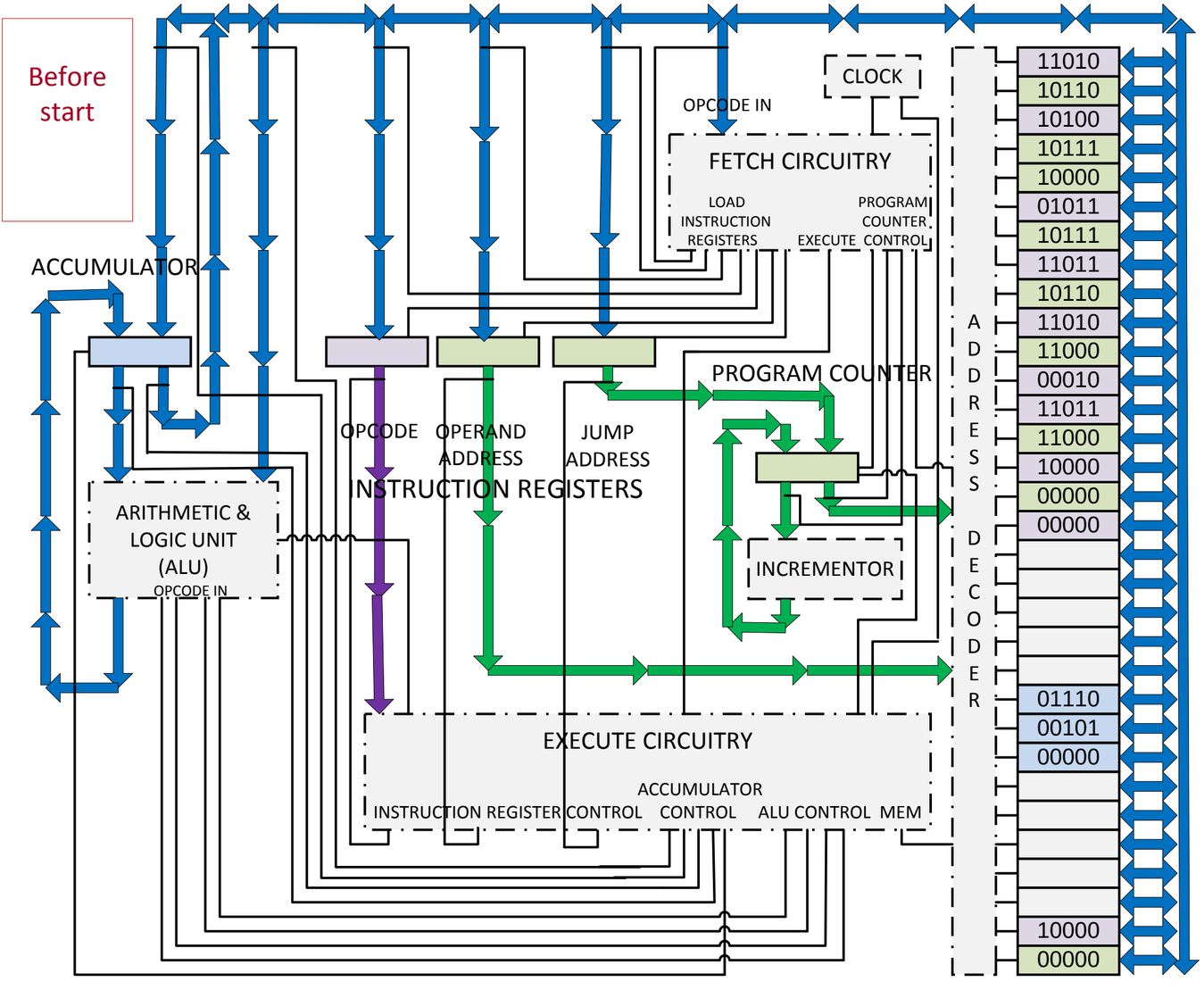
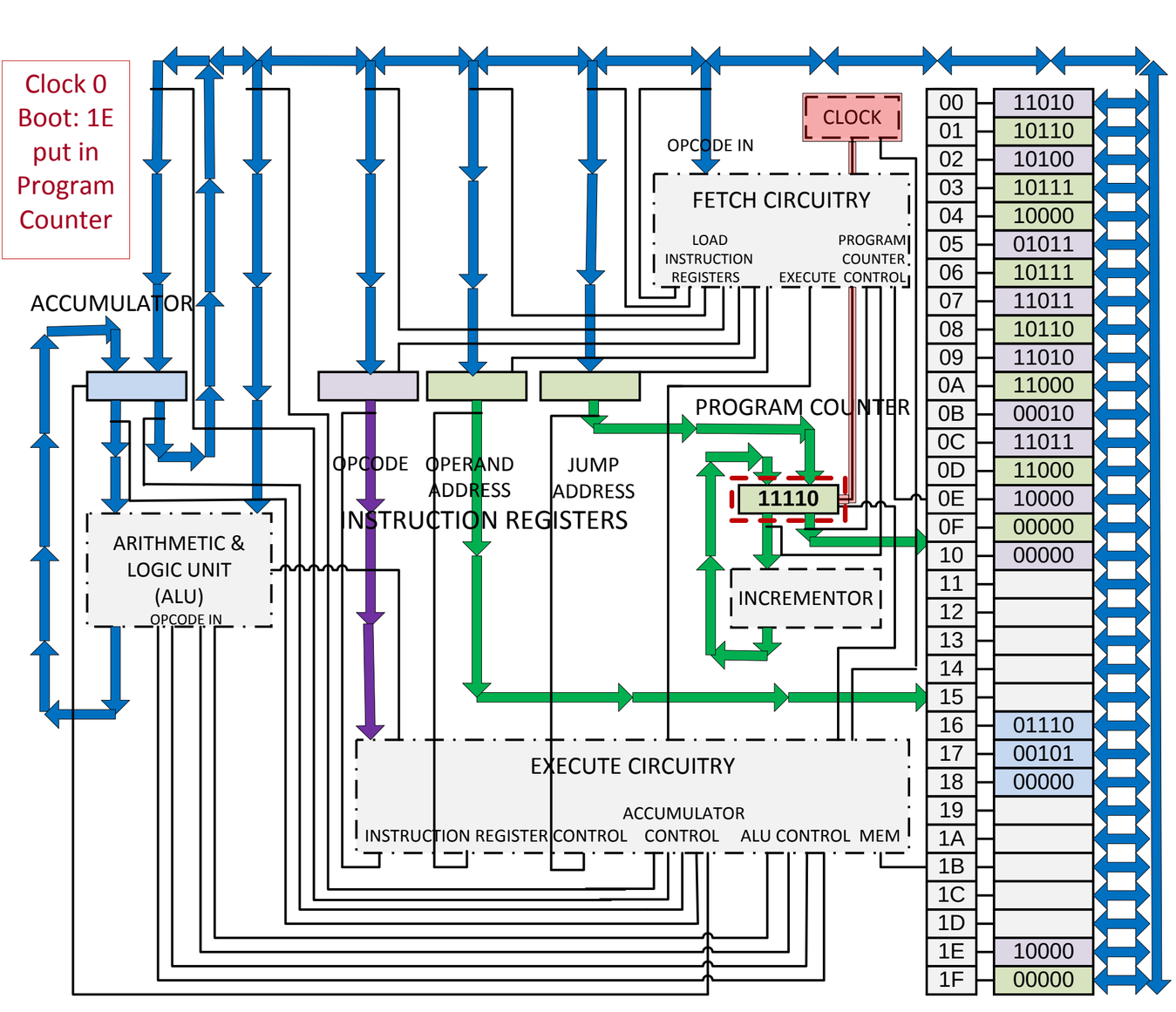
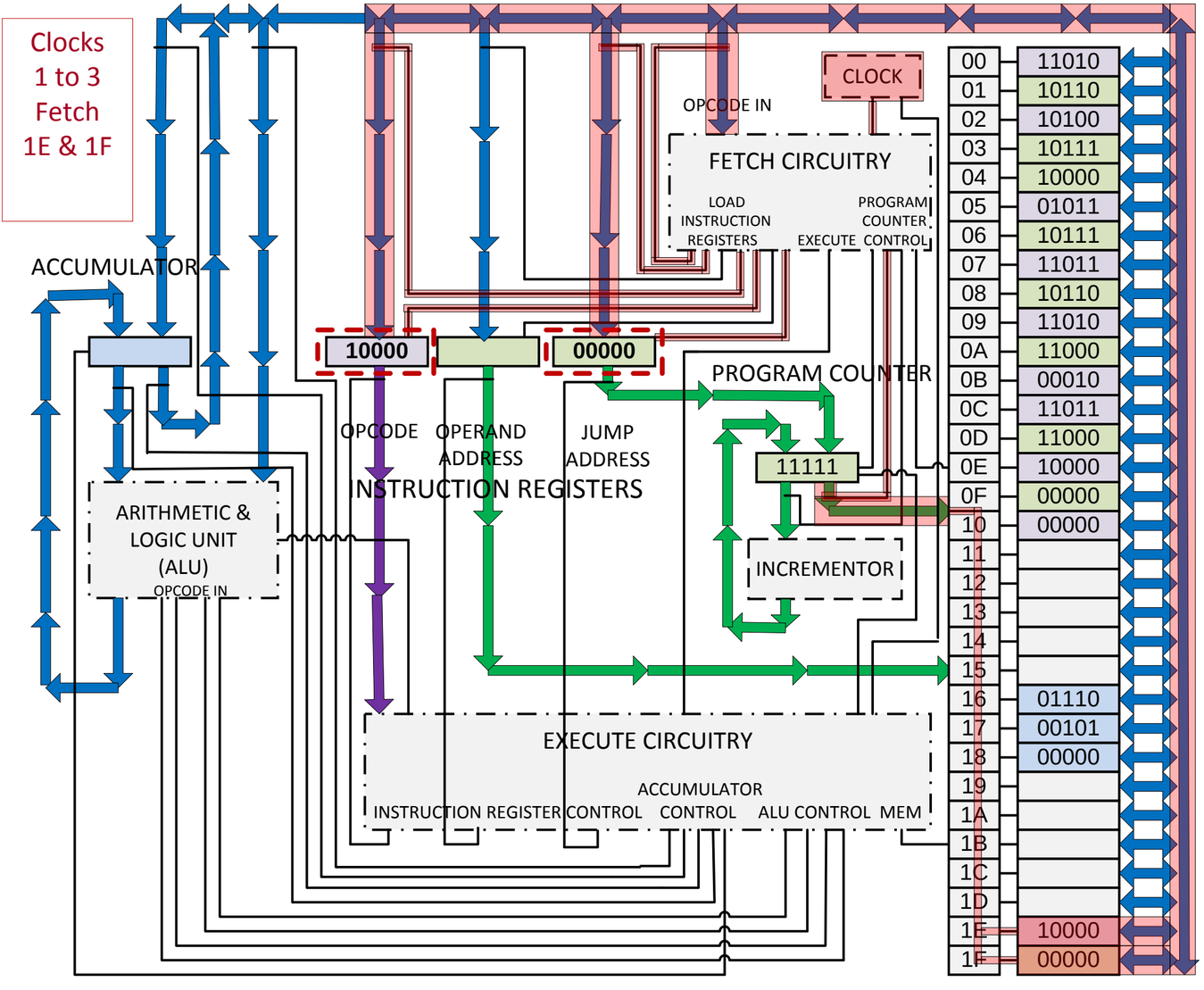


