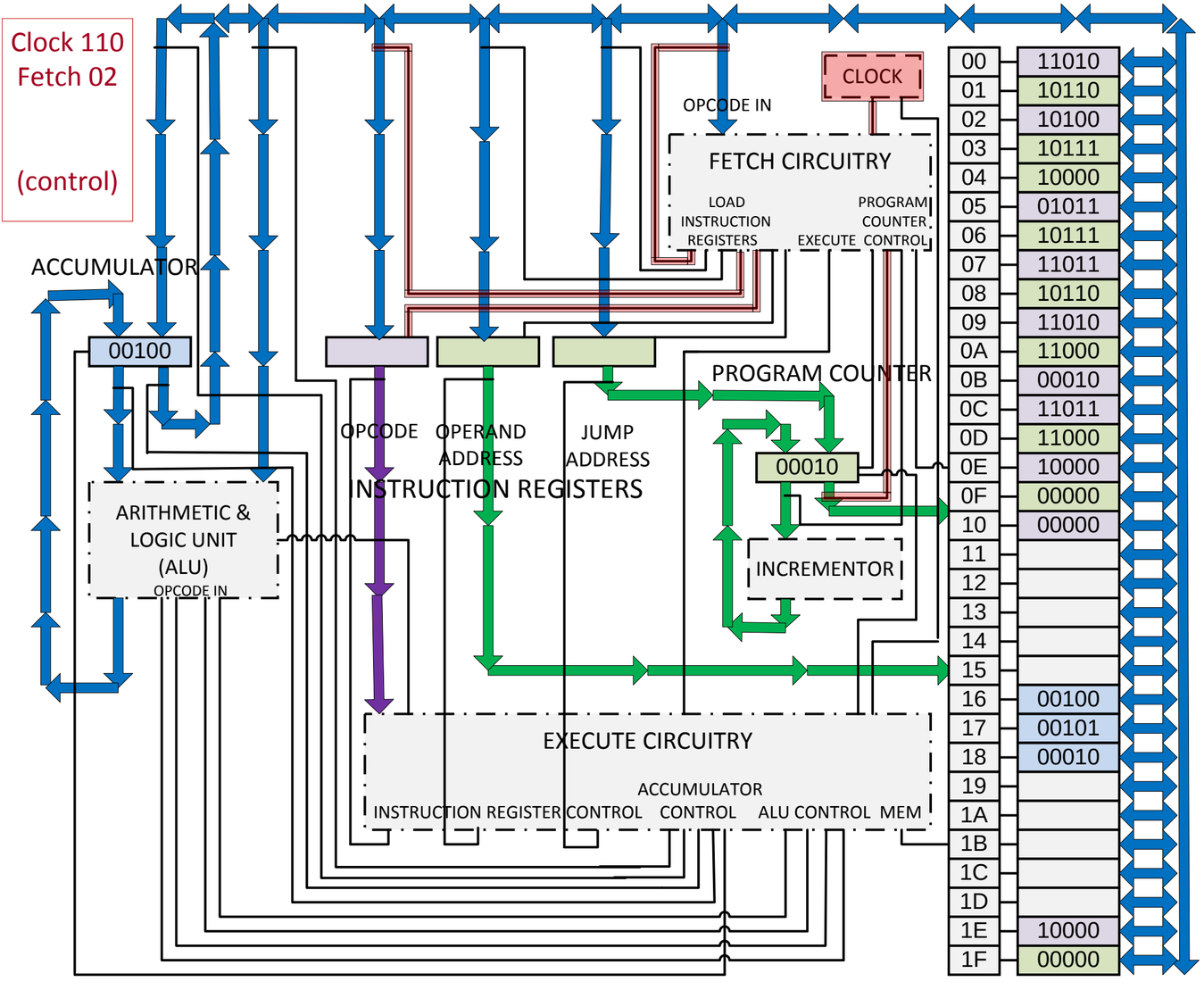
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



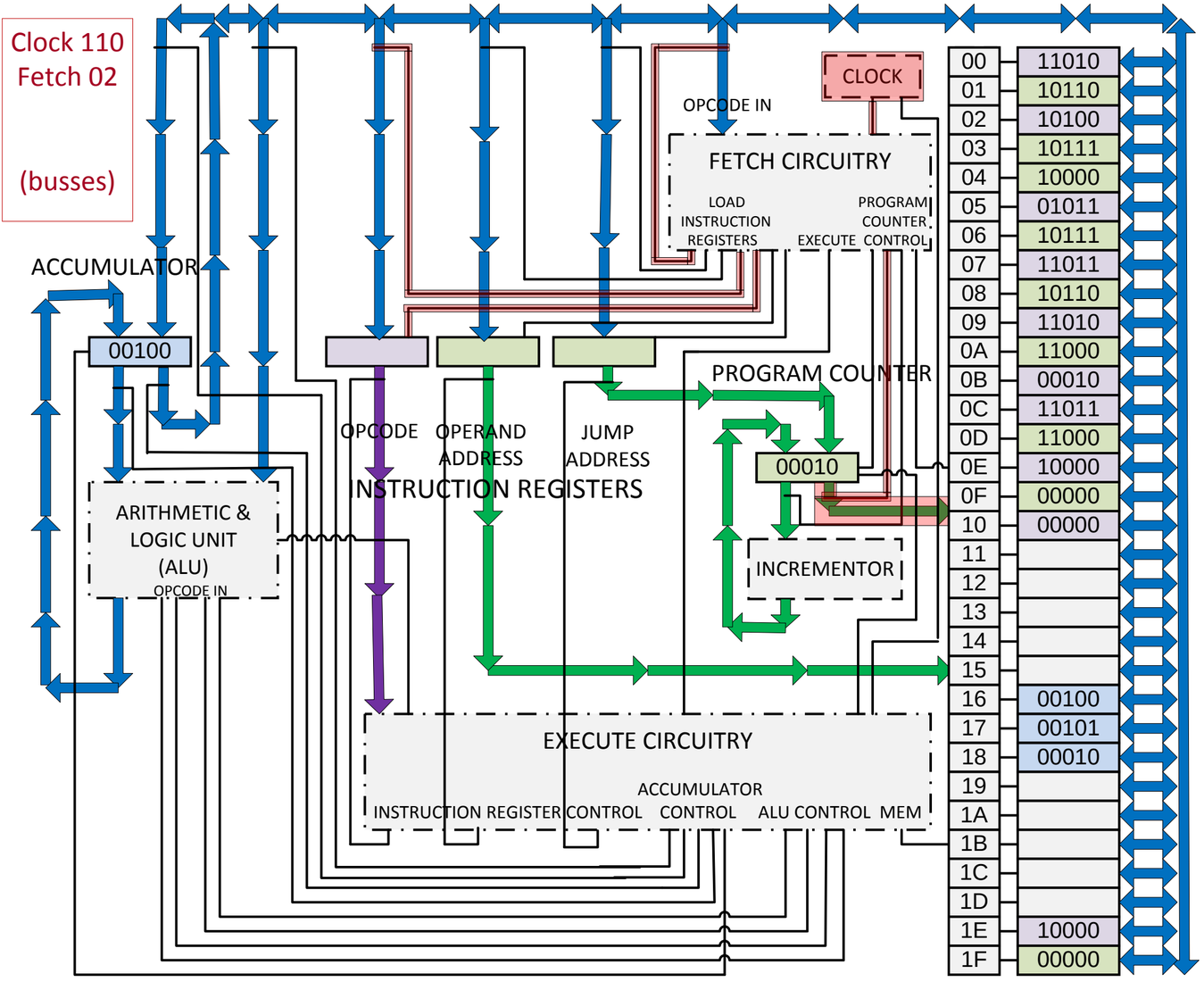
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



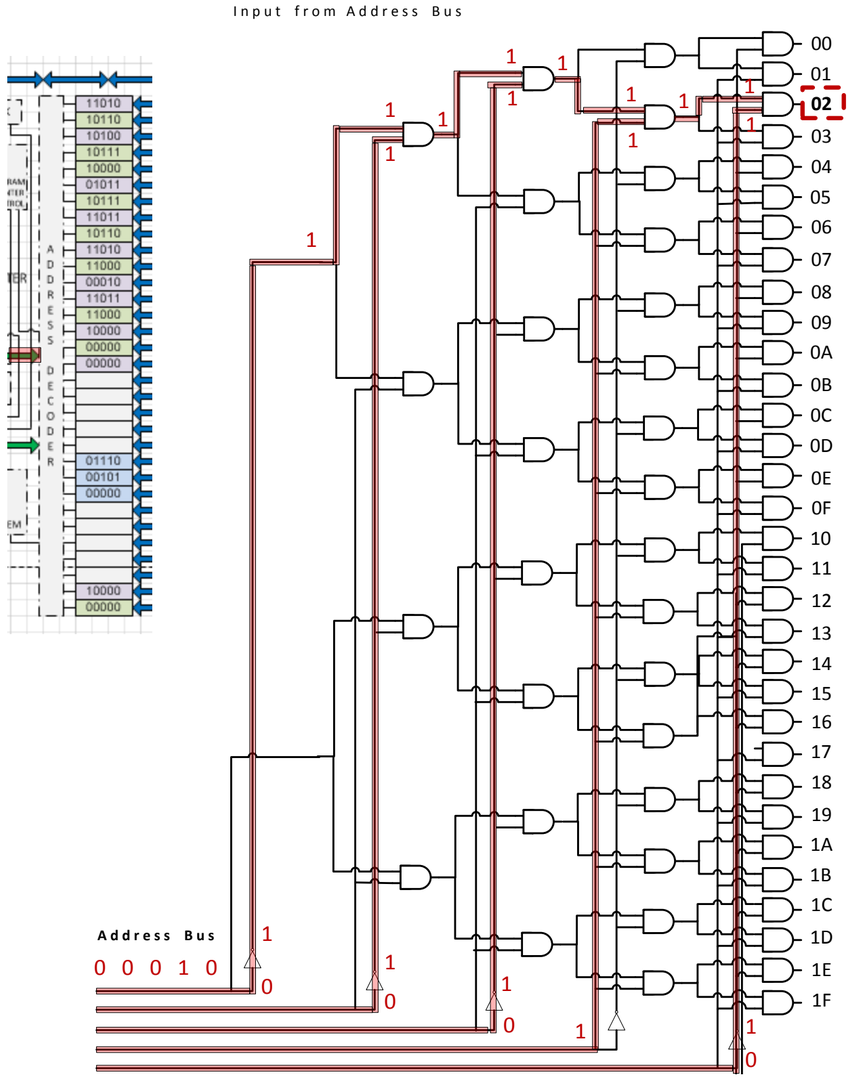


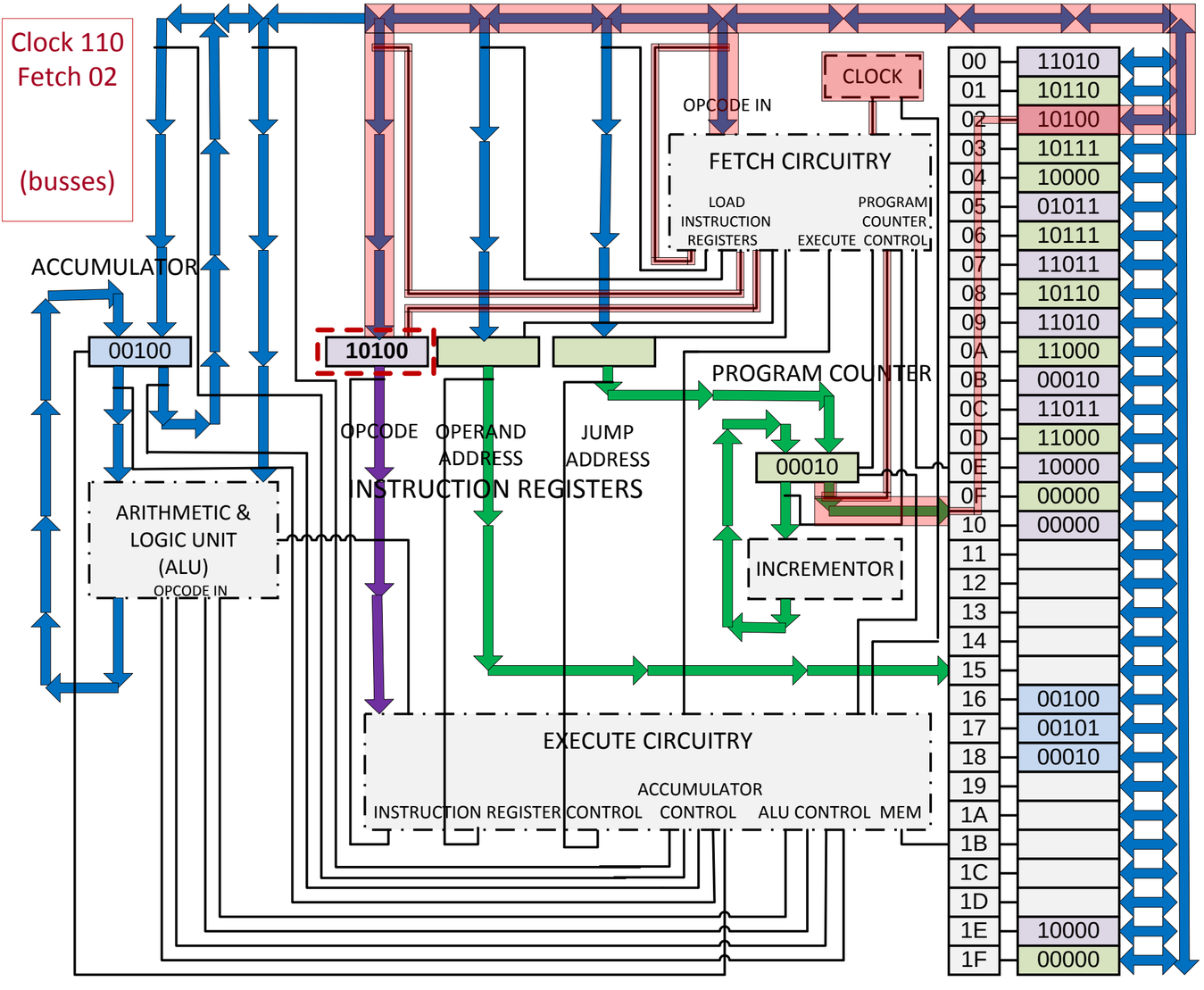
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