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

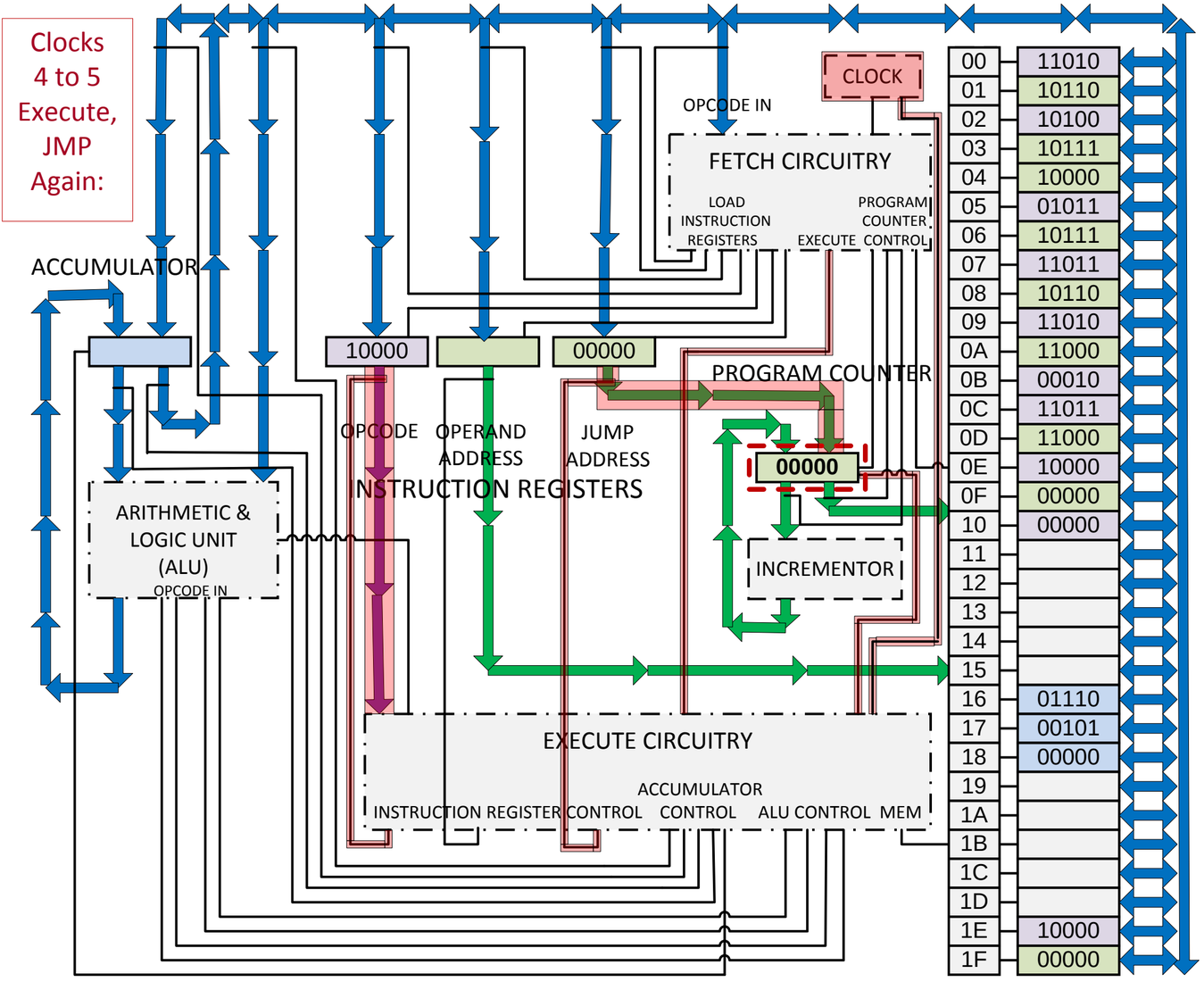


00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	01110
17	00101
18	00000
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000

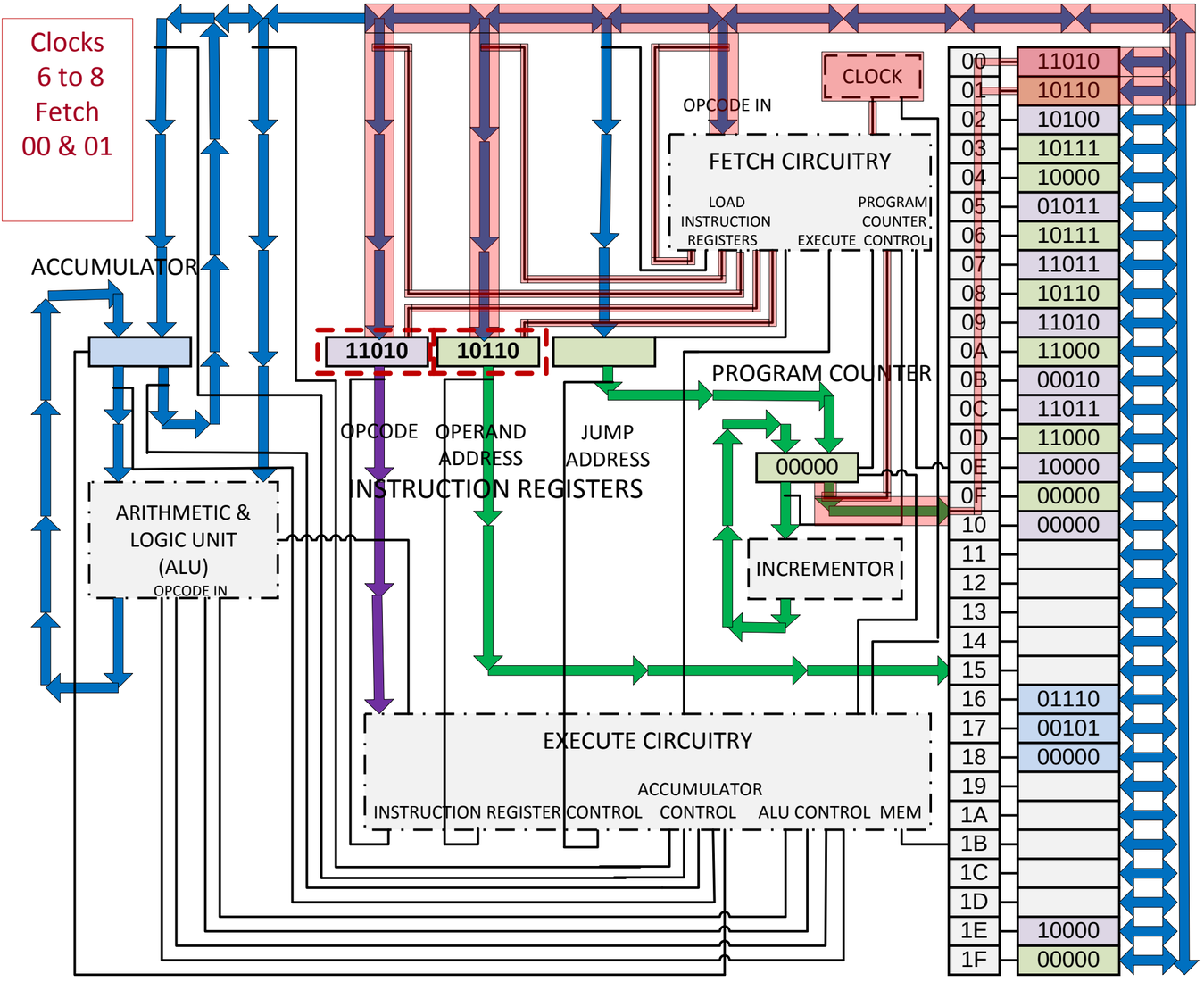
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



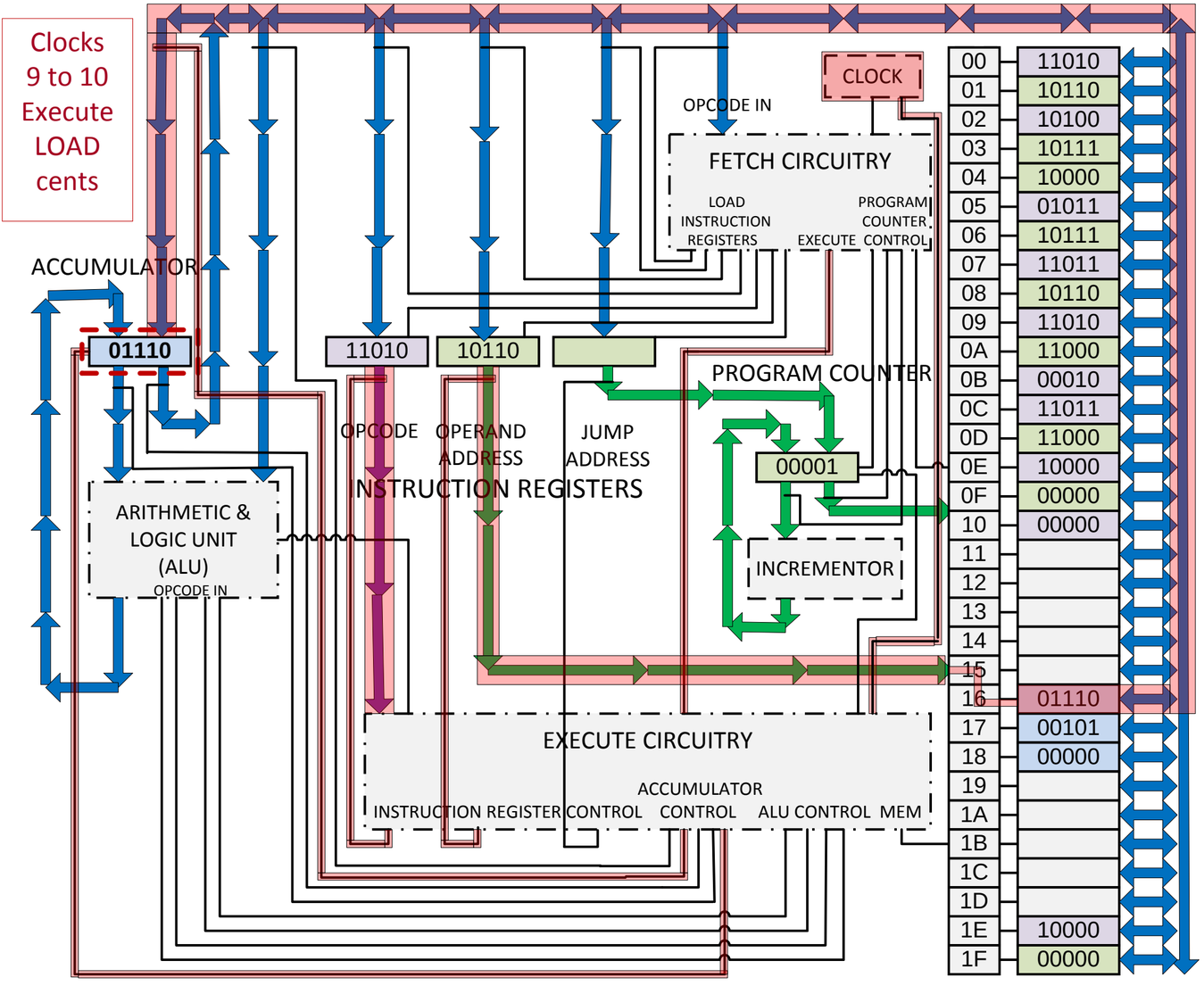
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



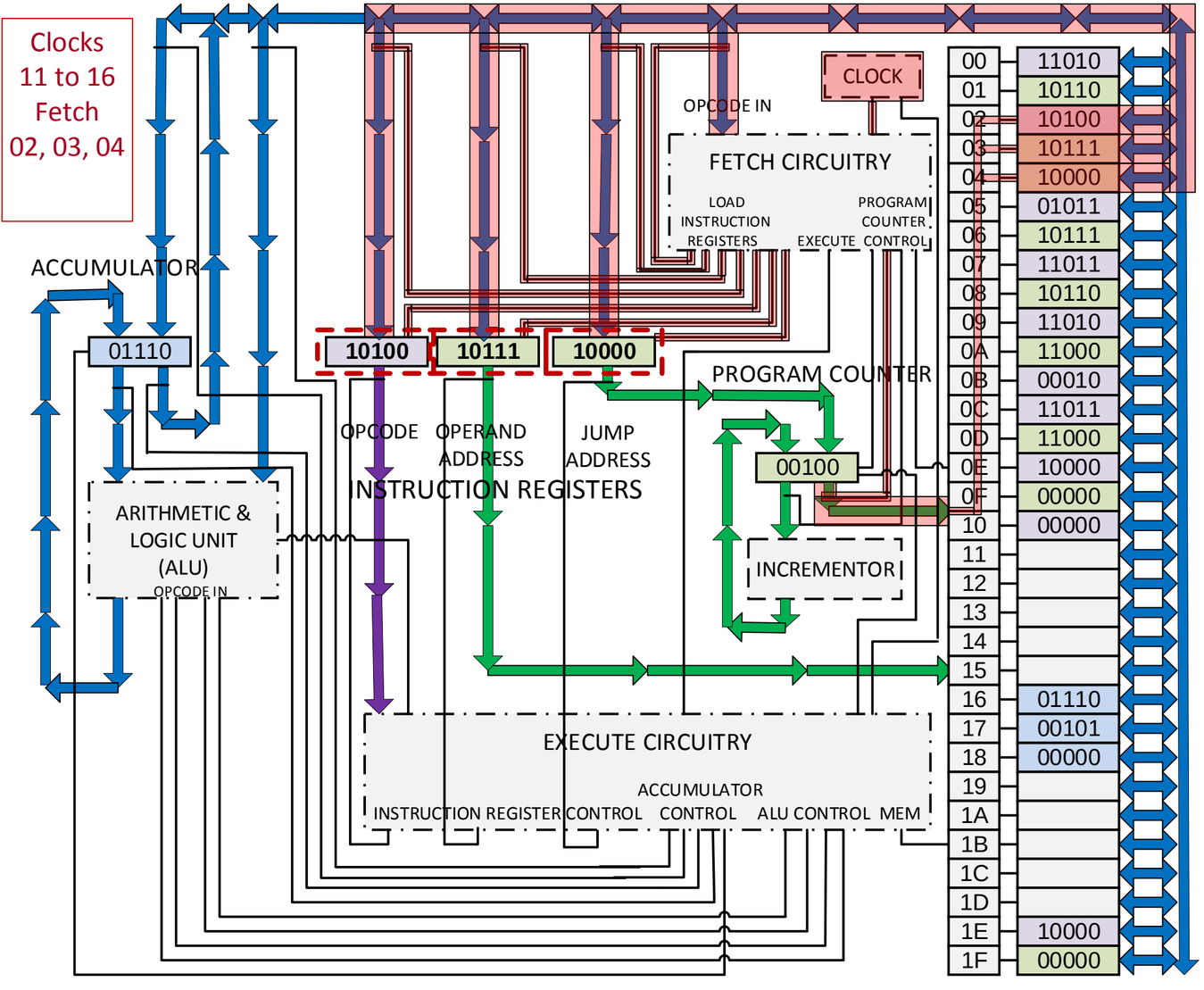
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



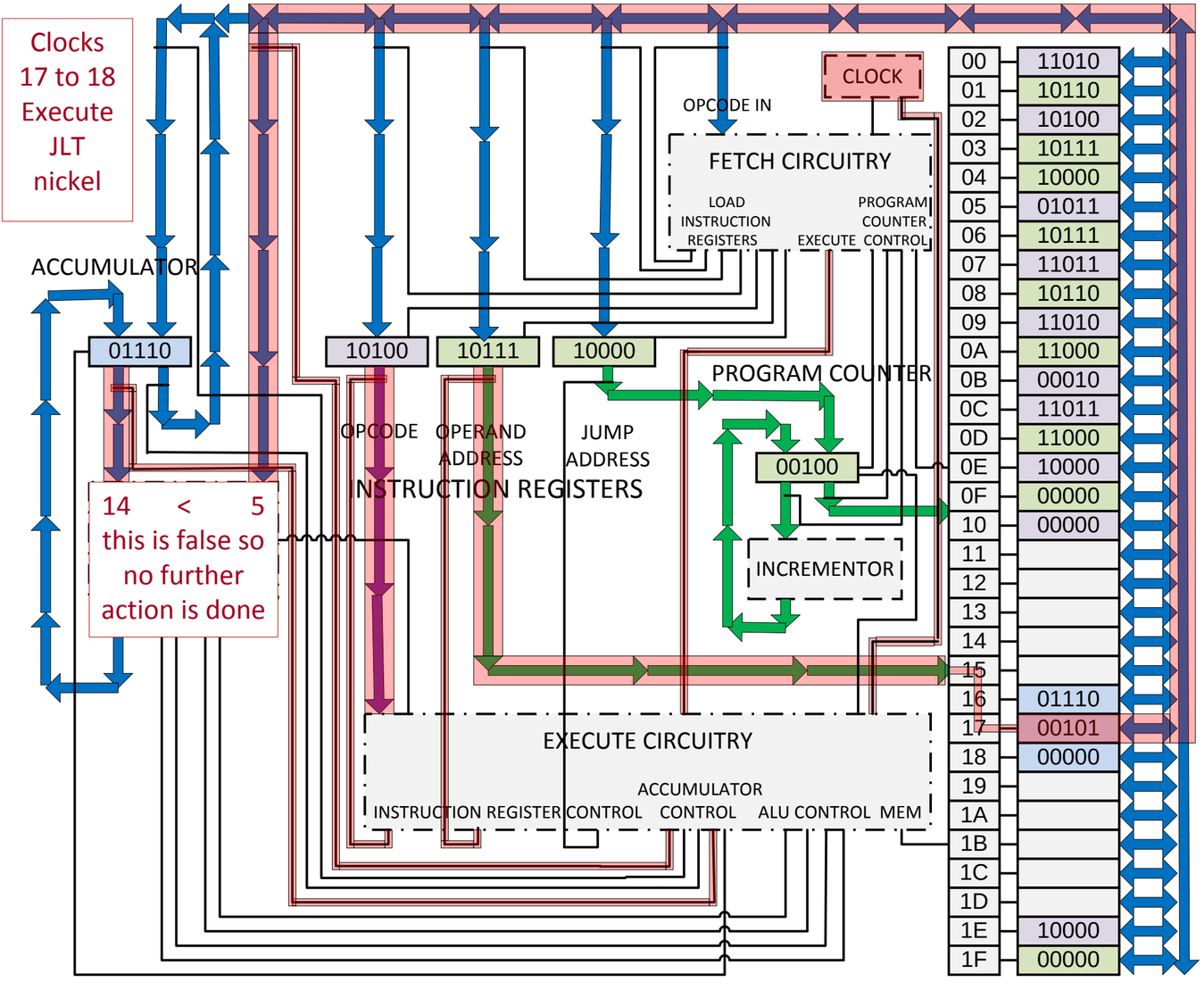
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



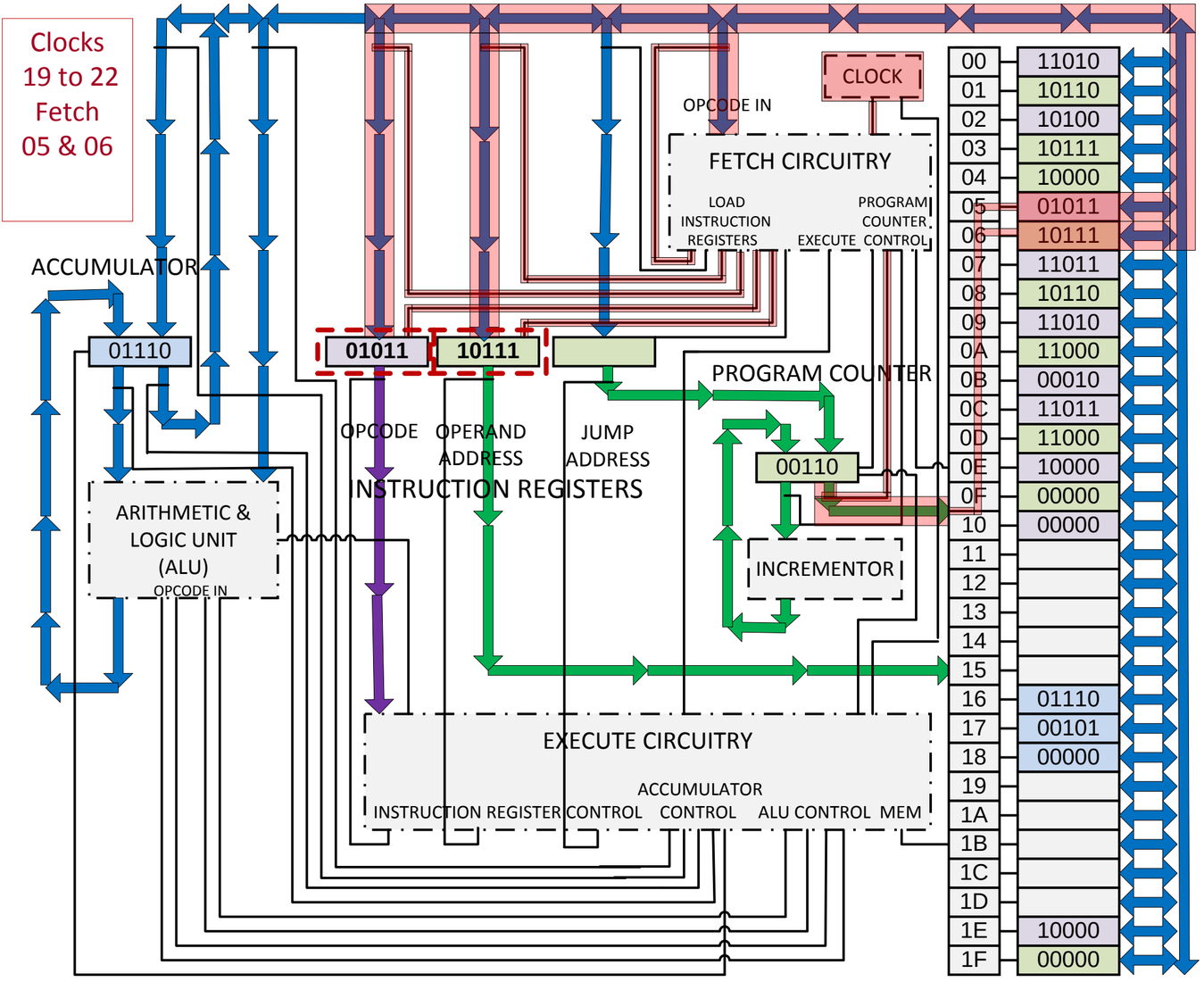
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