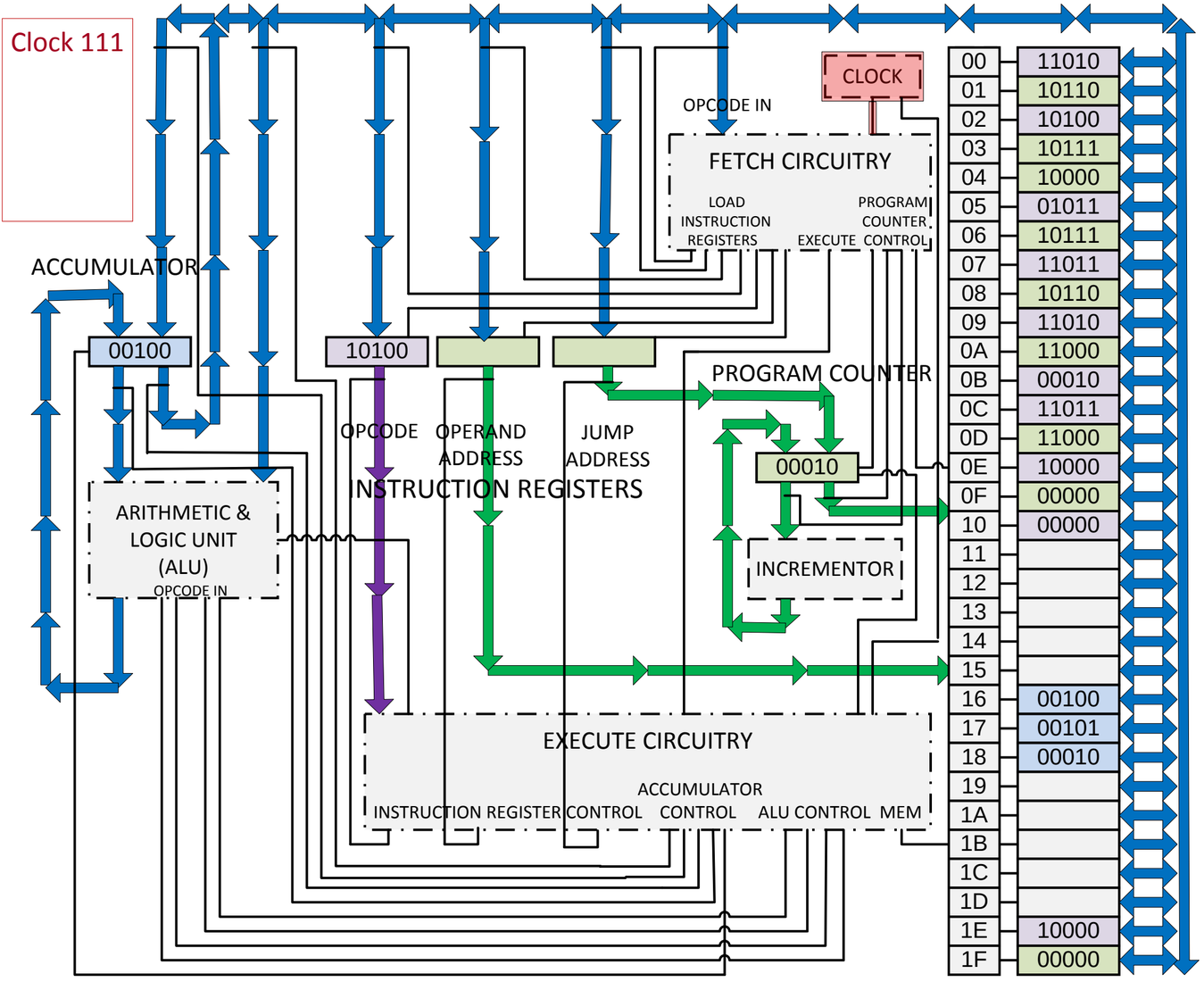
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



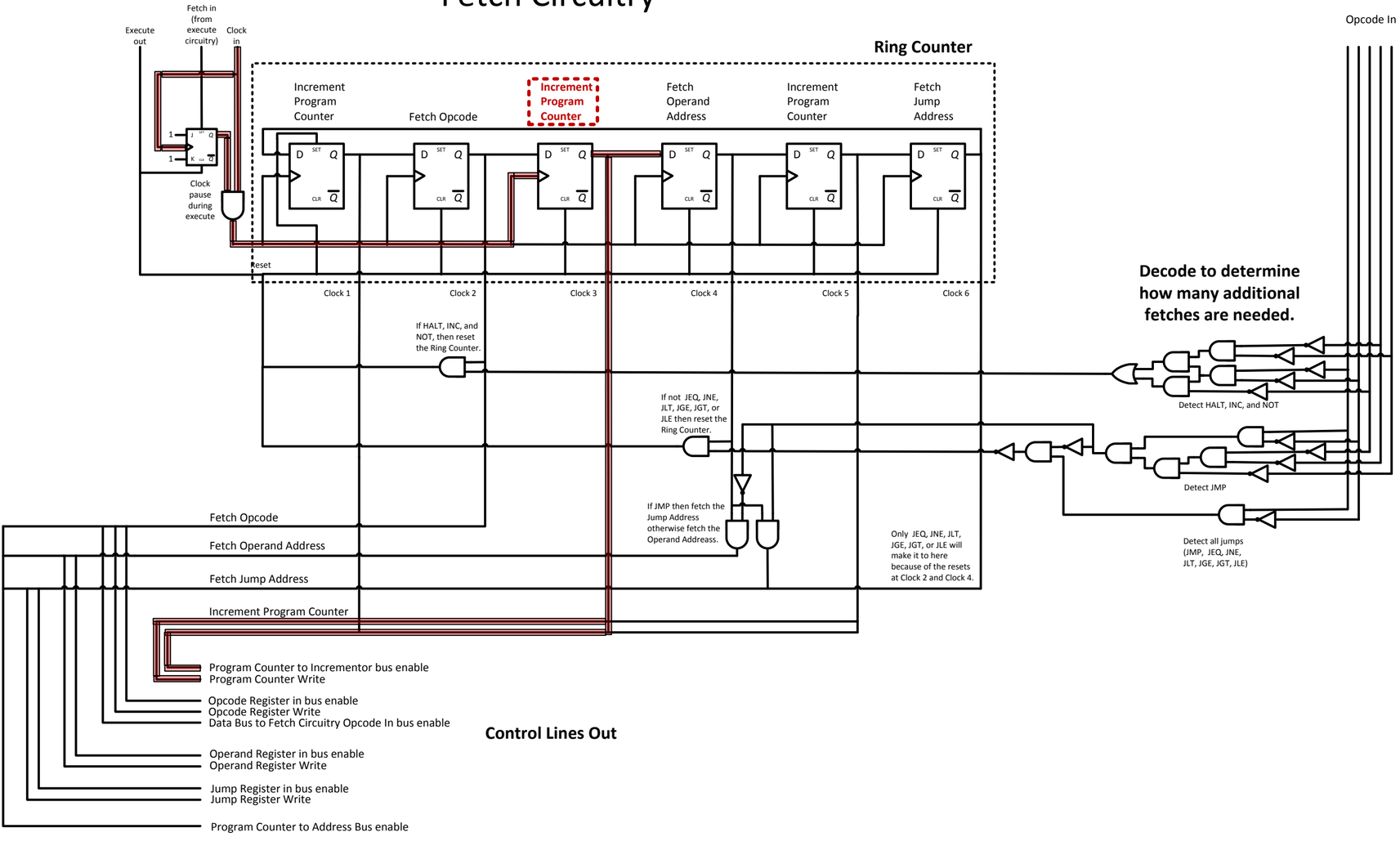


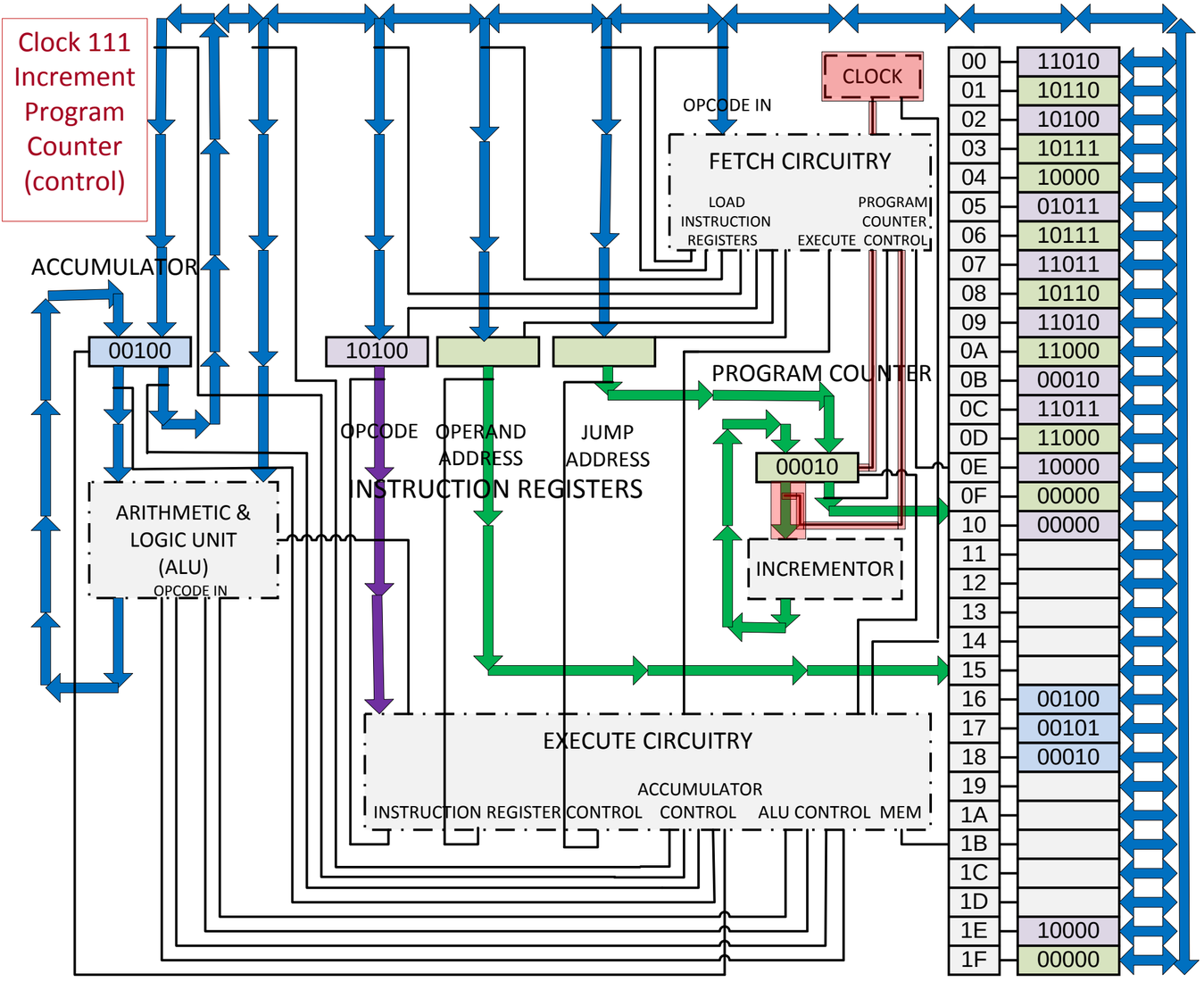
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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	4	# 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	2	# 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	4	# 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	2	# 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