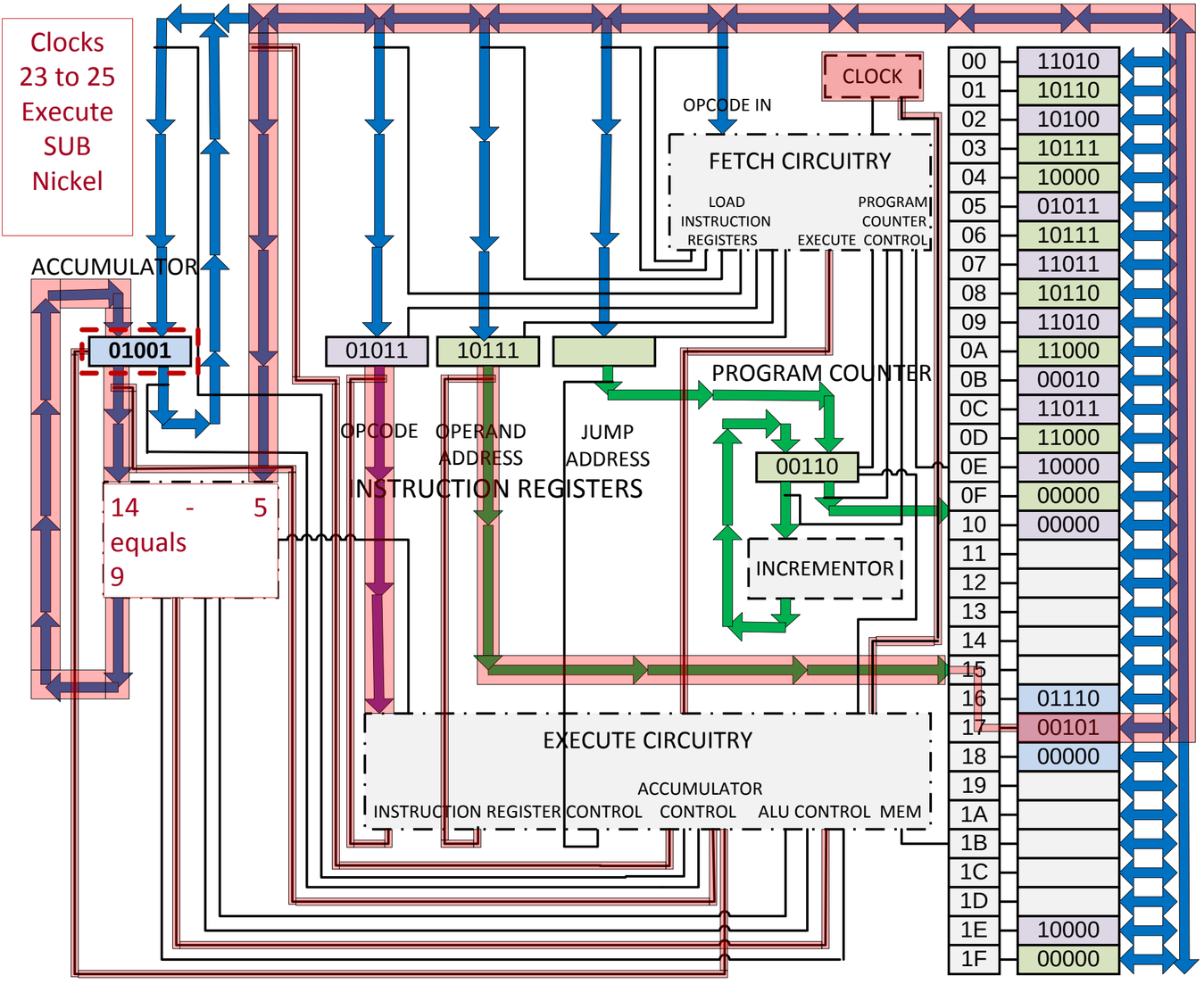
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



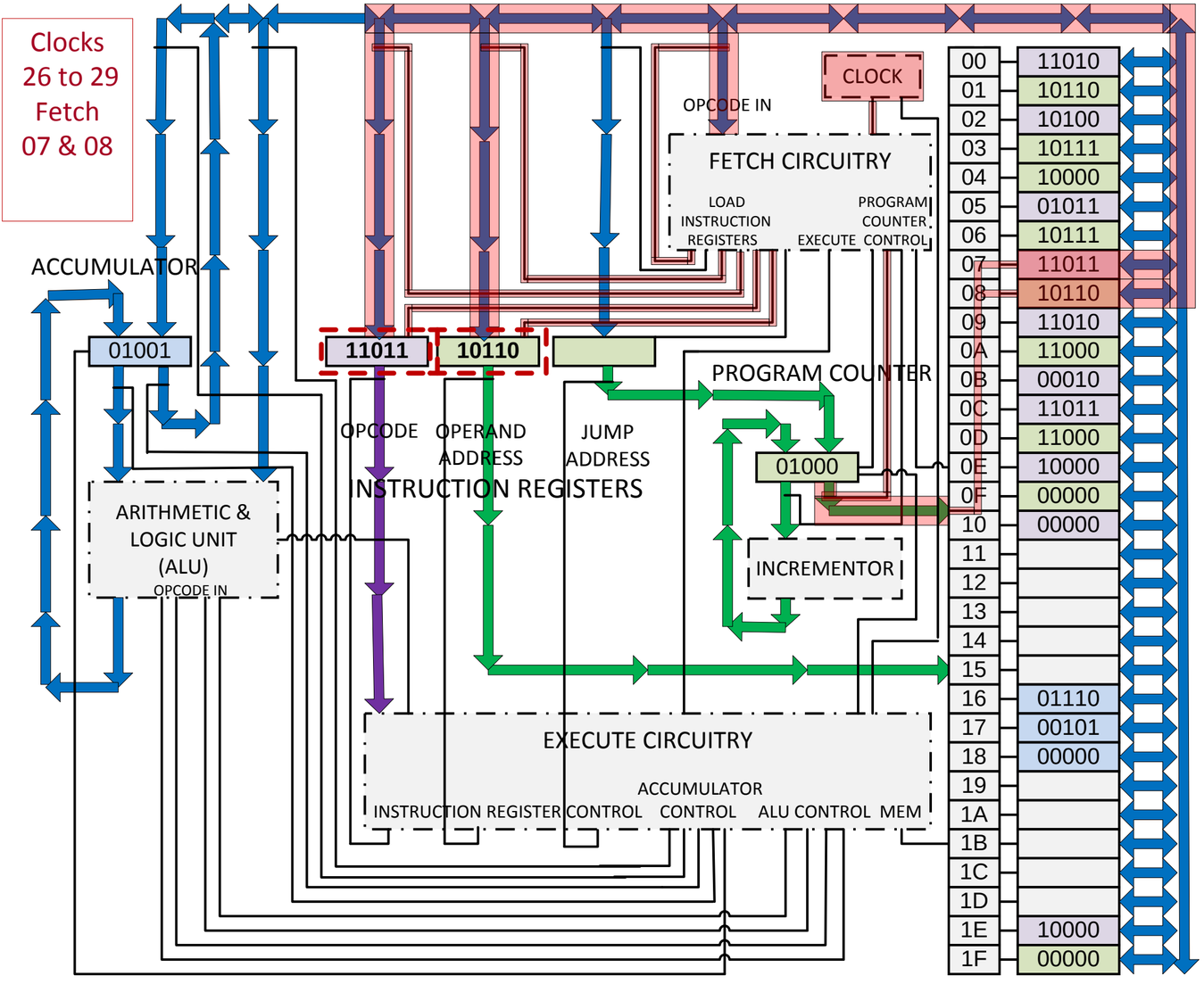
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



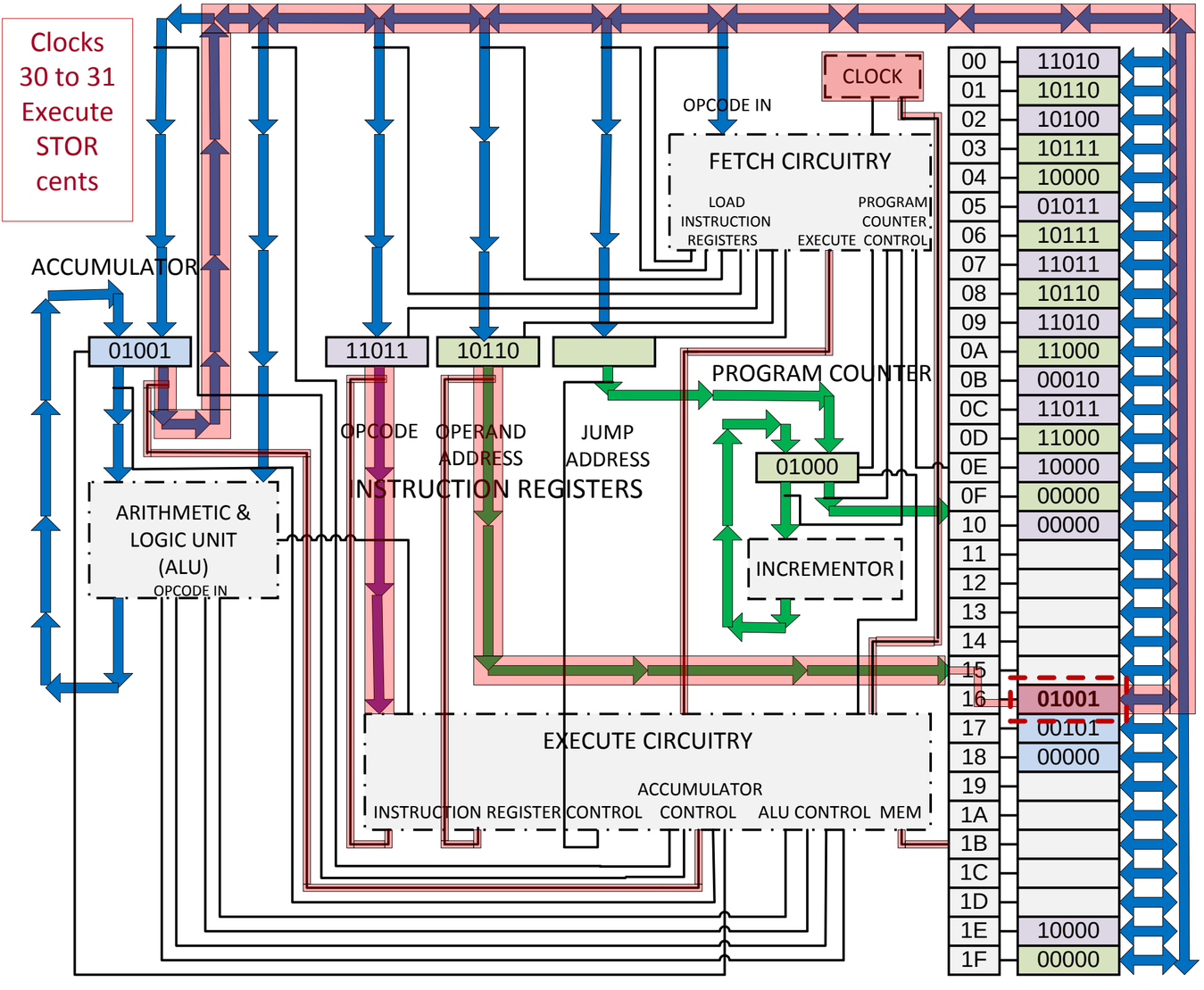
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