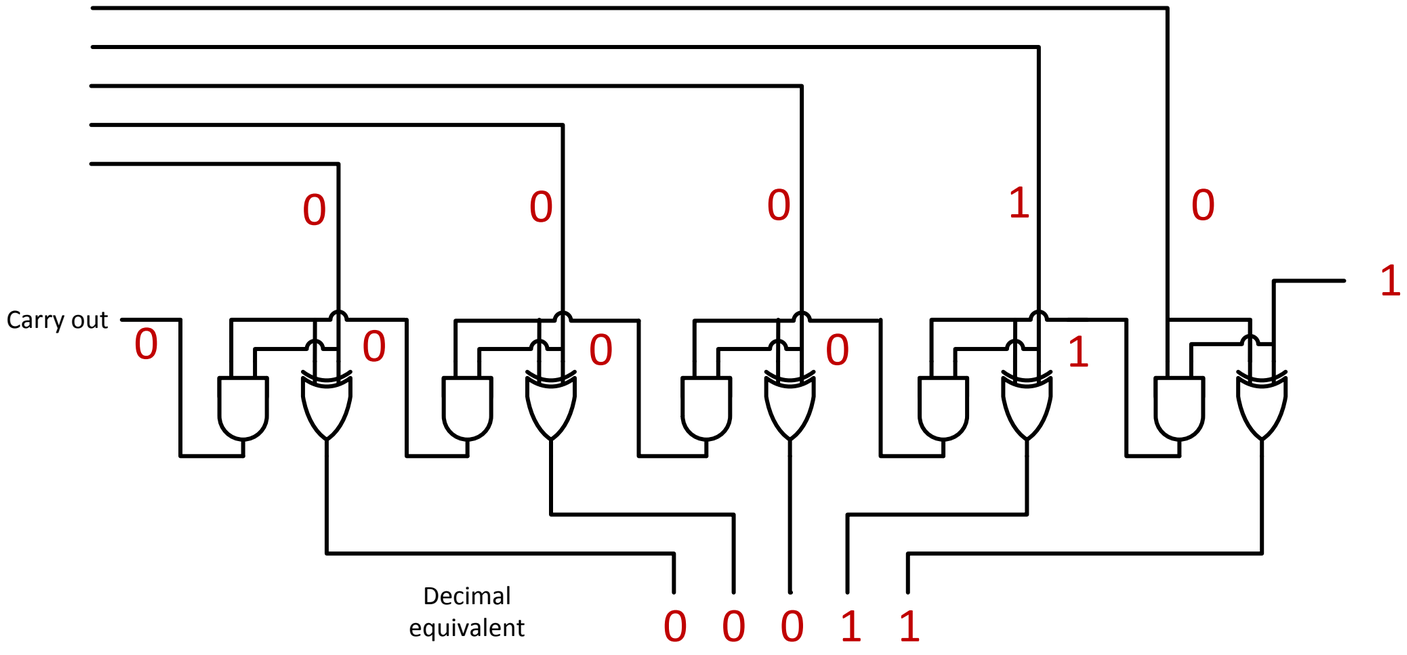
INC opcode

00010 + 00001

2 + 1

Left bus into ALU

0 0 0 1 0

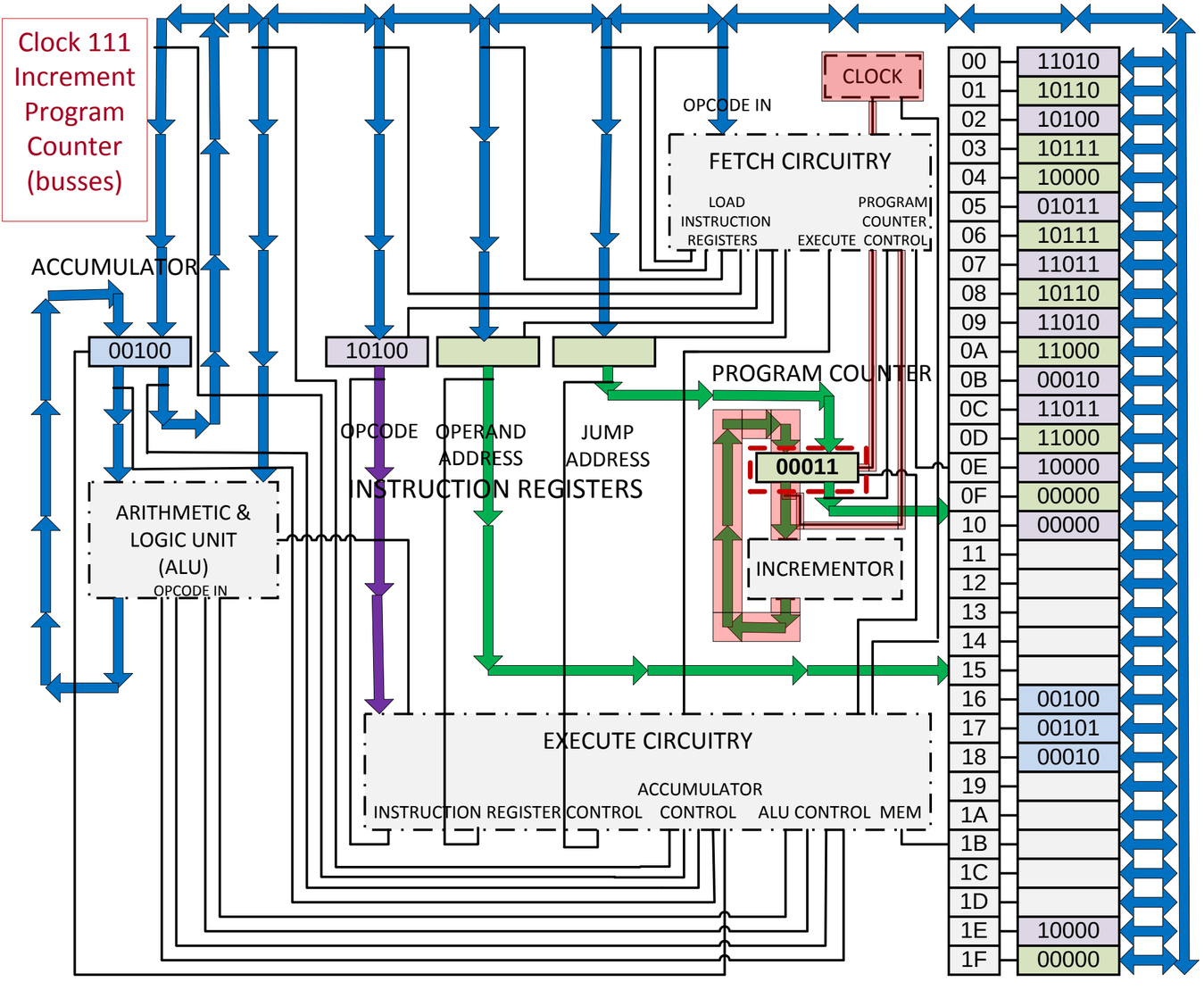


Decimal equivalent

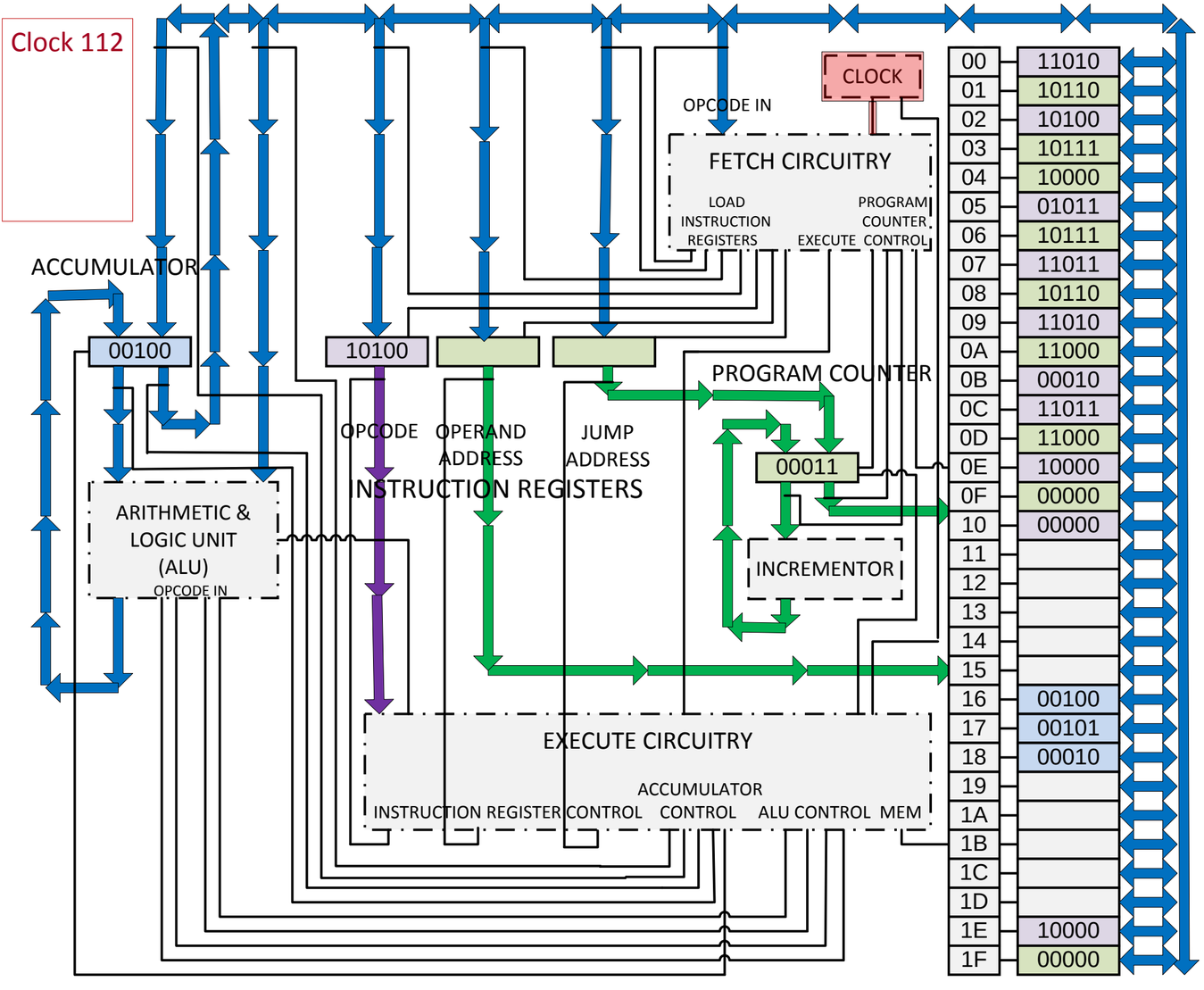
0 0 0 1 1 — 2

\+ 0 0 0 0 1 — \+ 1

0 0 0 1 1 — 3

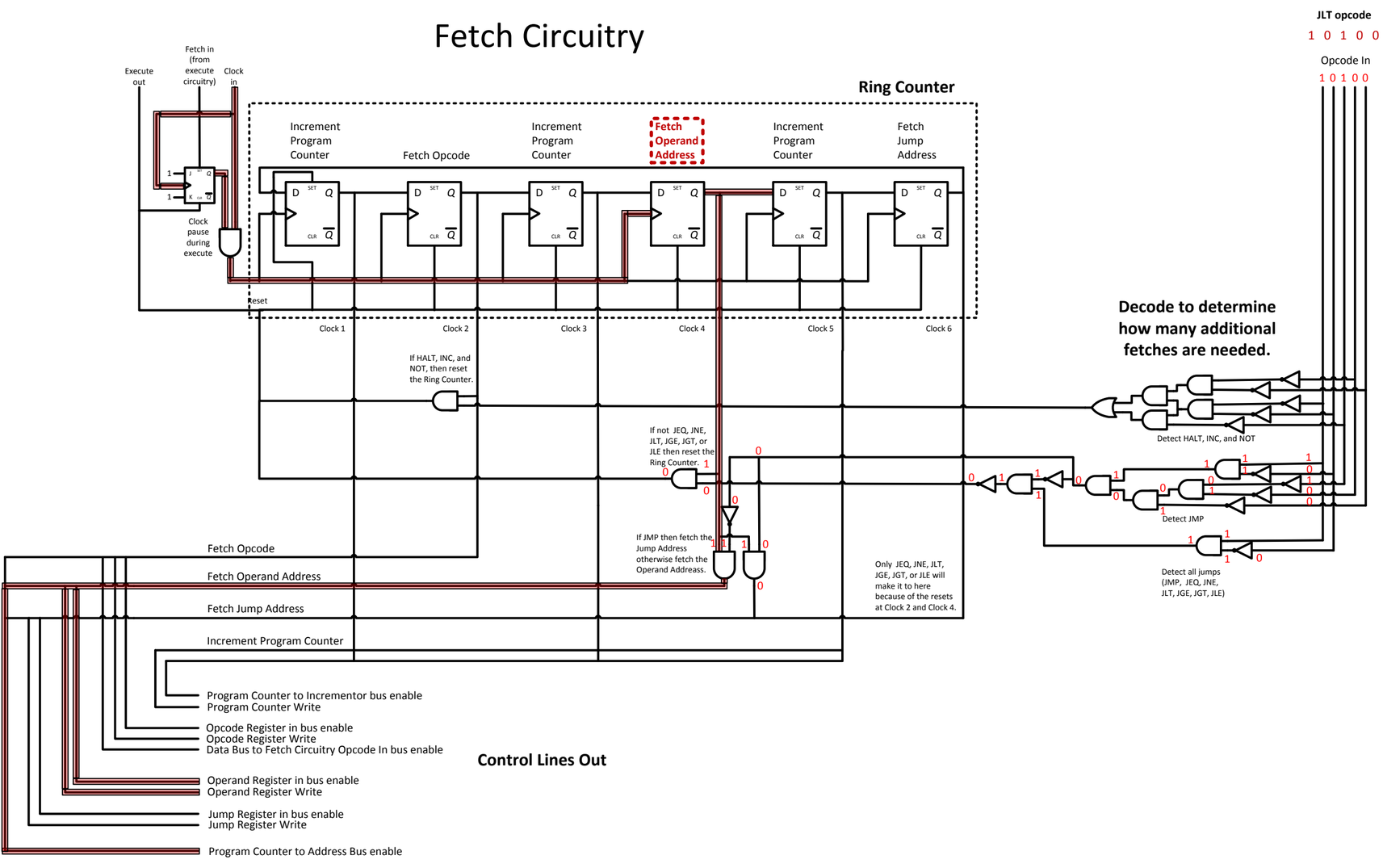


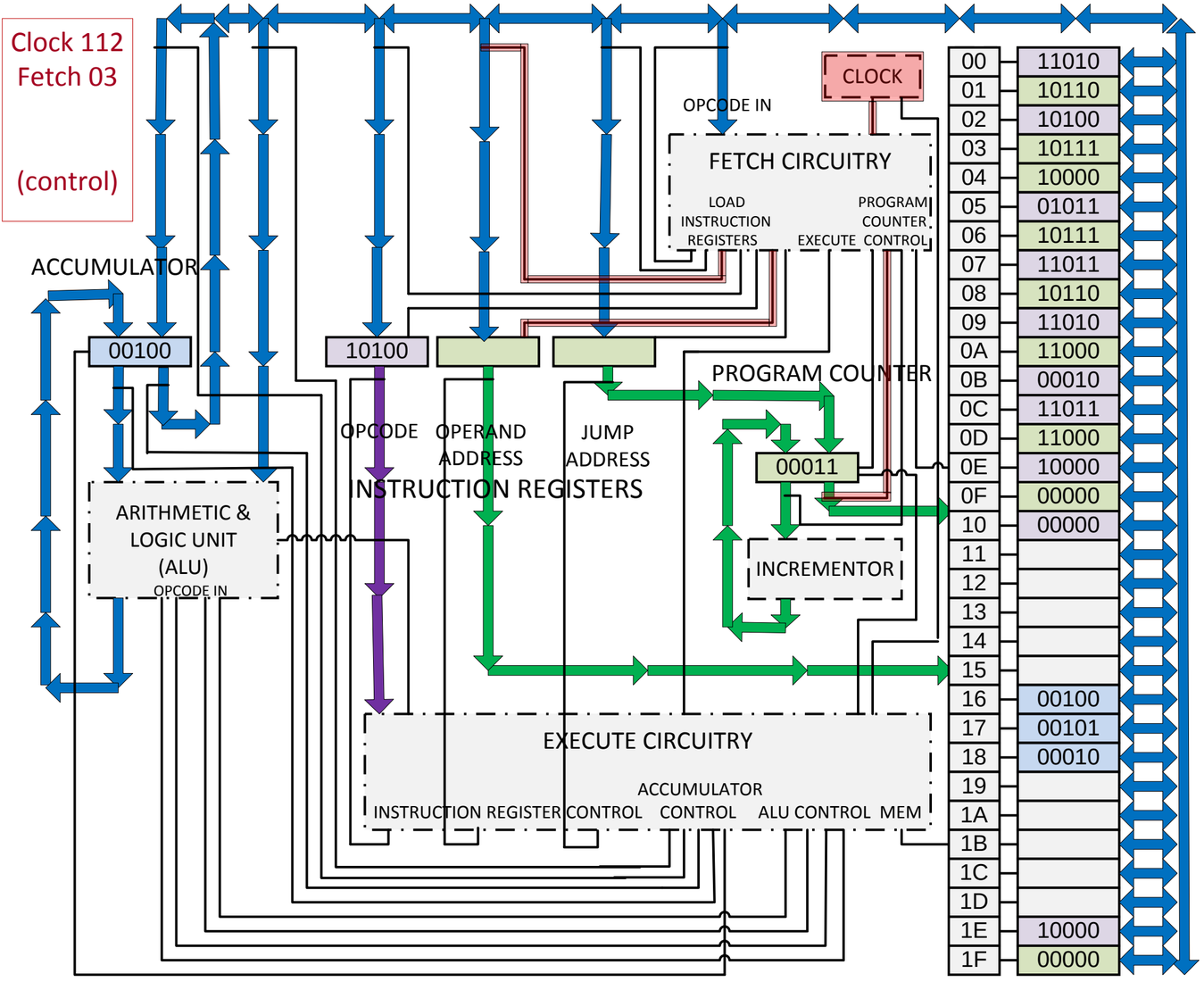
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



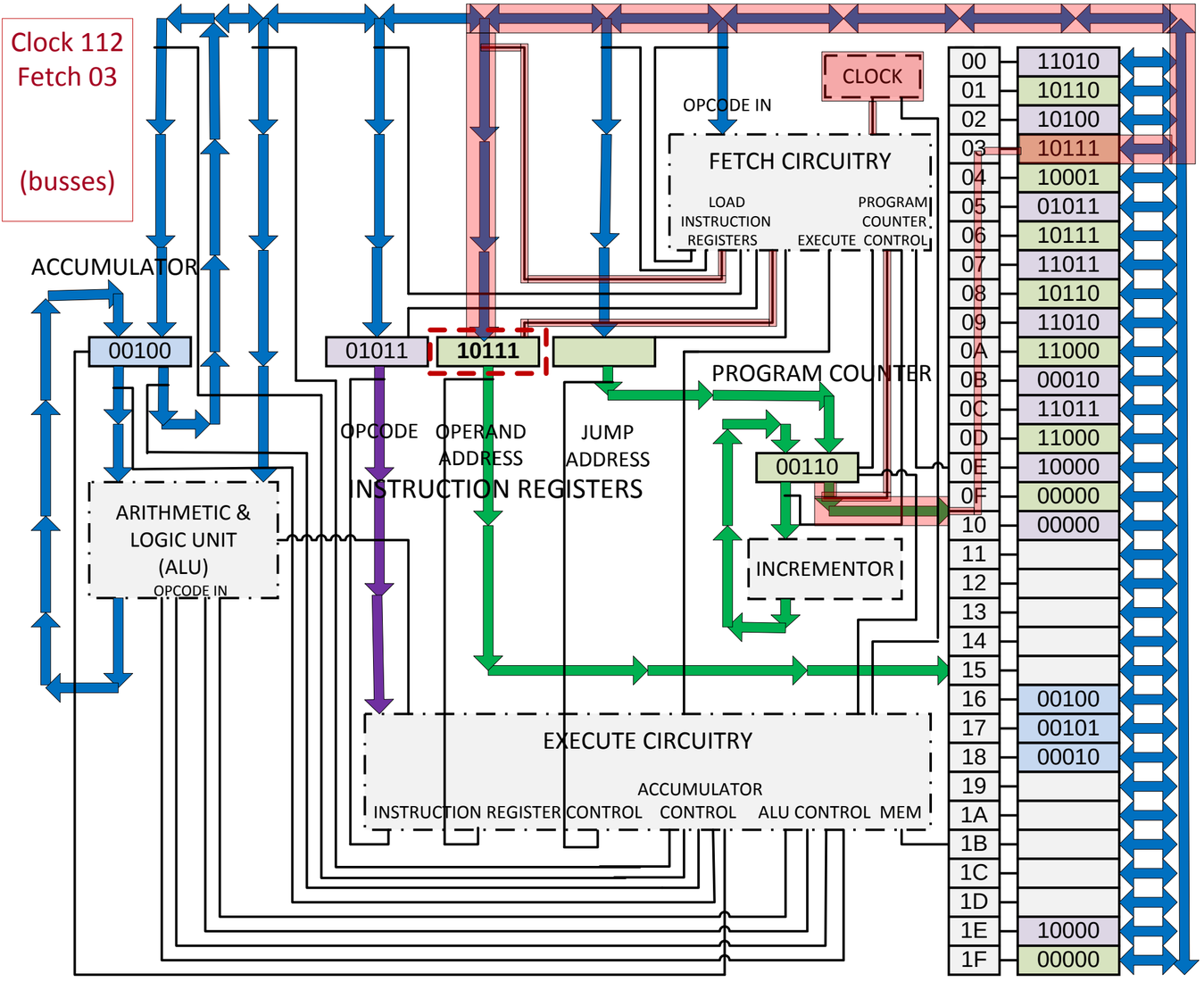
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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

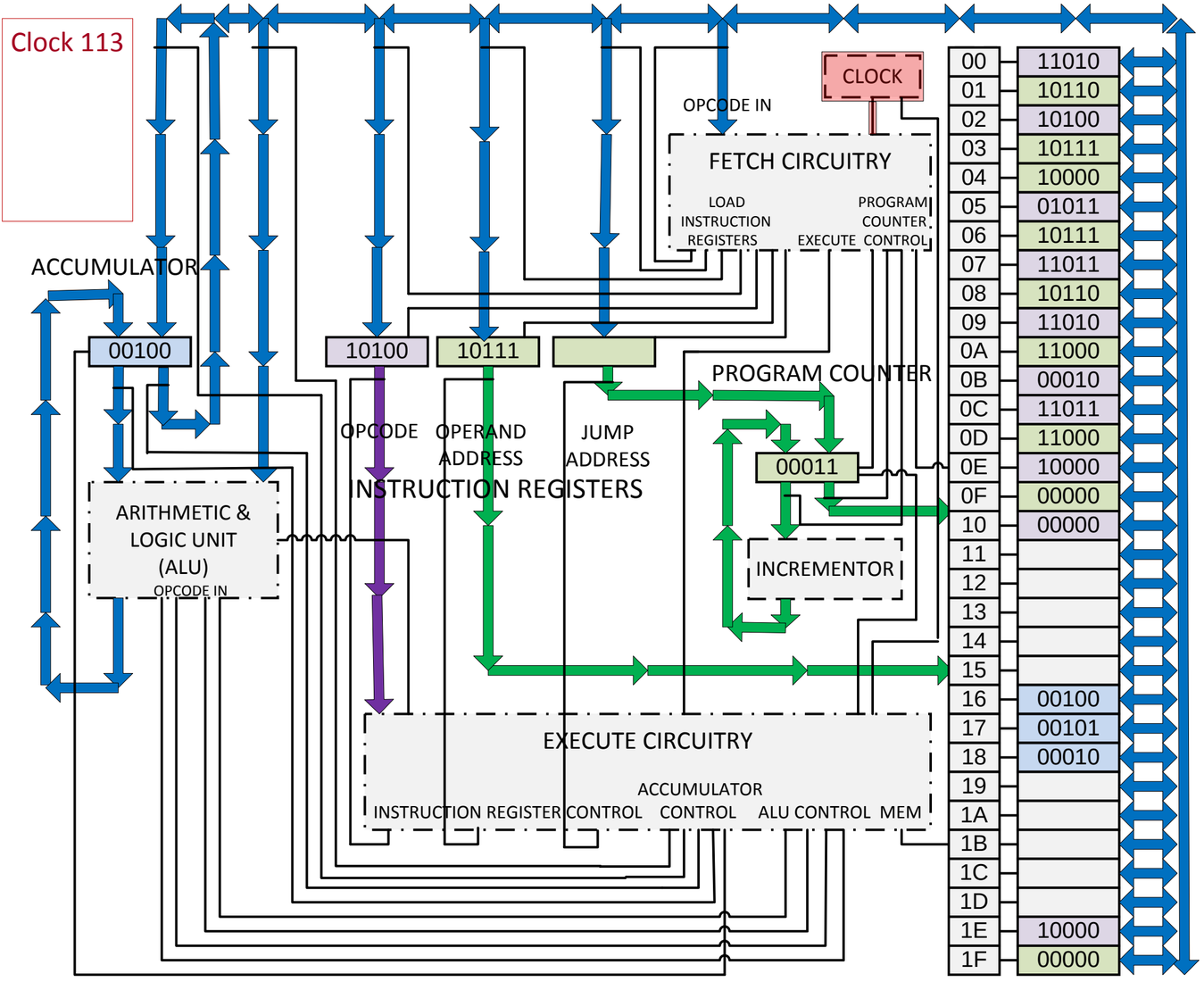




Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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

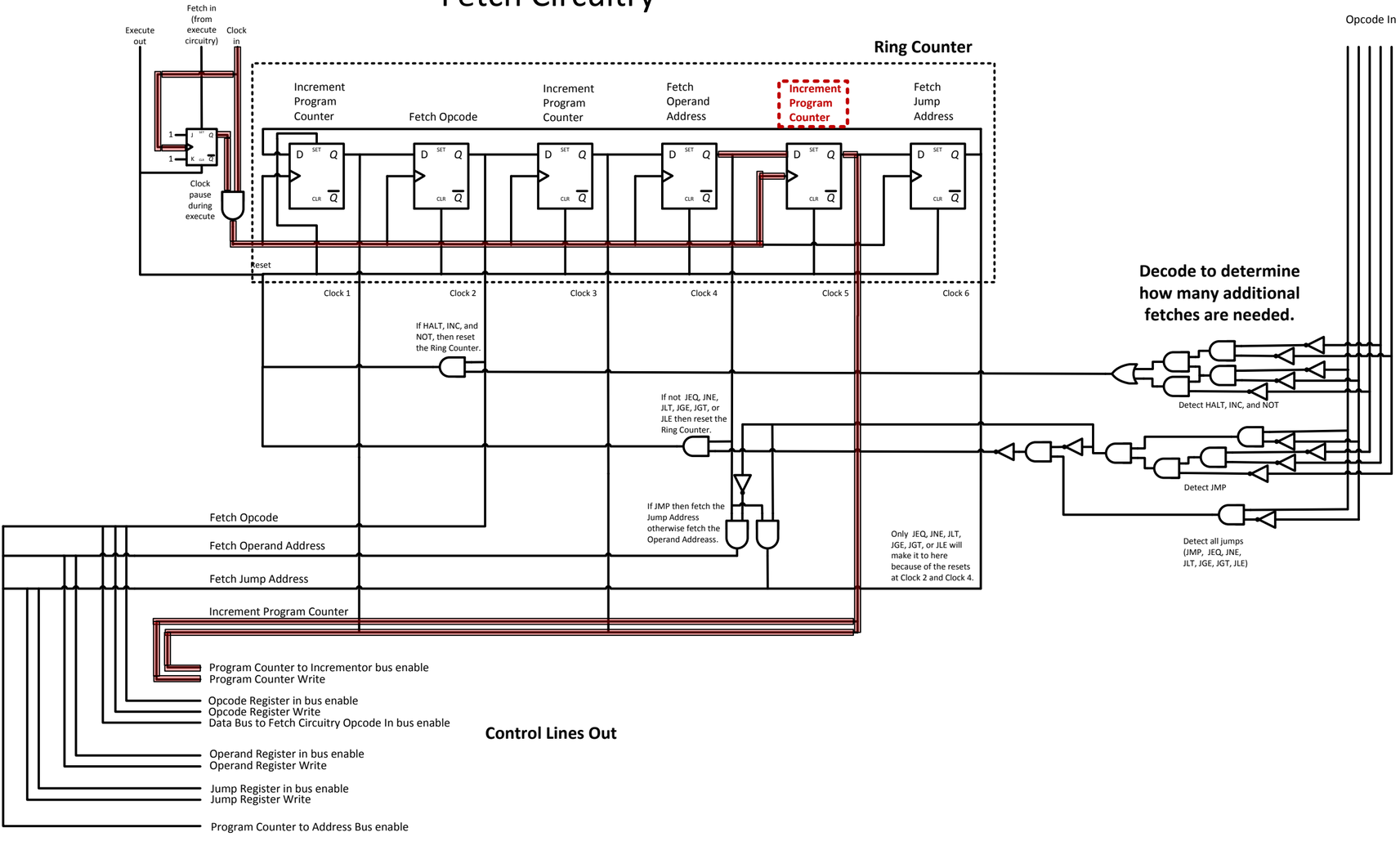


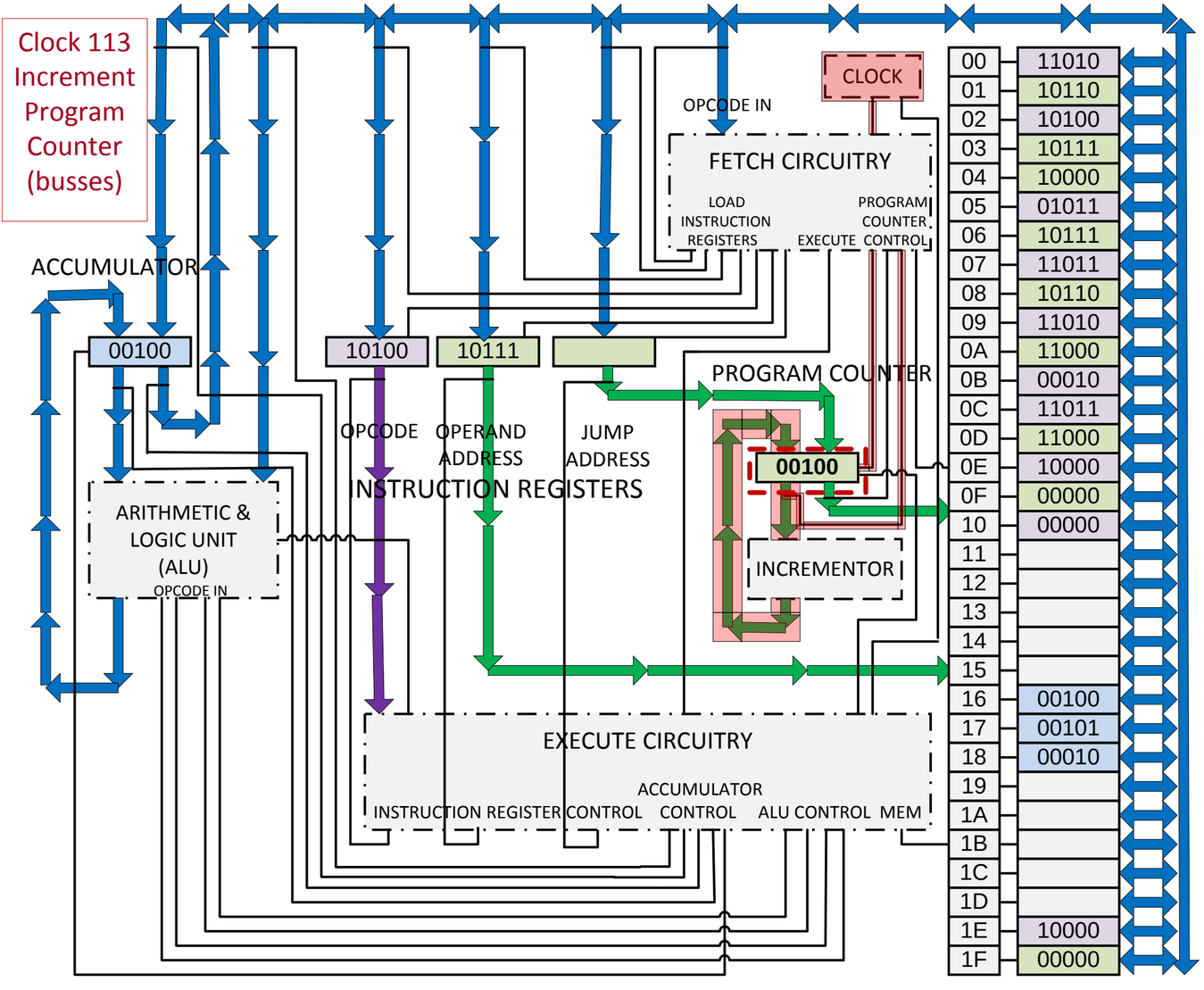
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



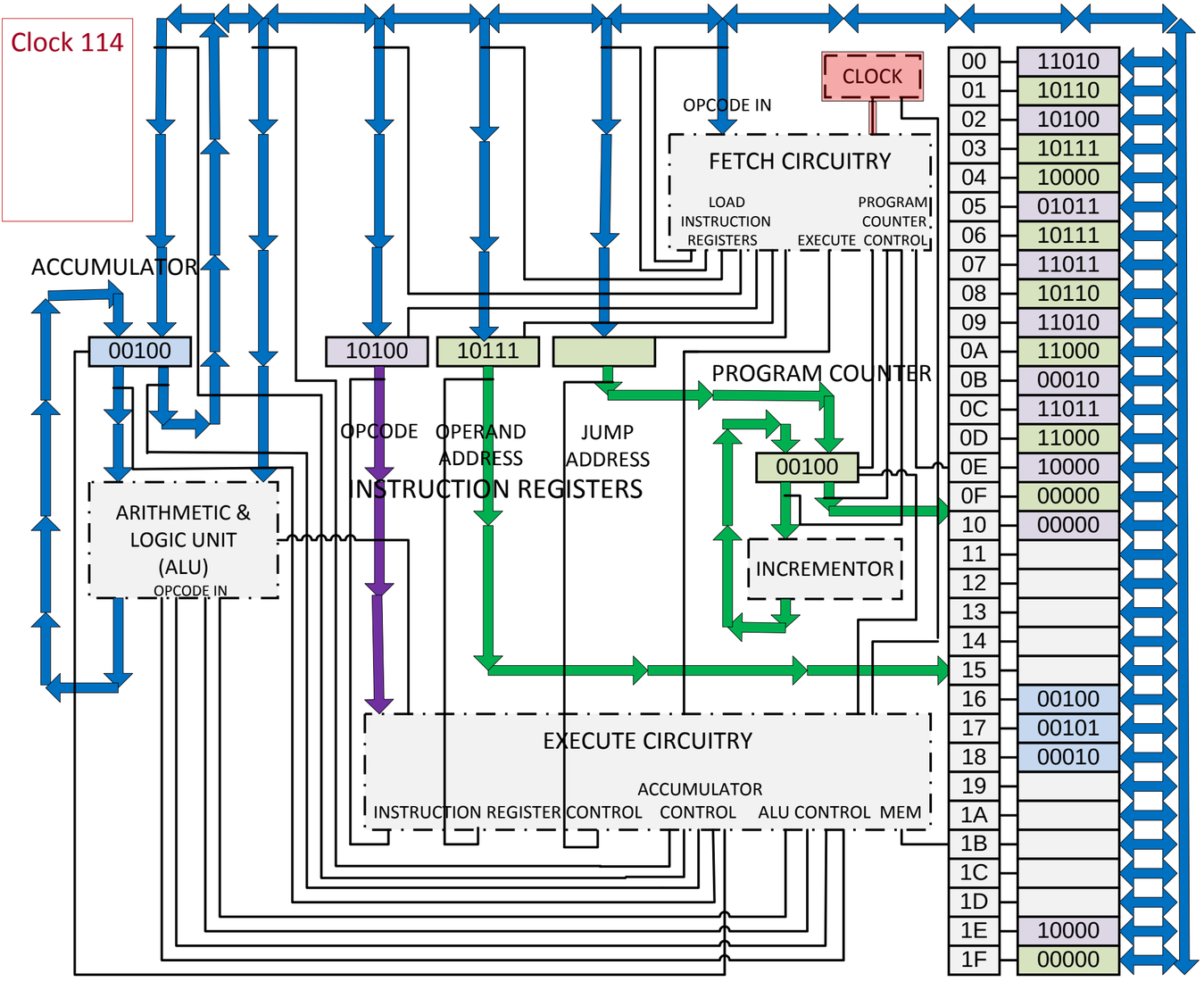
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