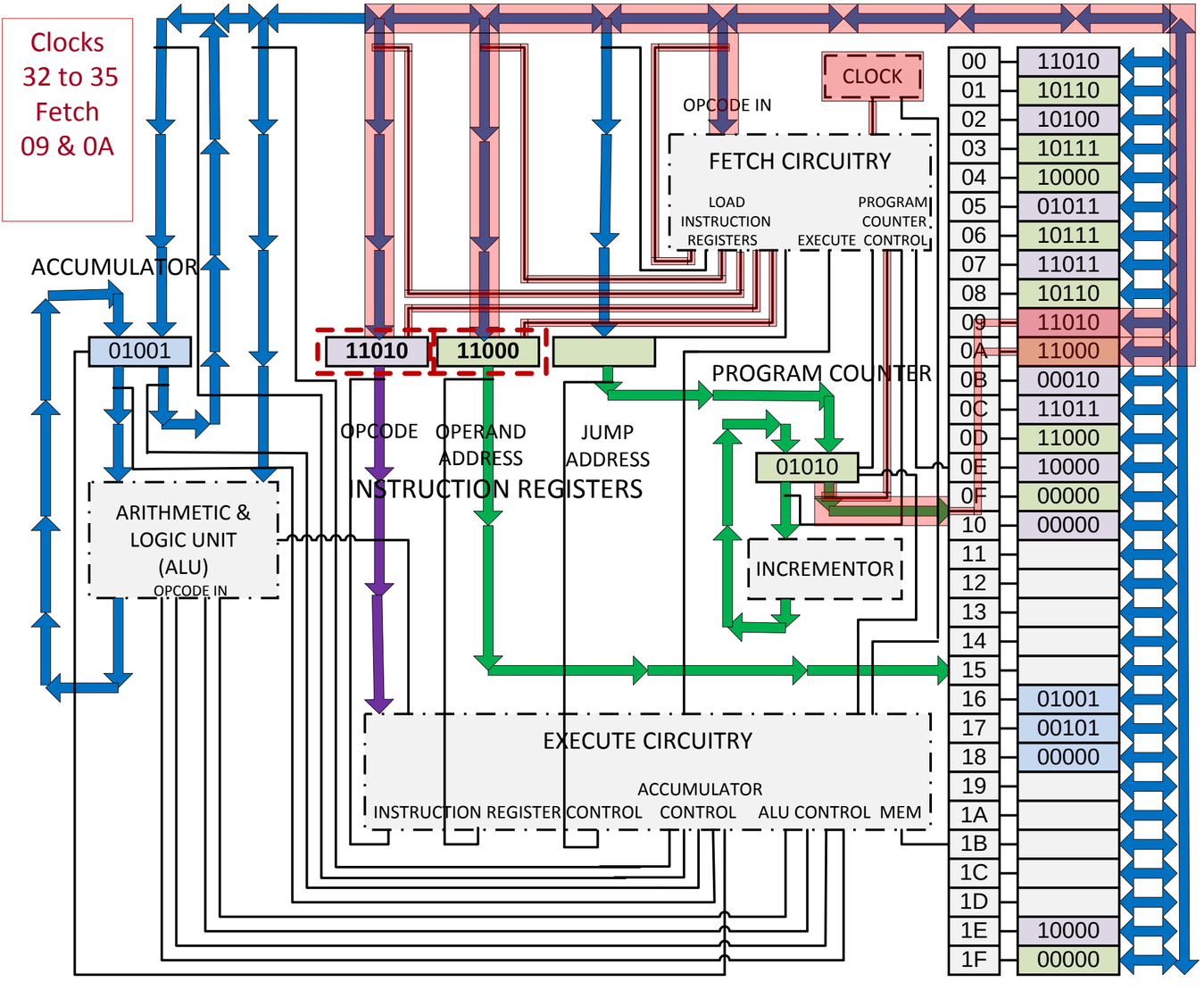
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



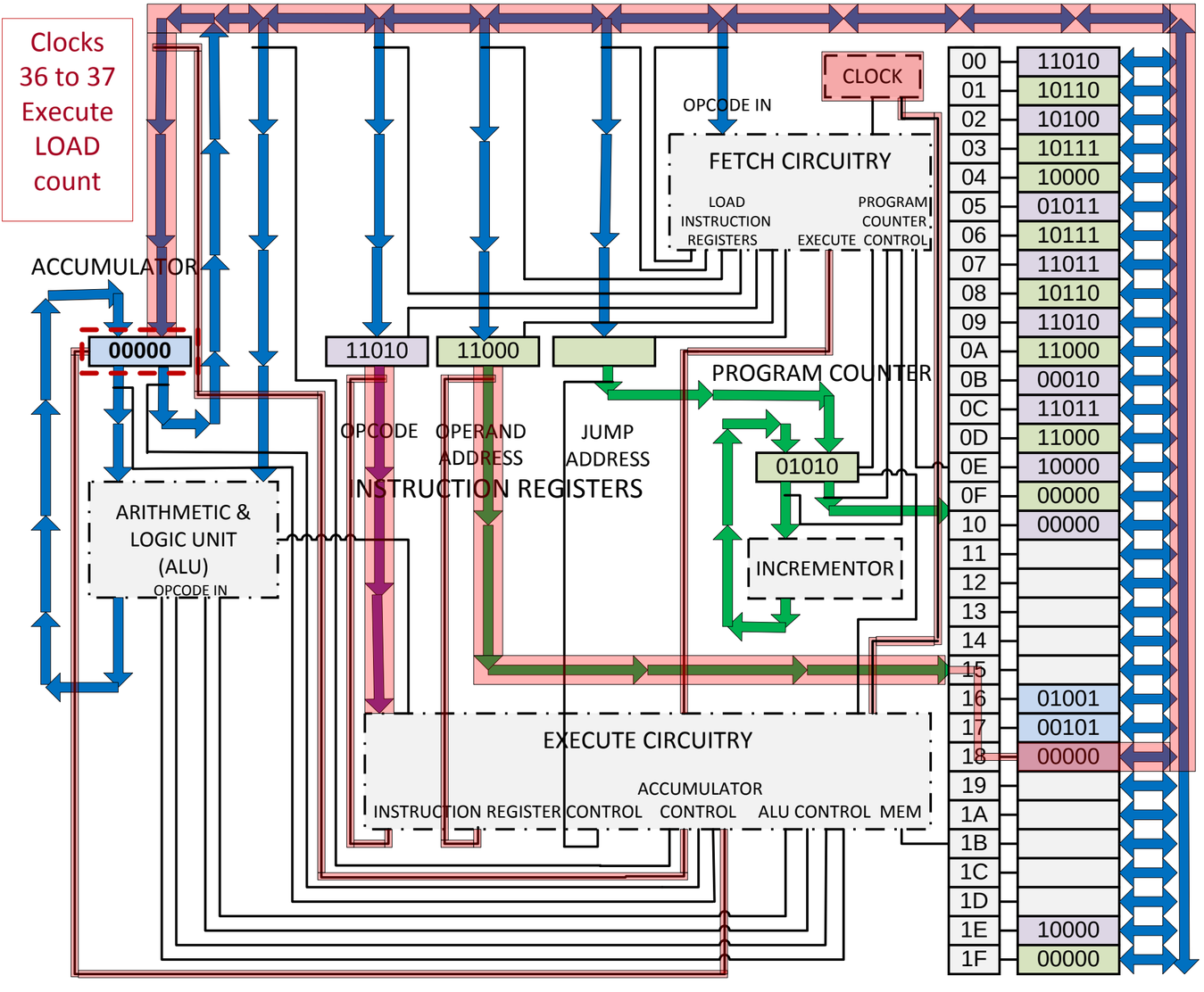
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



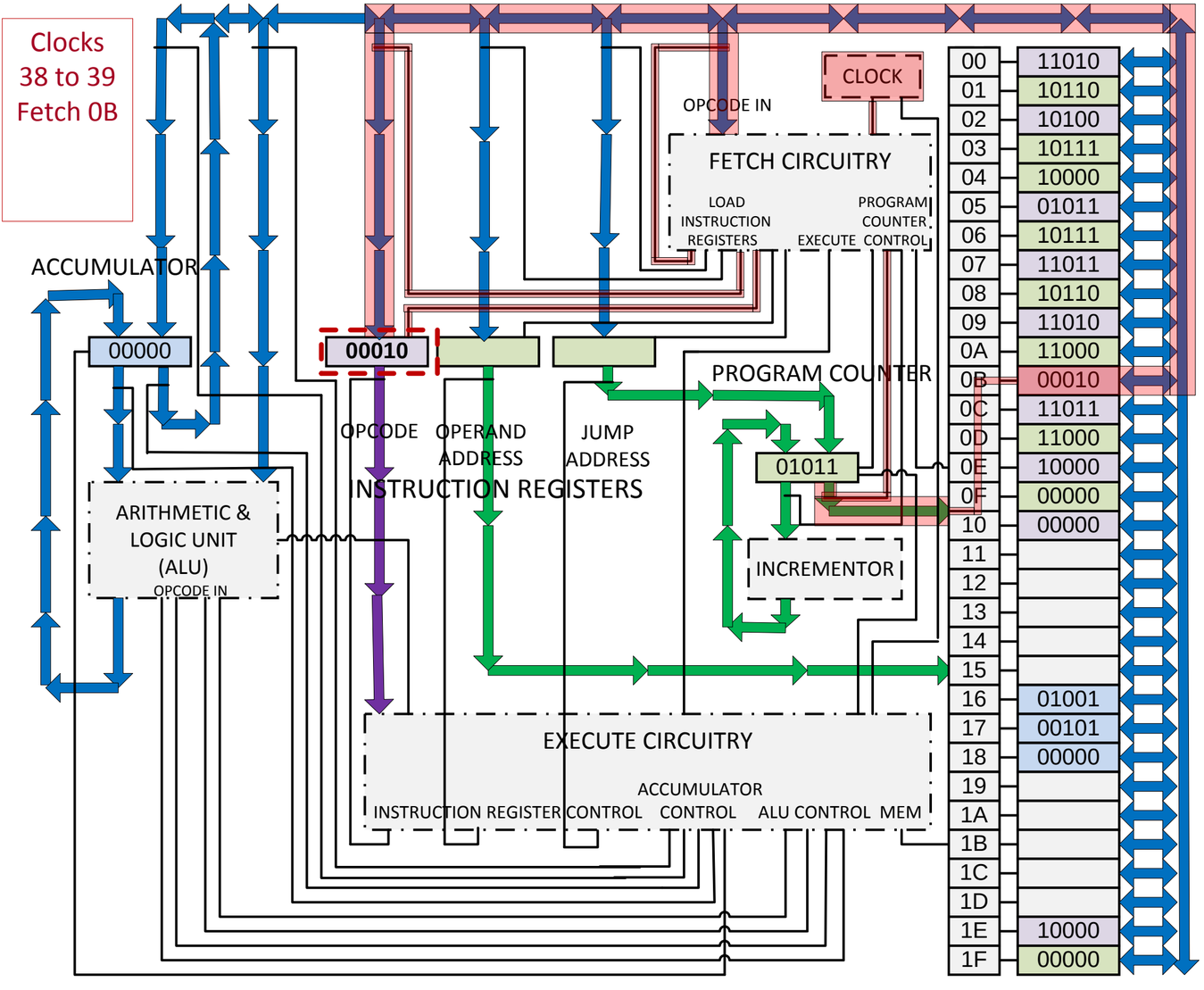
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



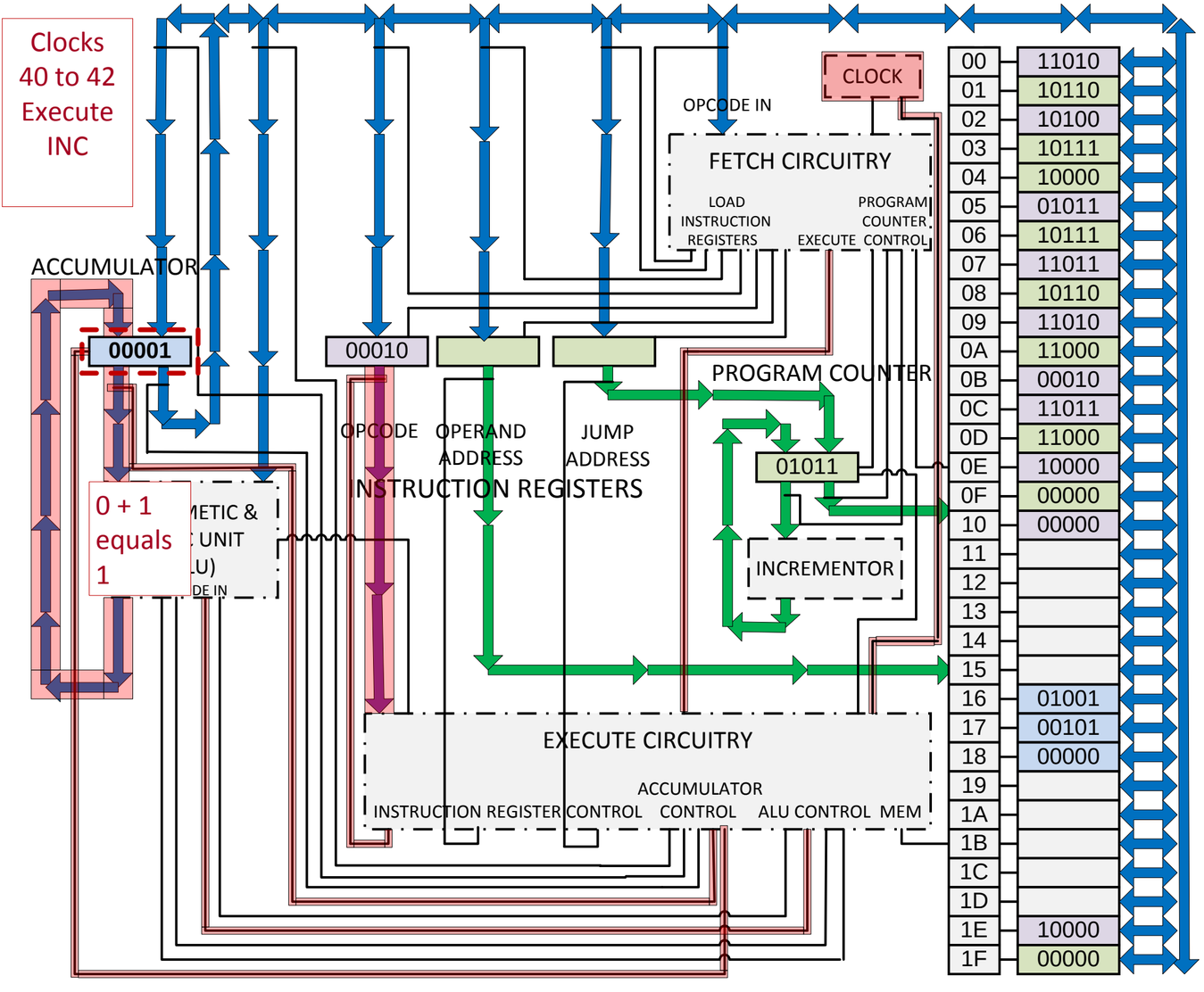
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