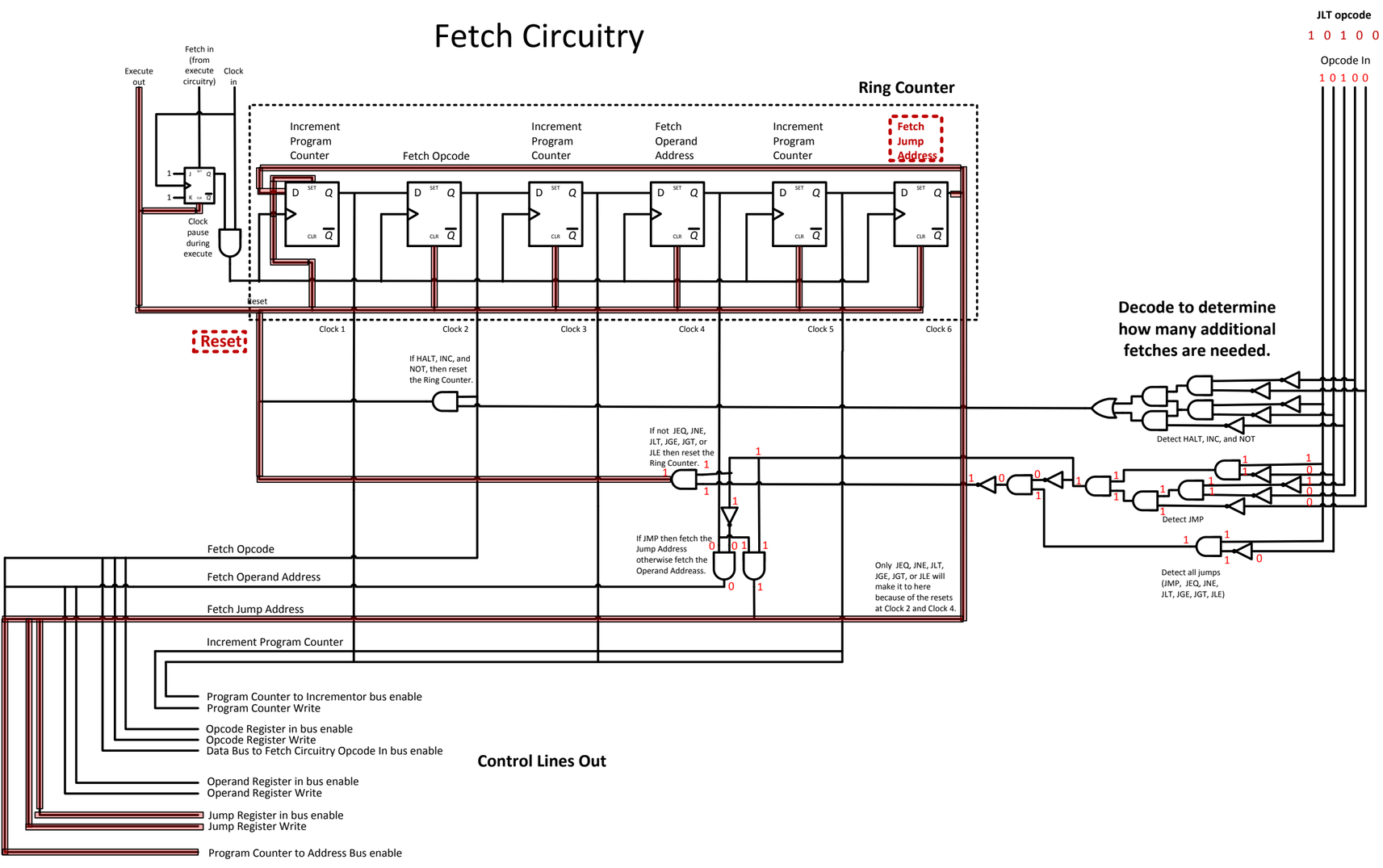


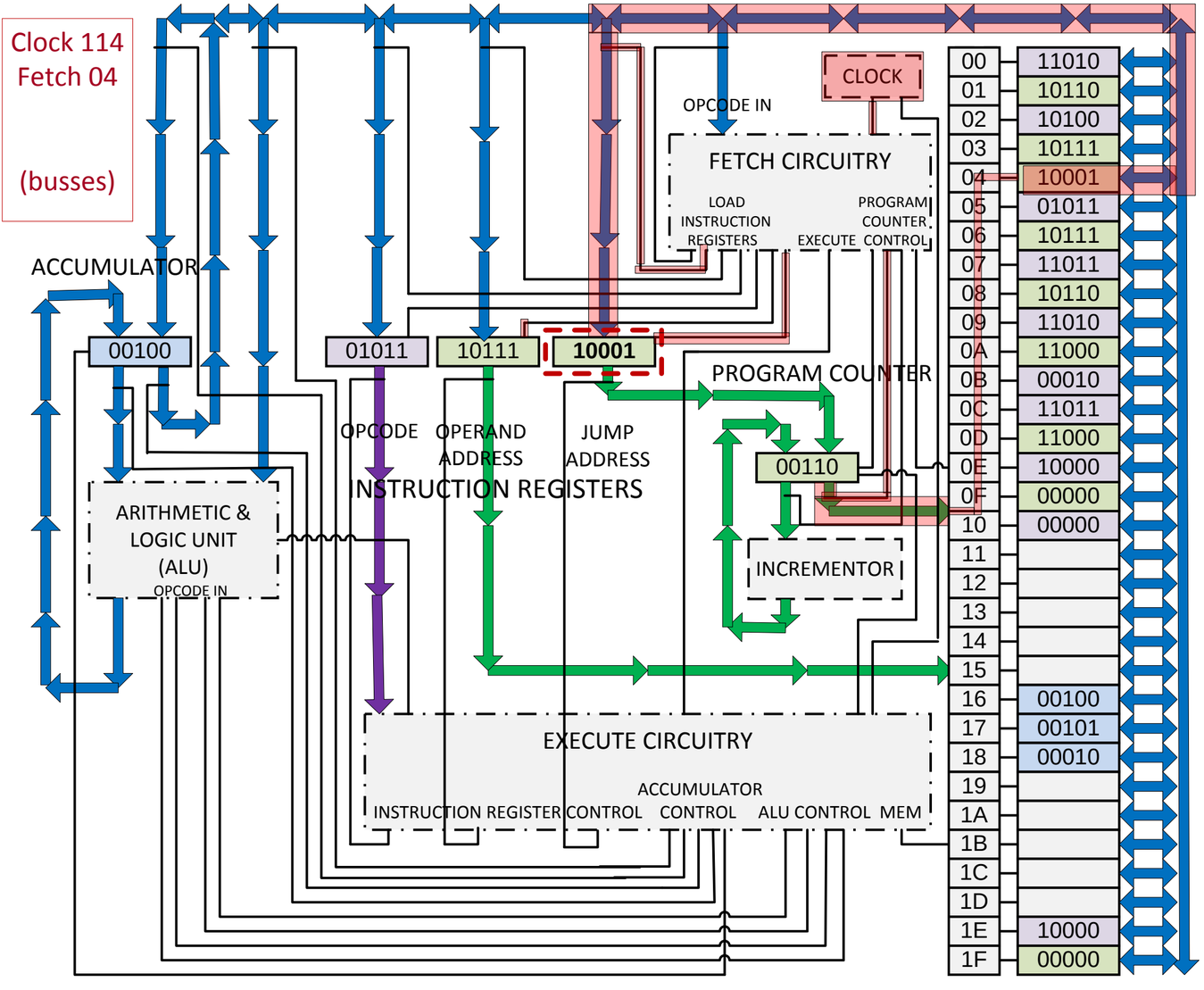
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



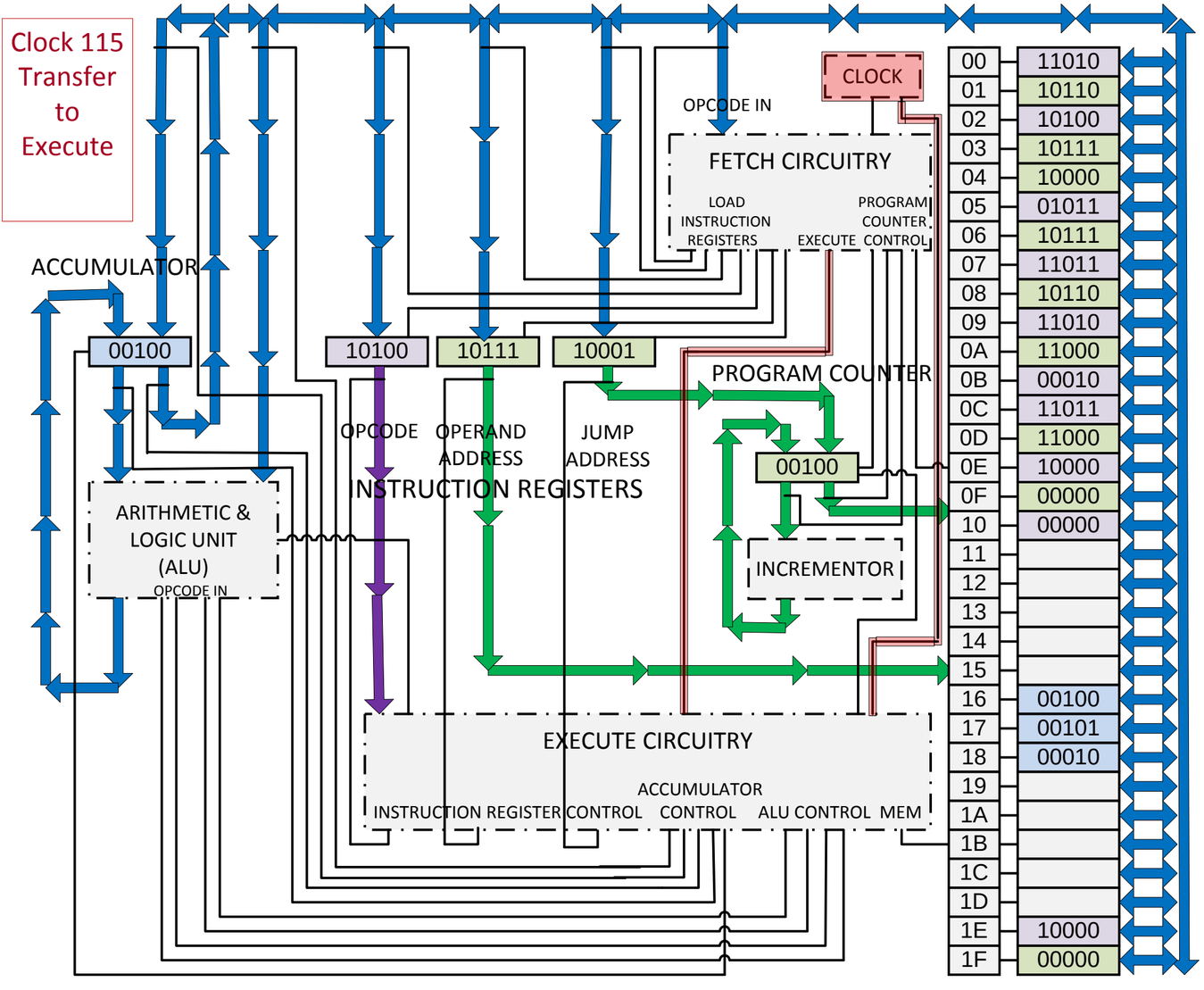
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