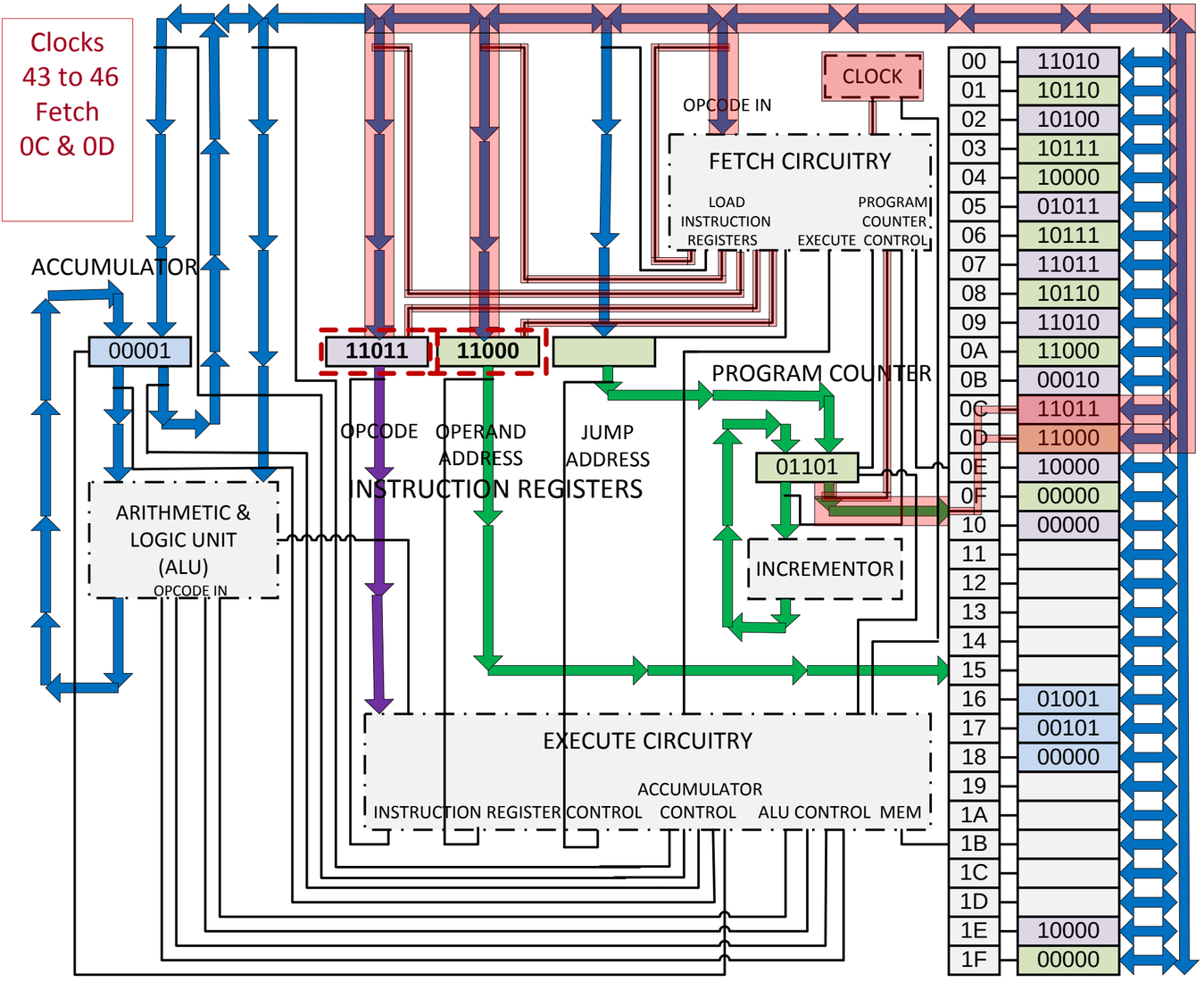
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



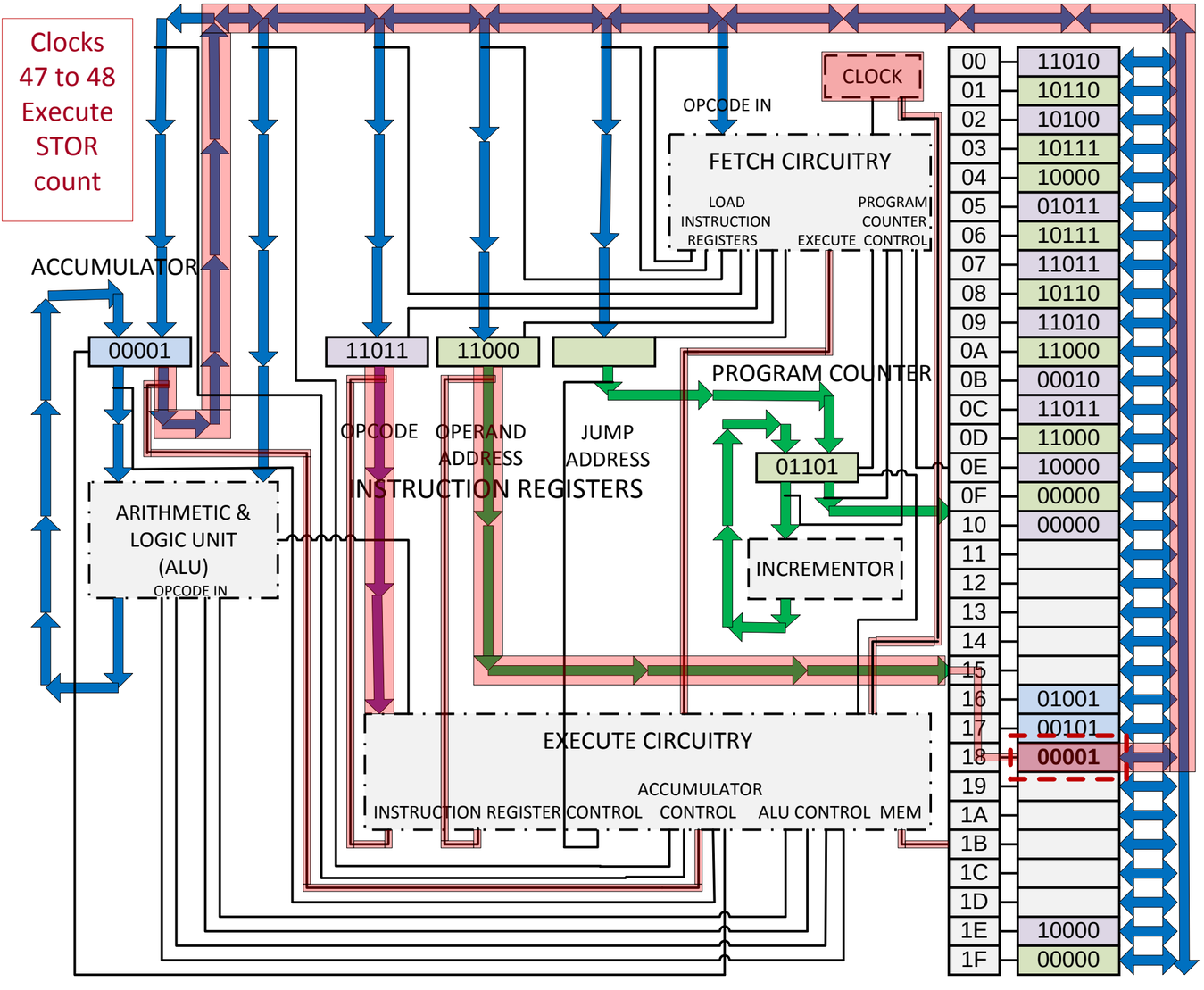
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



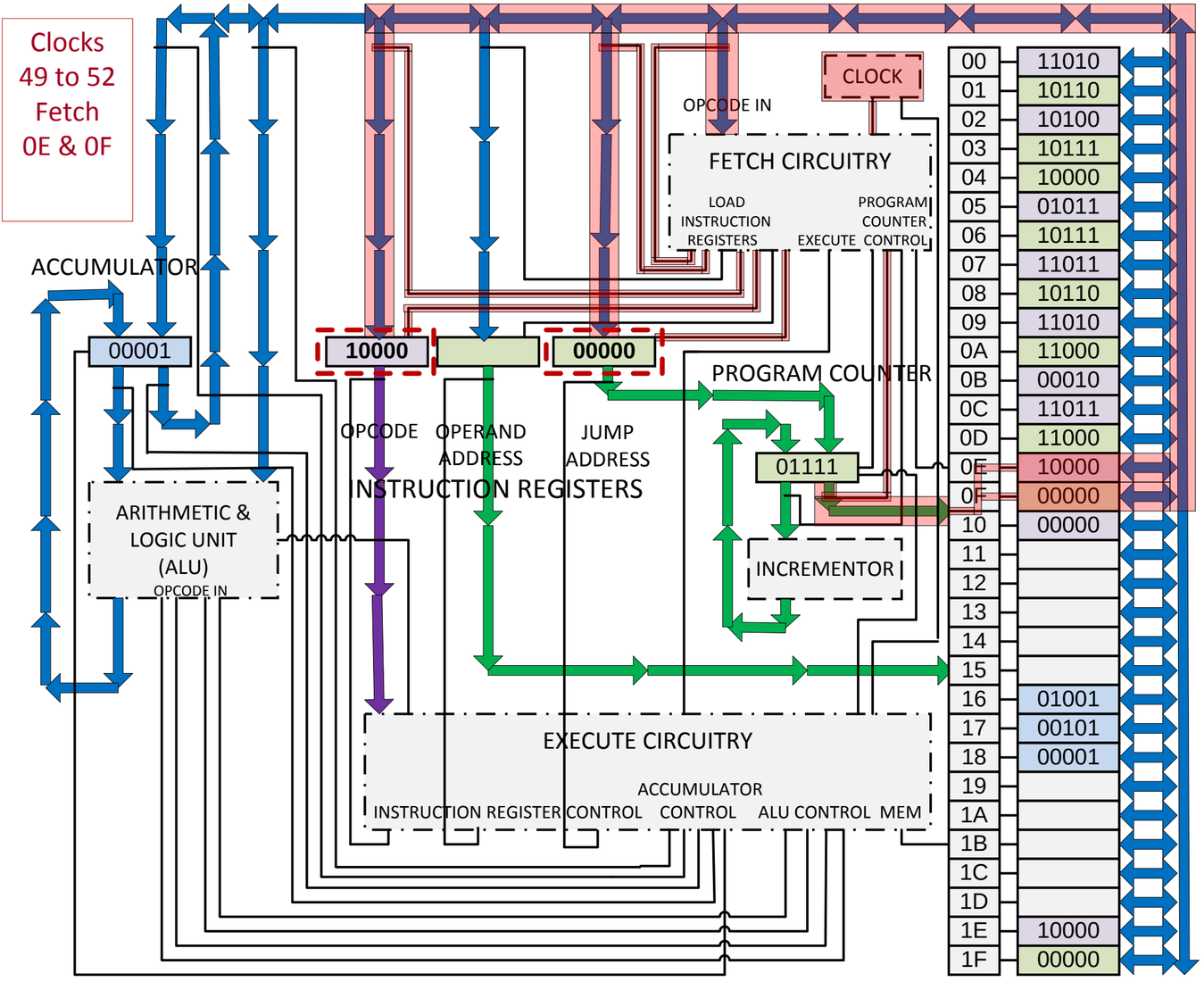
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