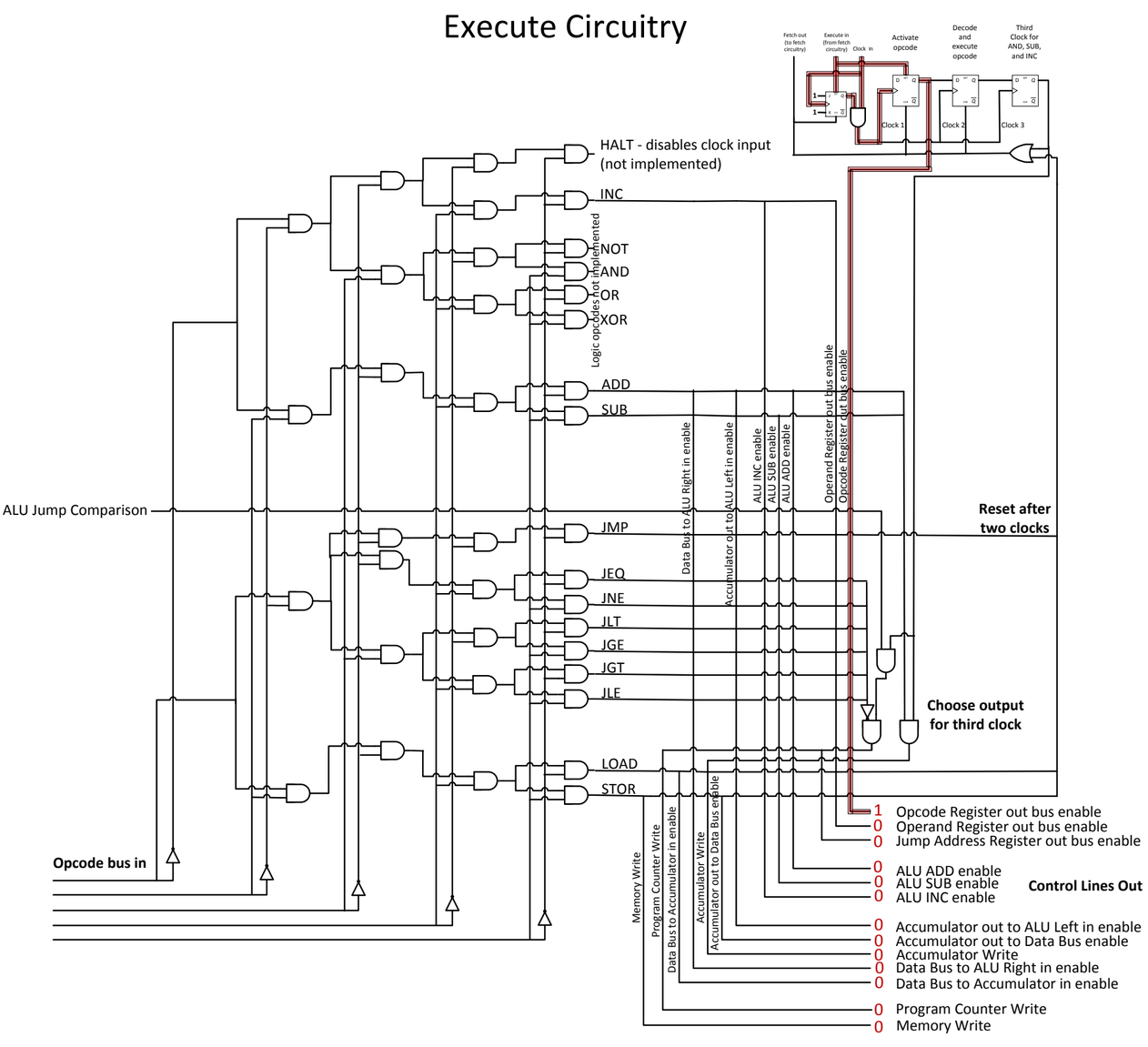


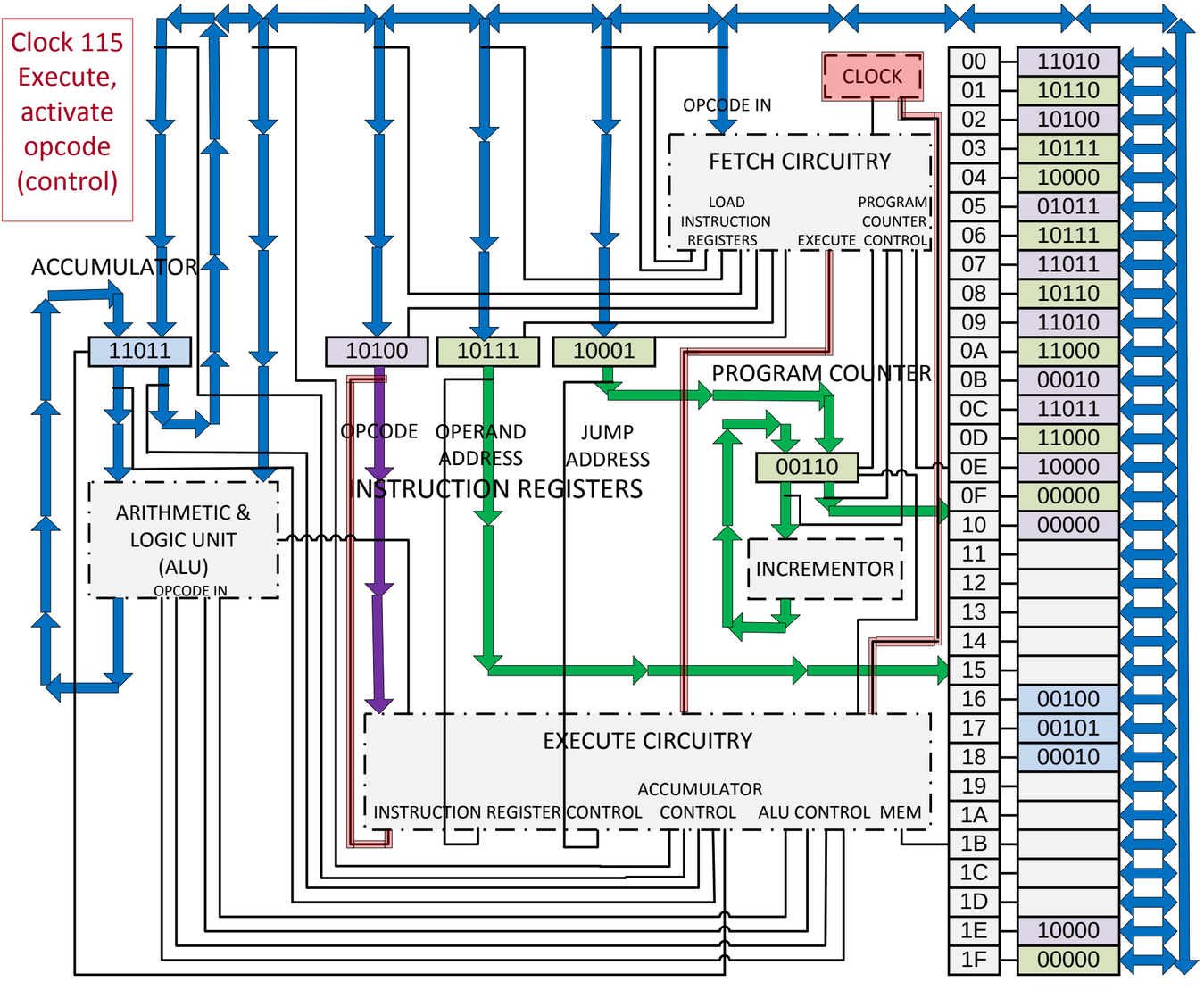
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



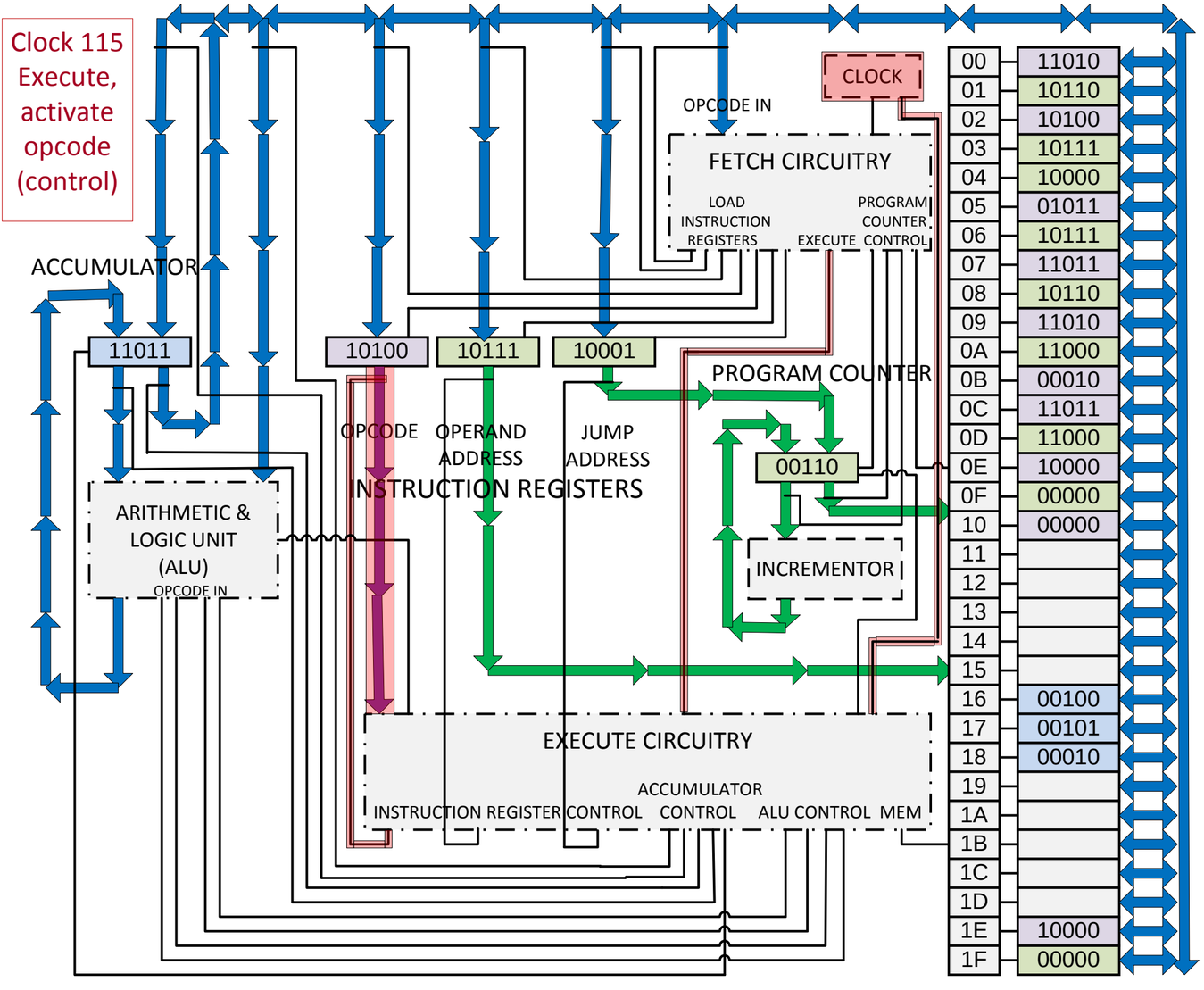
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



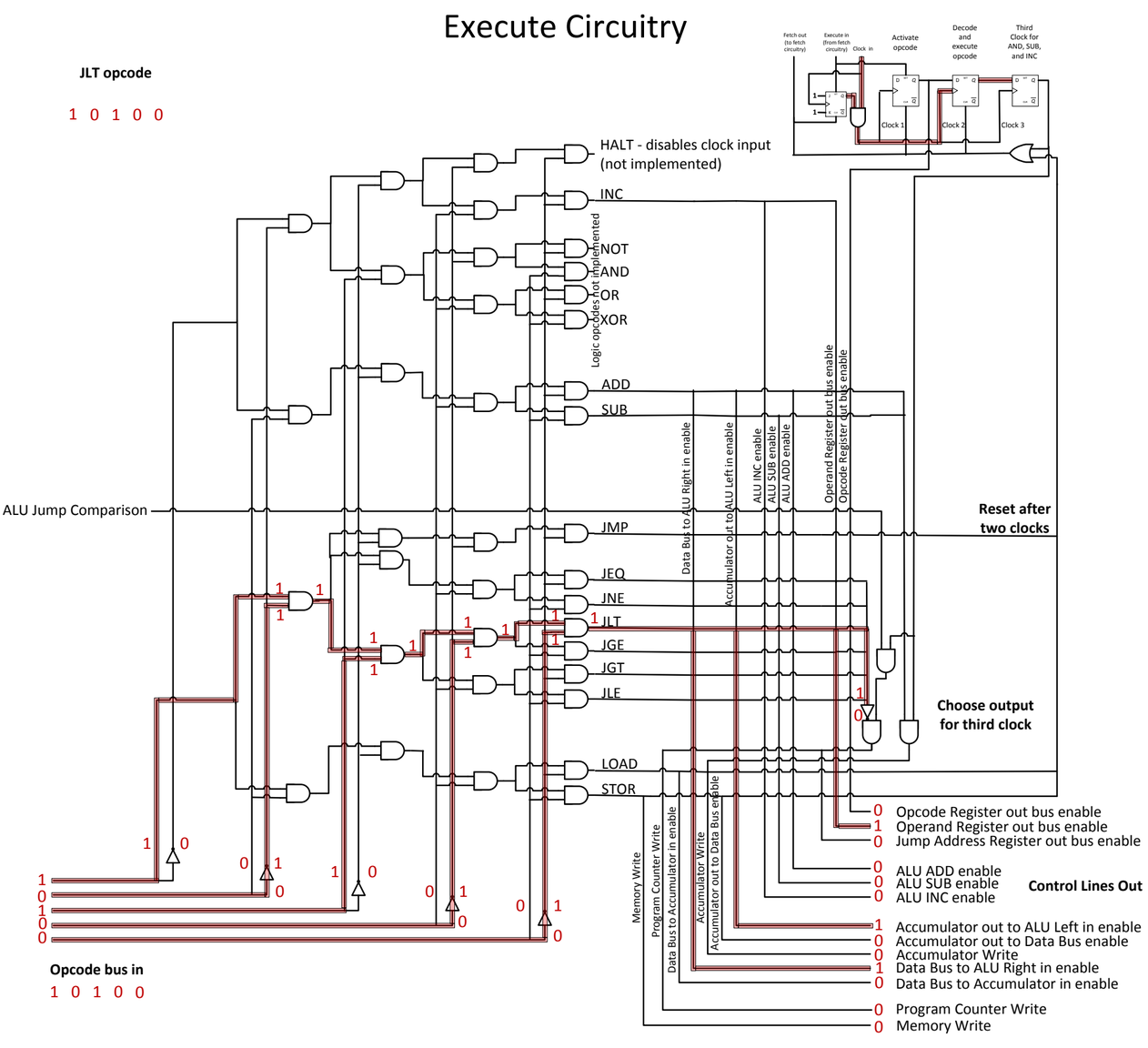


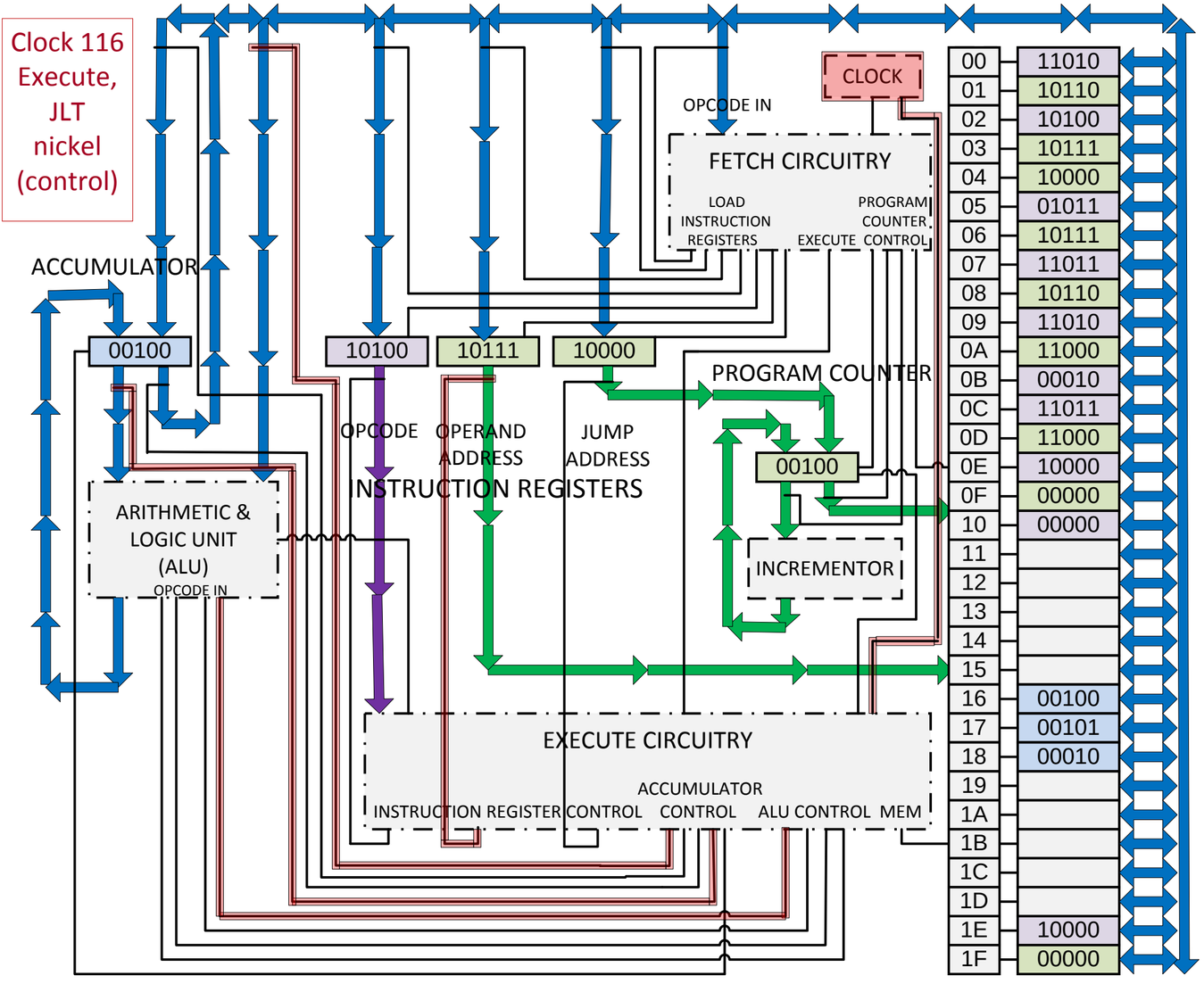
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



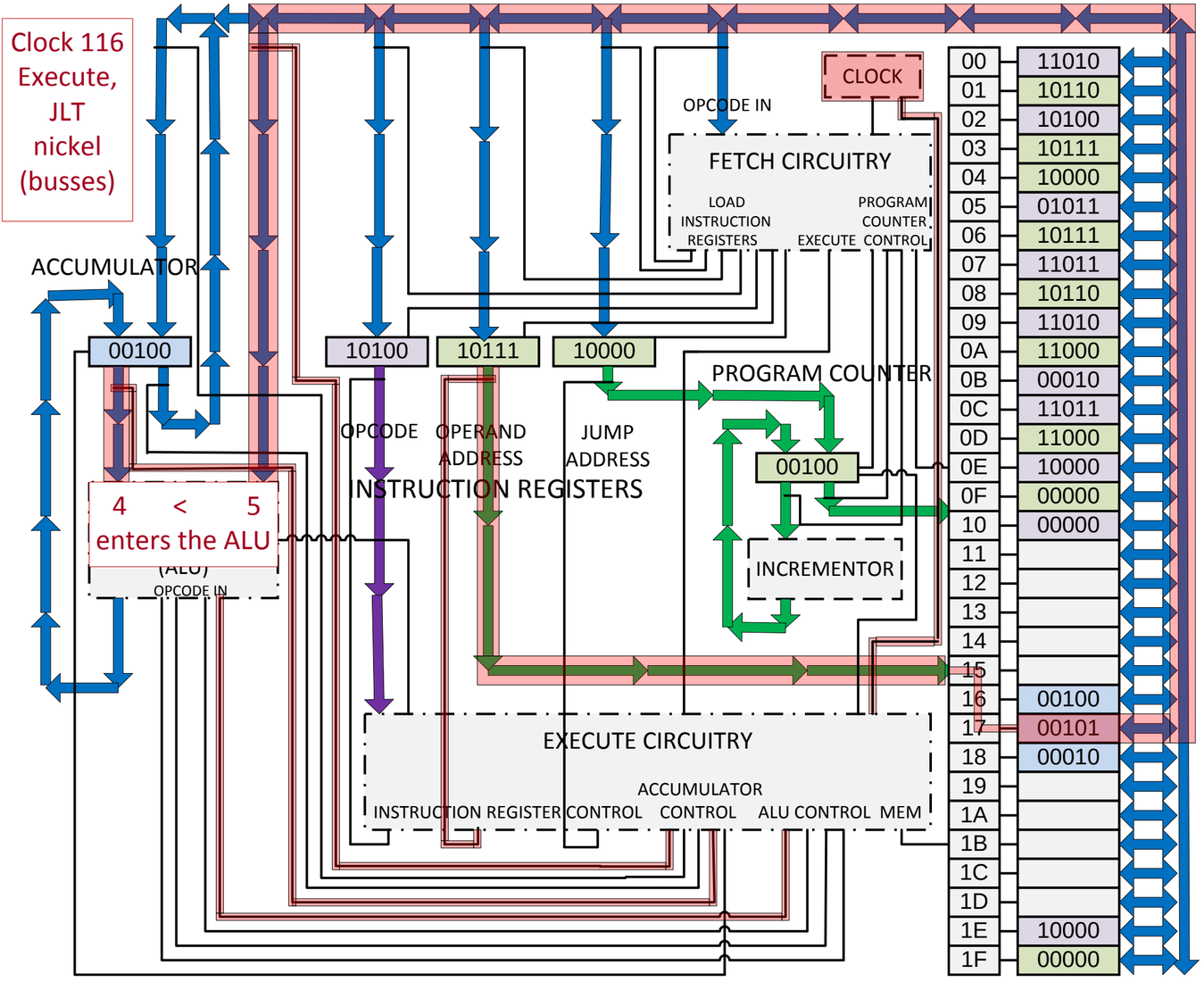
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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



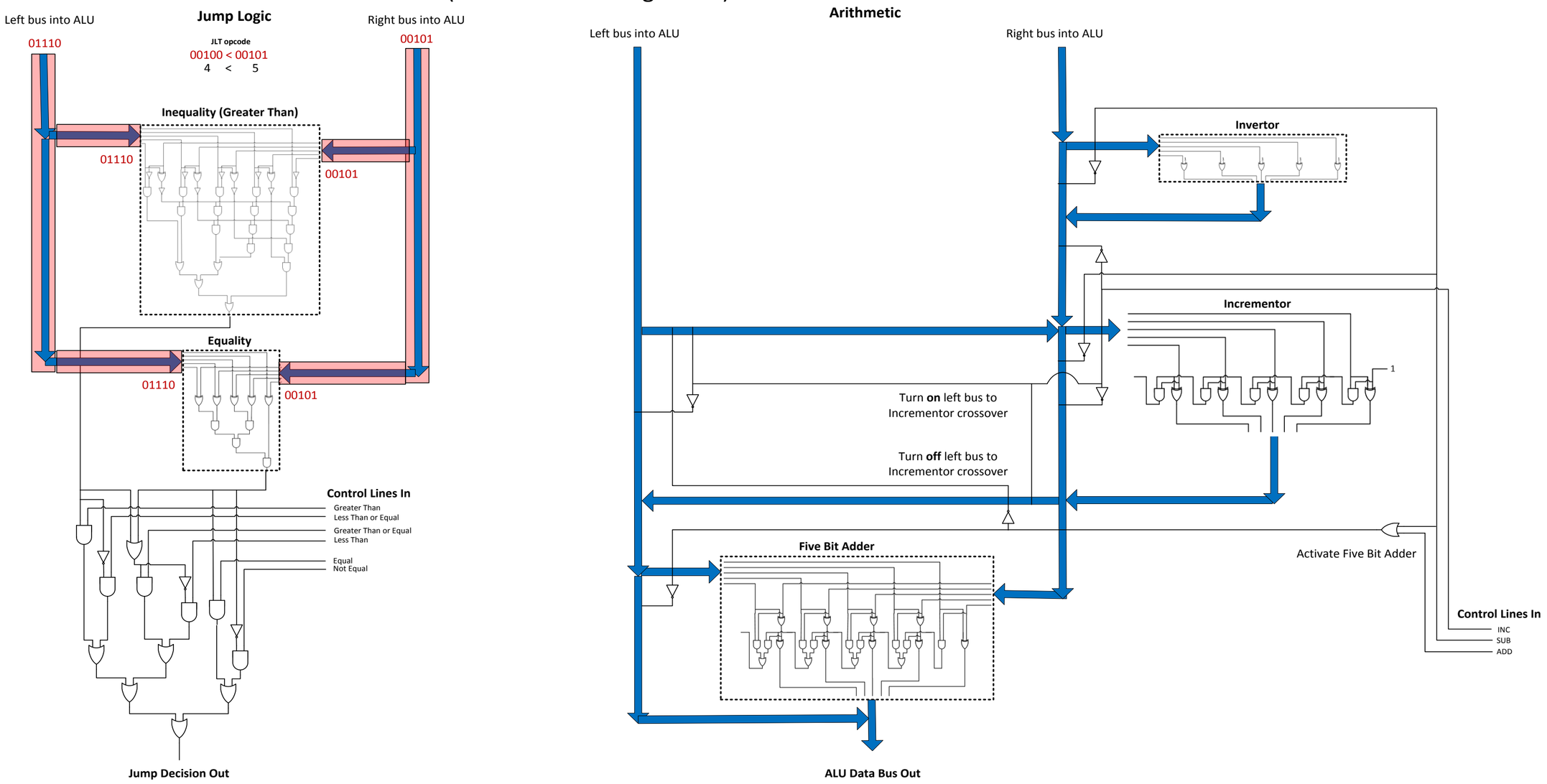


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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	4	# 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	2	# 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)



Inequality – Left > Right

00100 > 00101
4 > 5

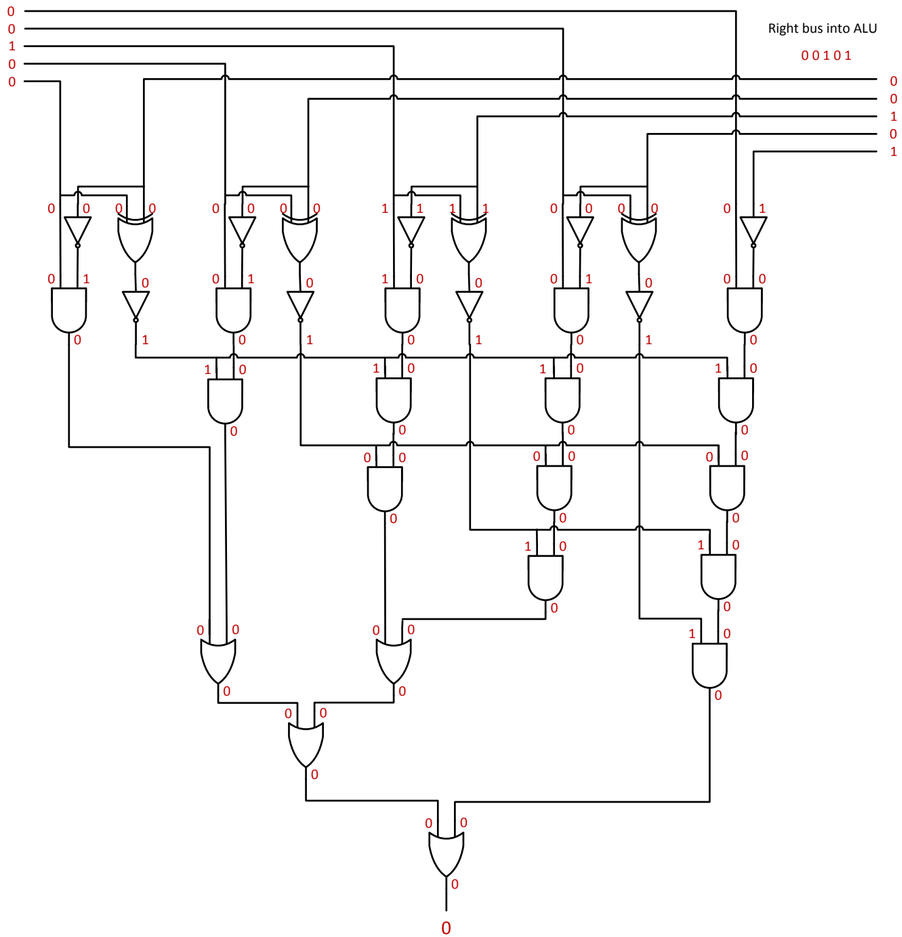
Inputs from Left and Right Busses into the ALU