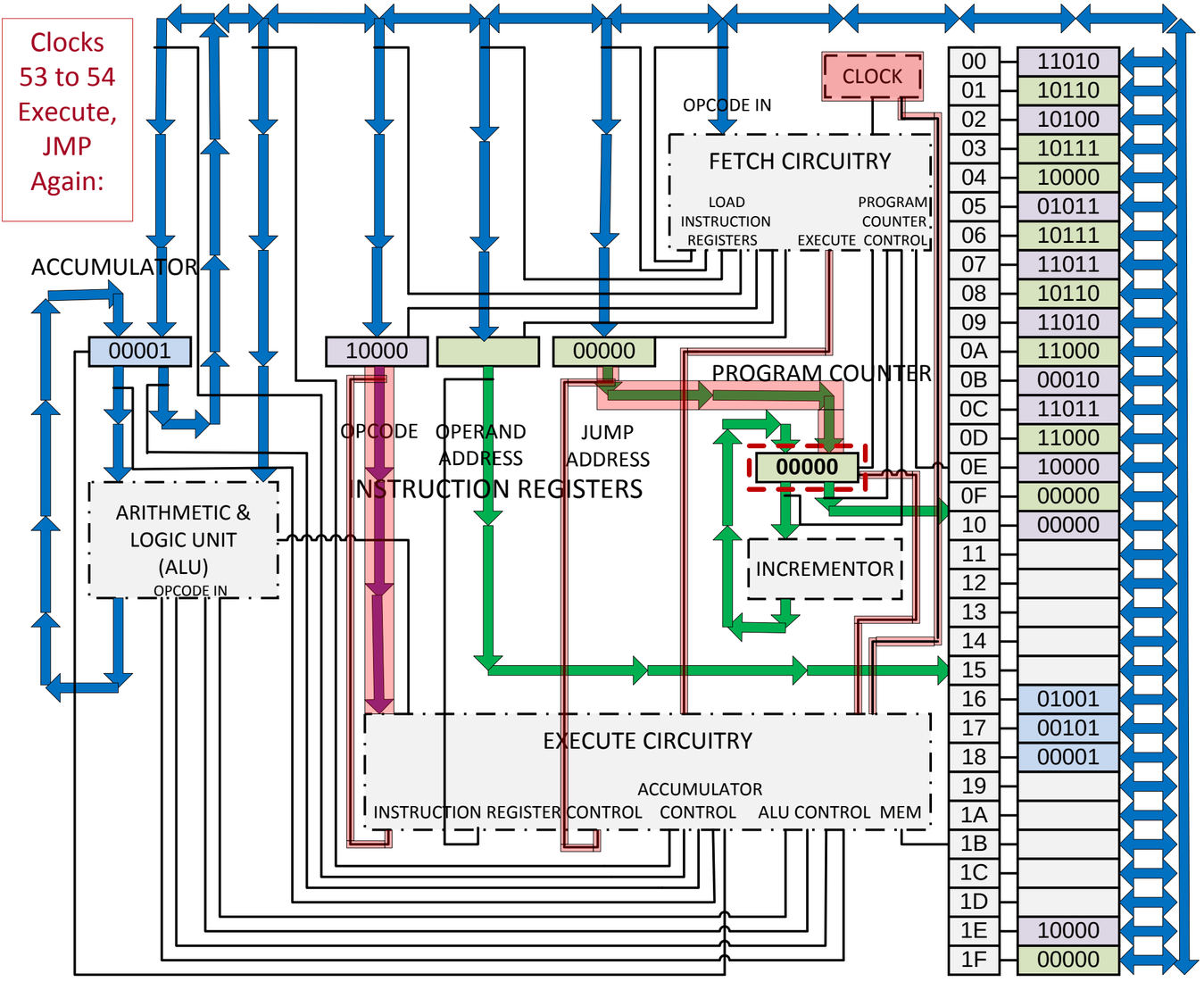
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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



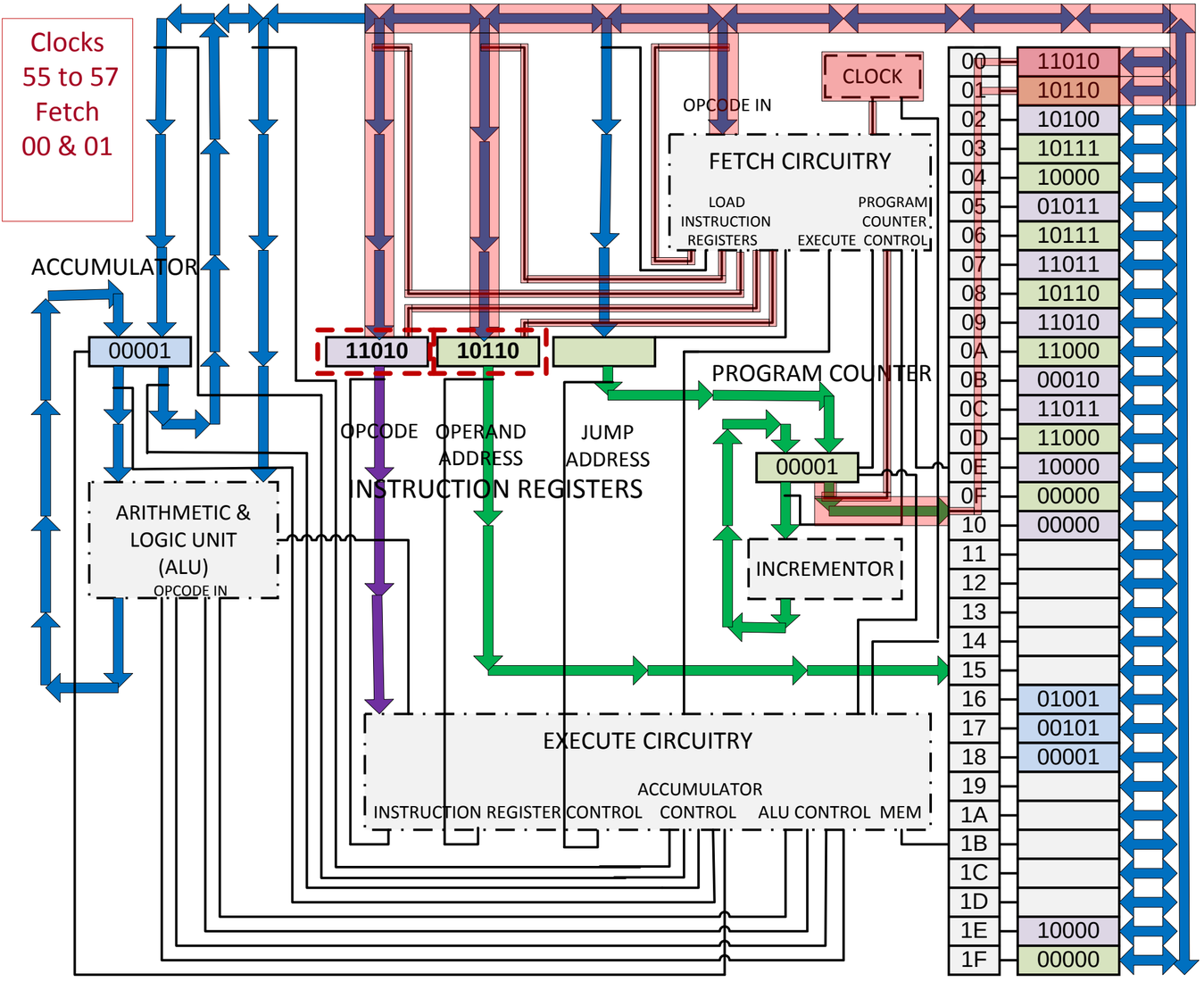
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



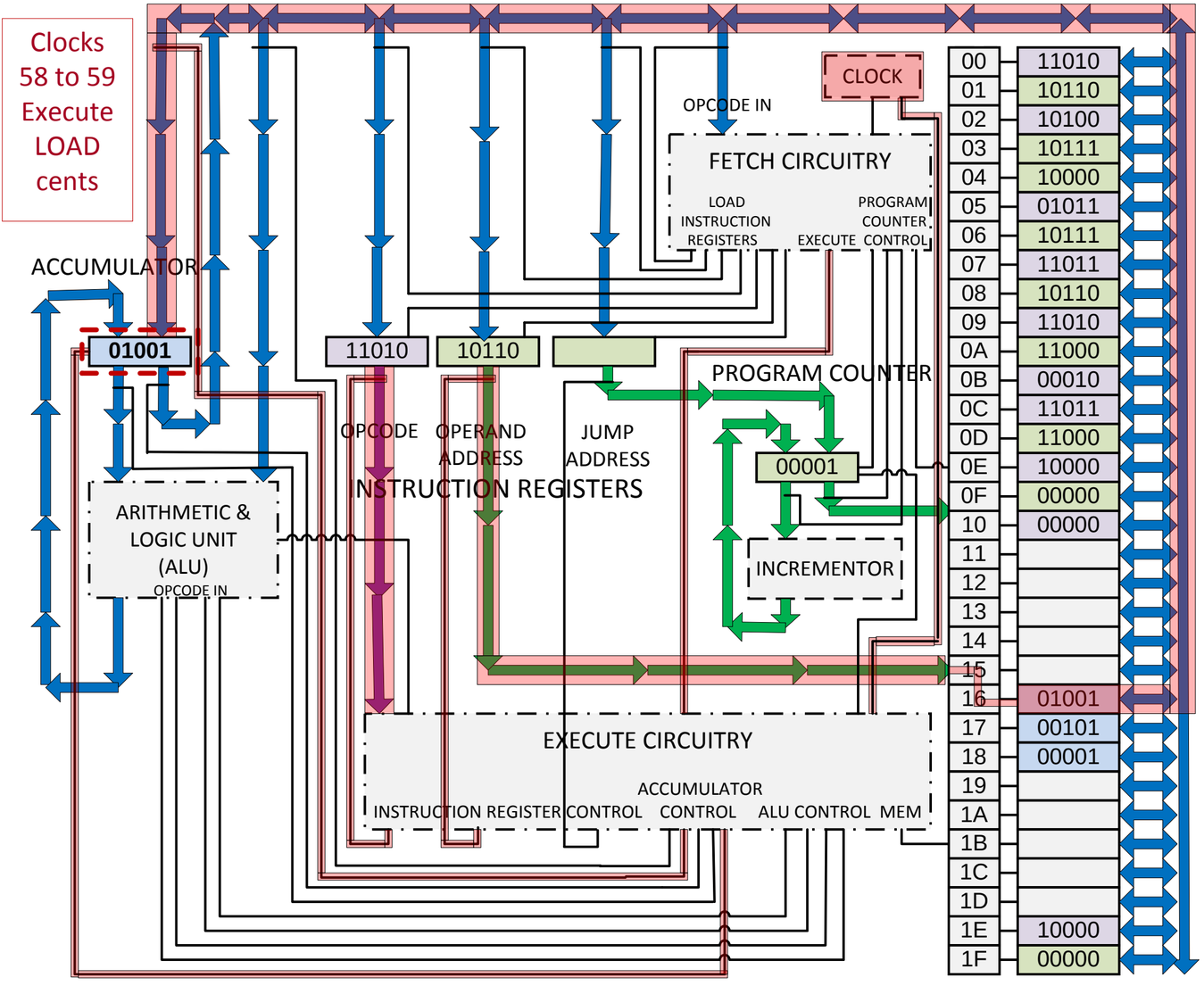
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