Left bus into ALU

0 0 1 0 0

Right bus into ALU

0 0 1 0 1



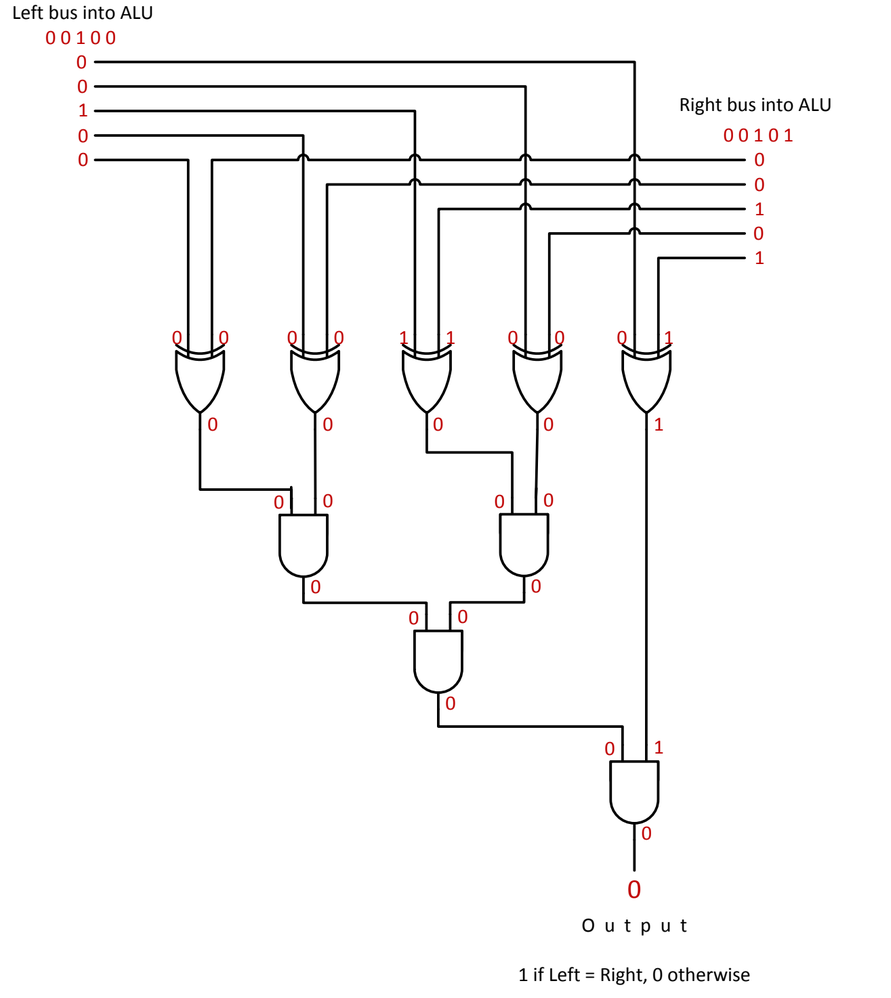
Output

1 if Left > Right, 0 otherwise

Equality

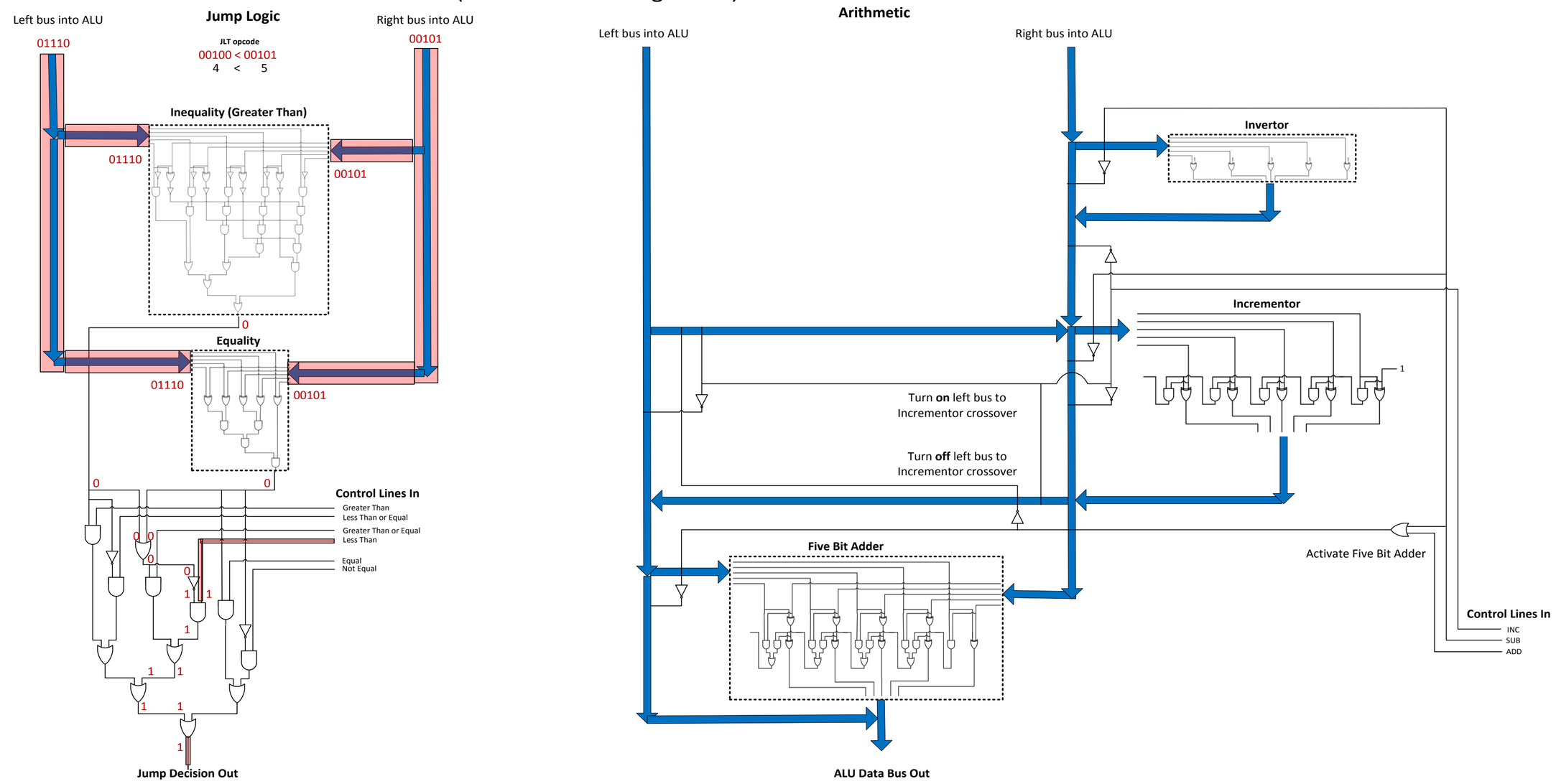
00100 = 00101
4 = 5

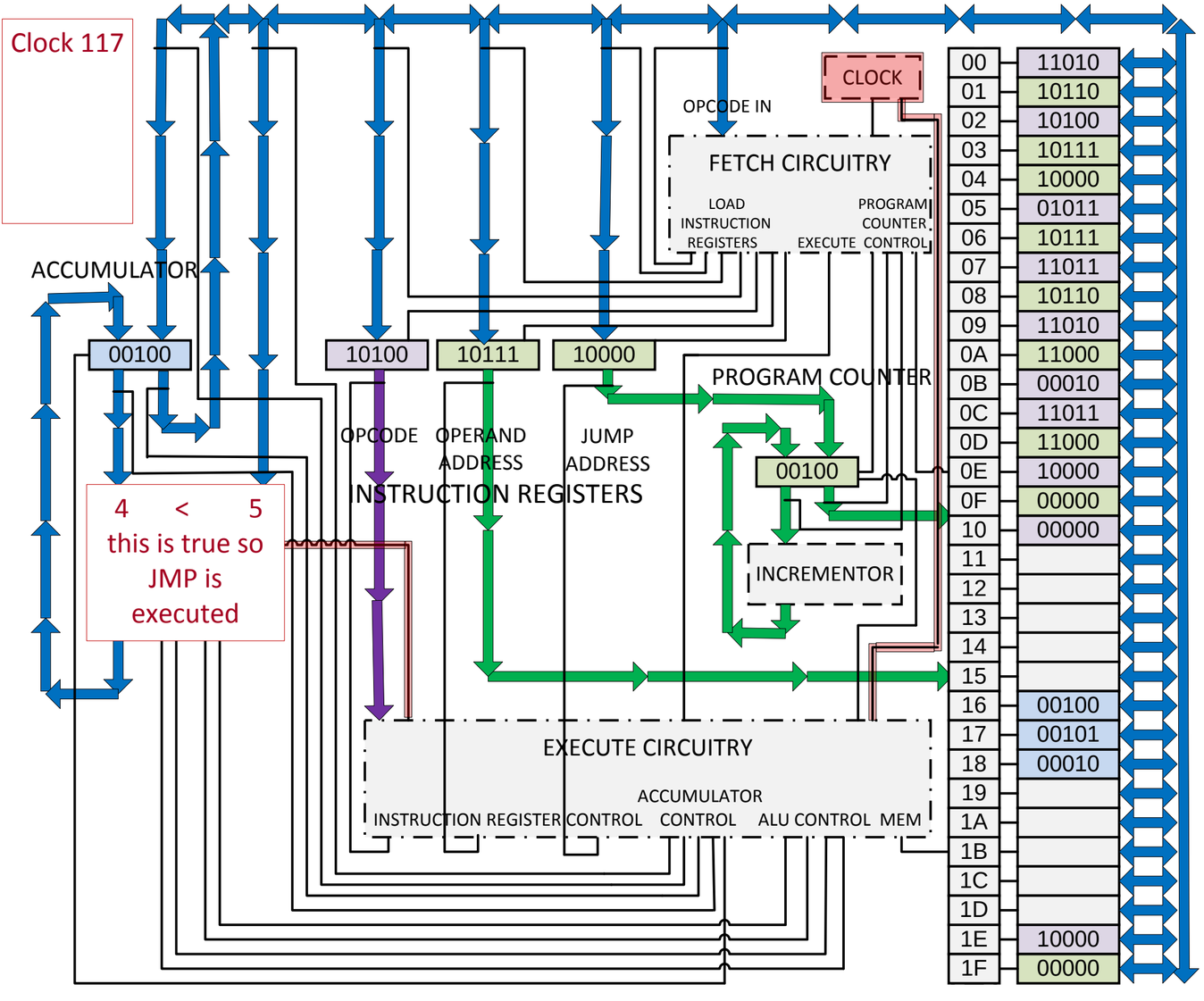
Inputs from Left and Right Busses into the ALU



1 if Left = Right, 0 otherwise

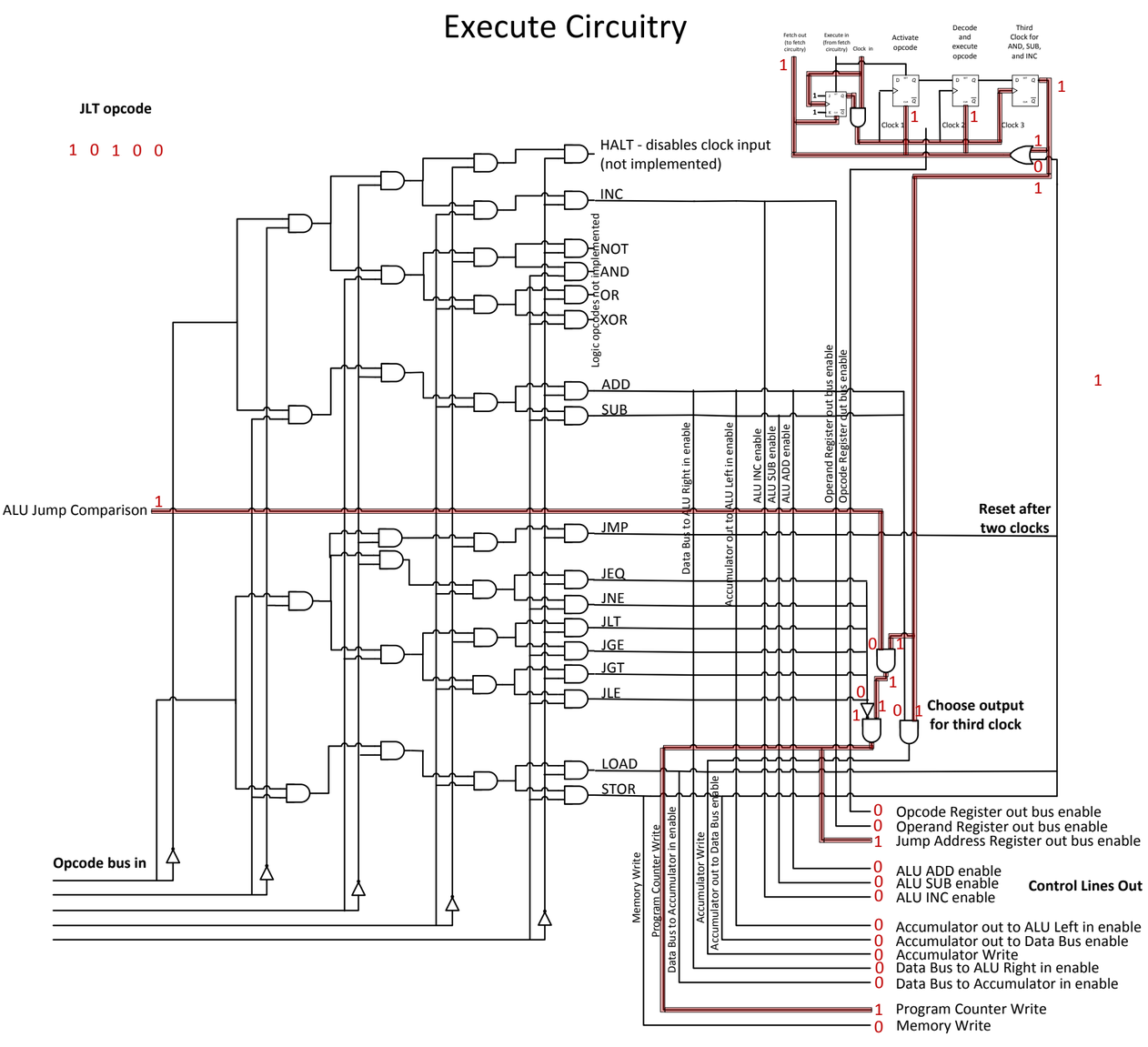
A L U
(Arithmetic and Logic Unit)

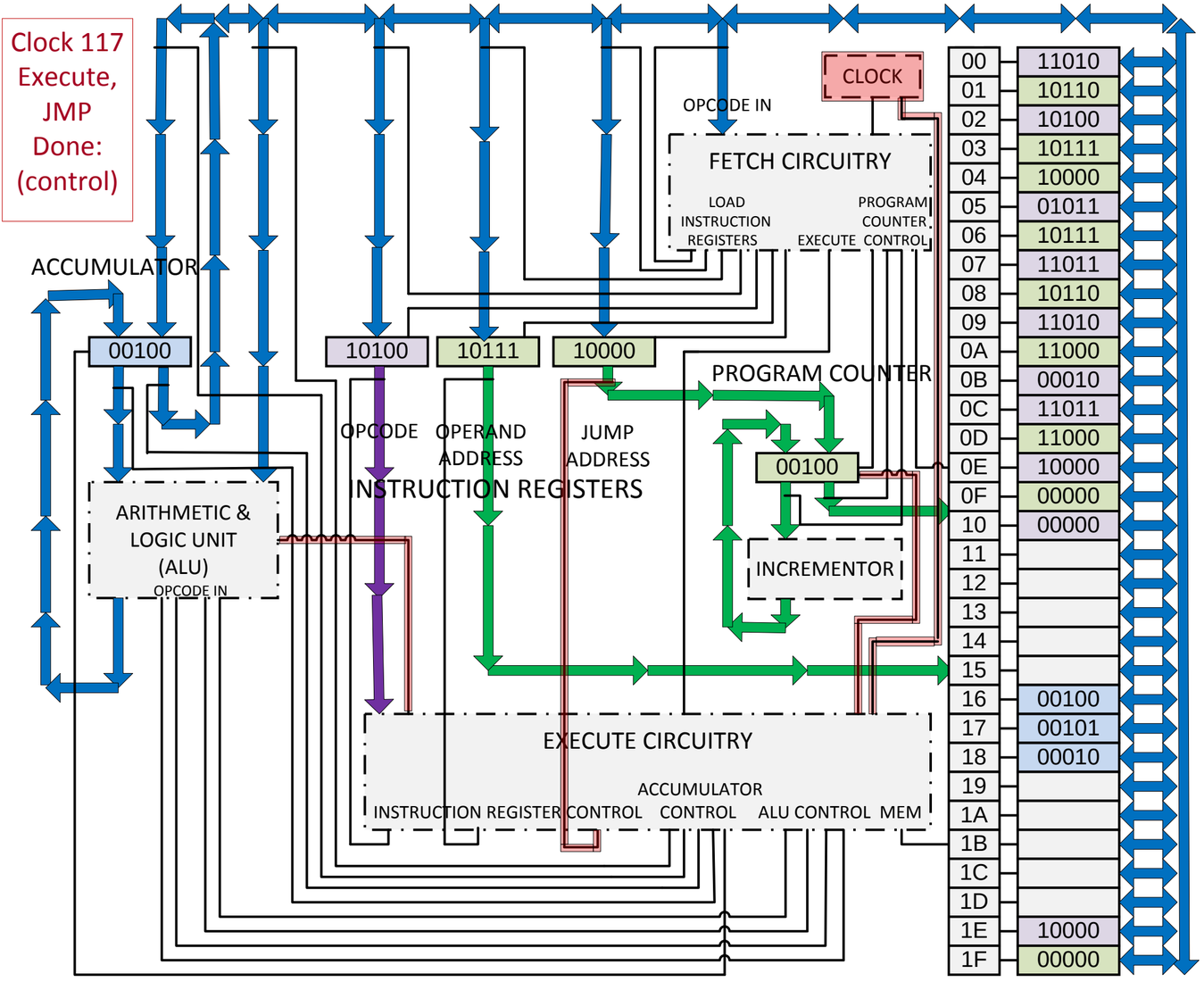




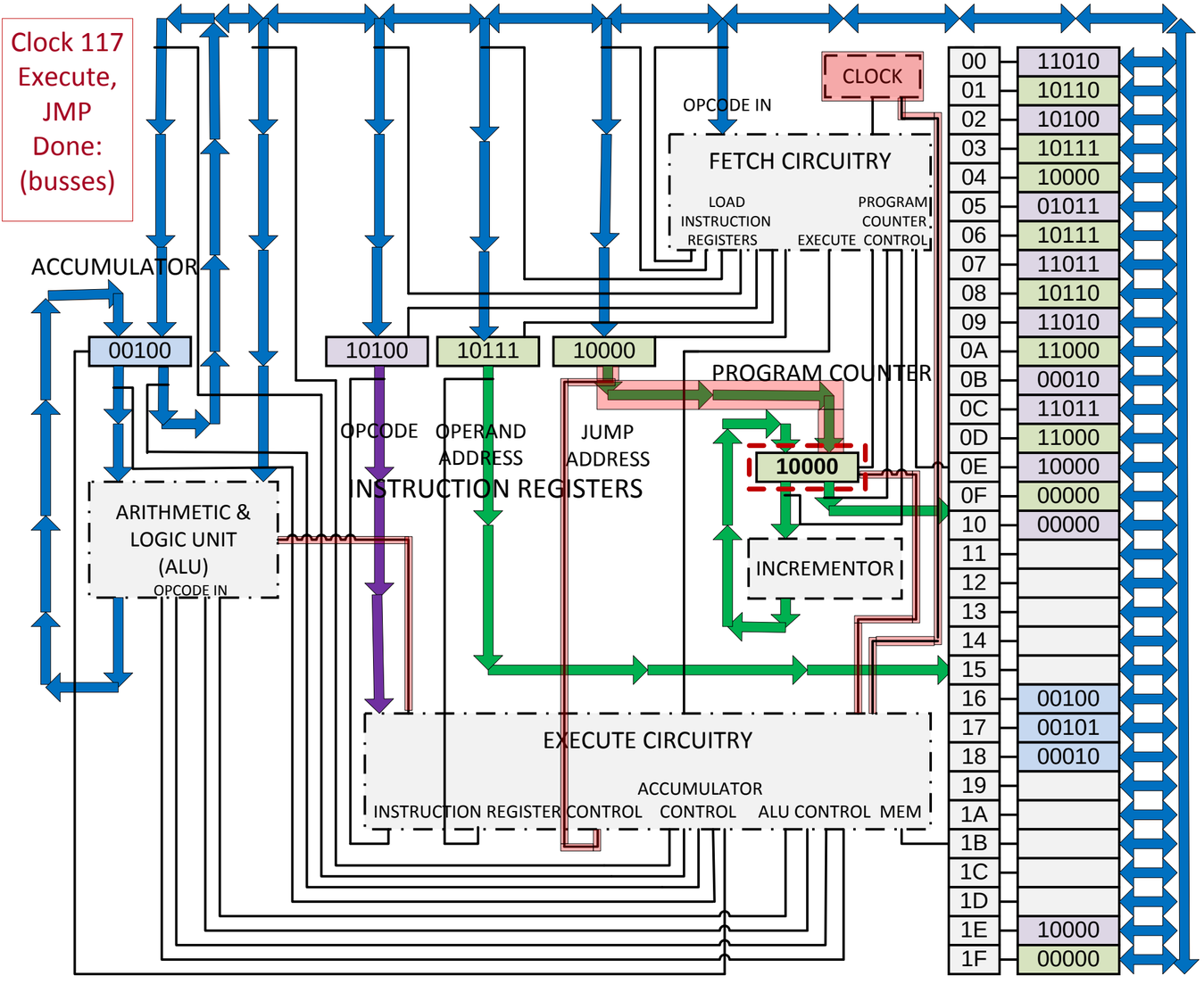
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	4	# 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	2	# 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	4	# 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	2	# 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	4	# 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	2	# 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

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

End of Presentation