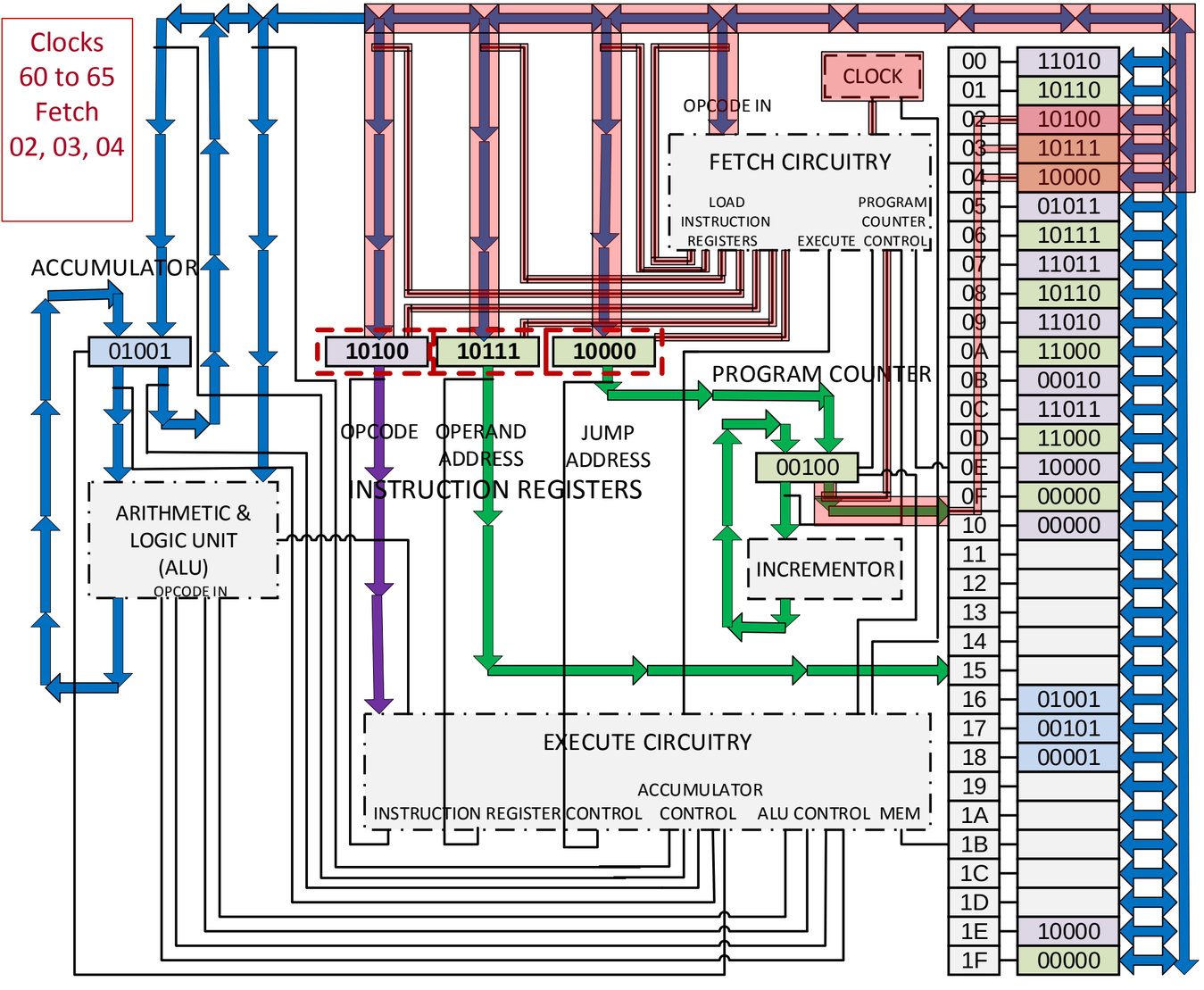
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



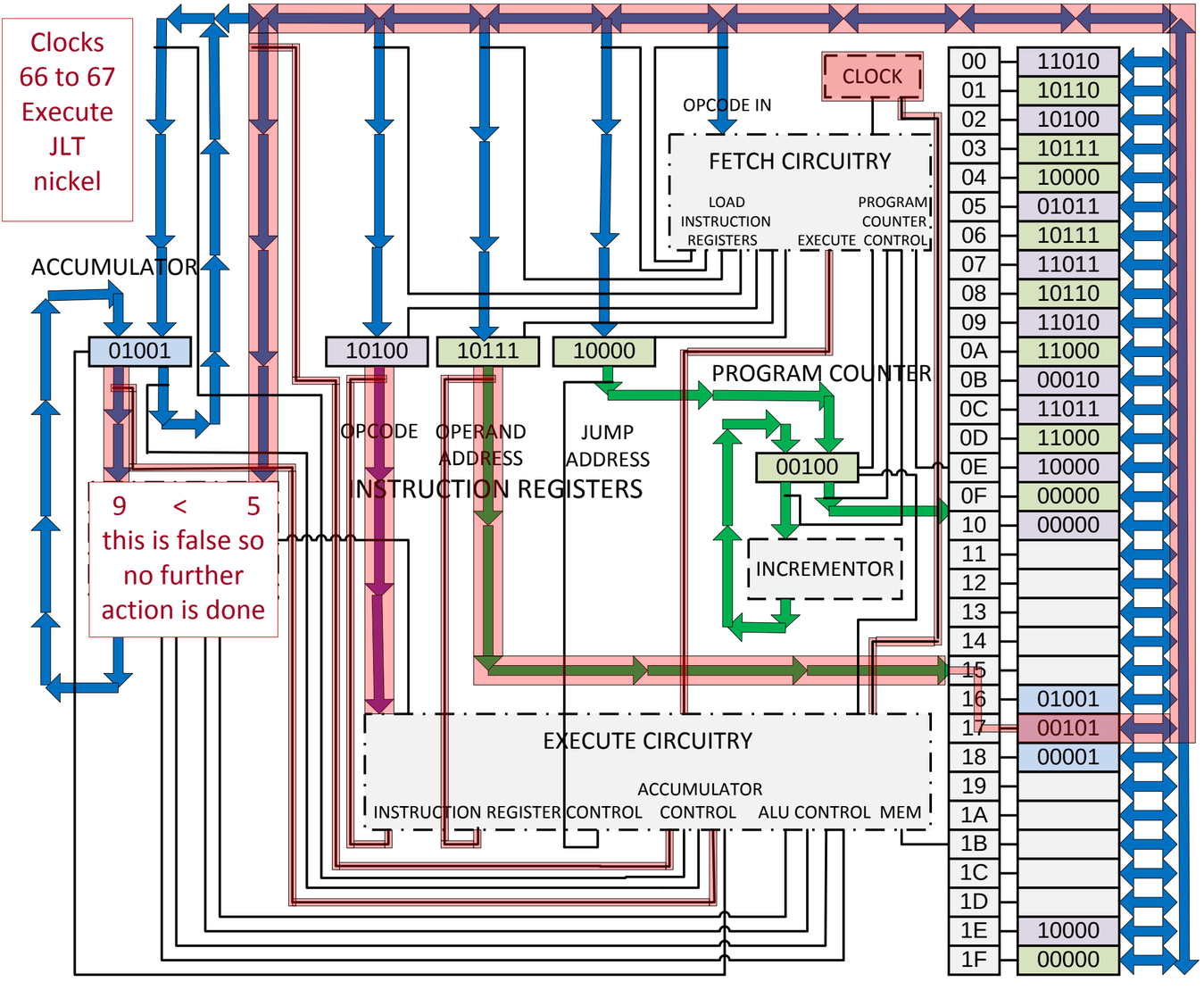
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



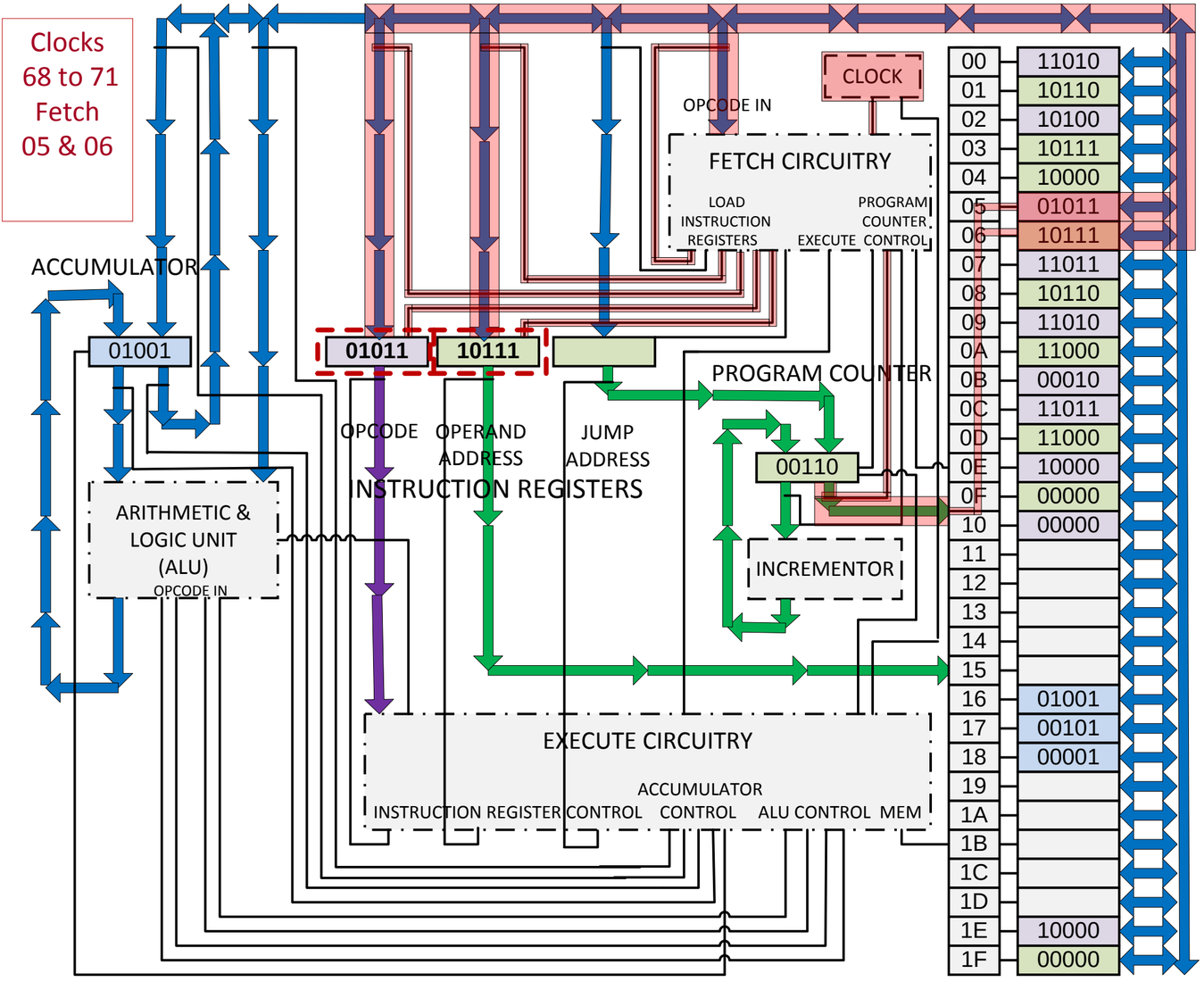
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



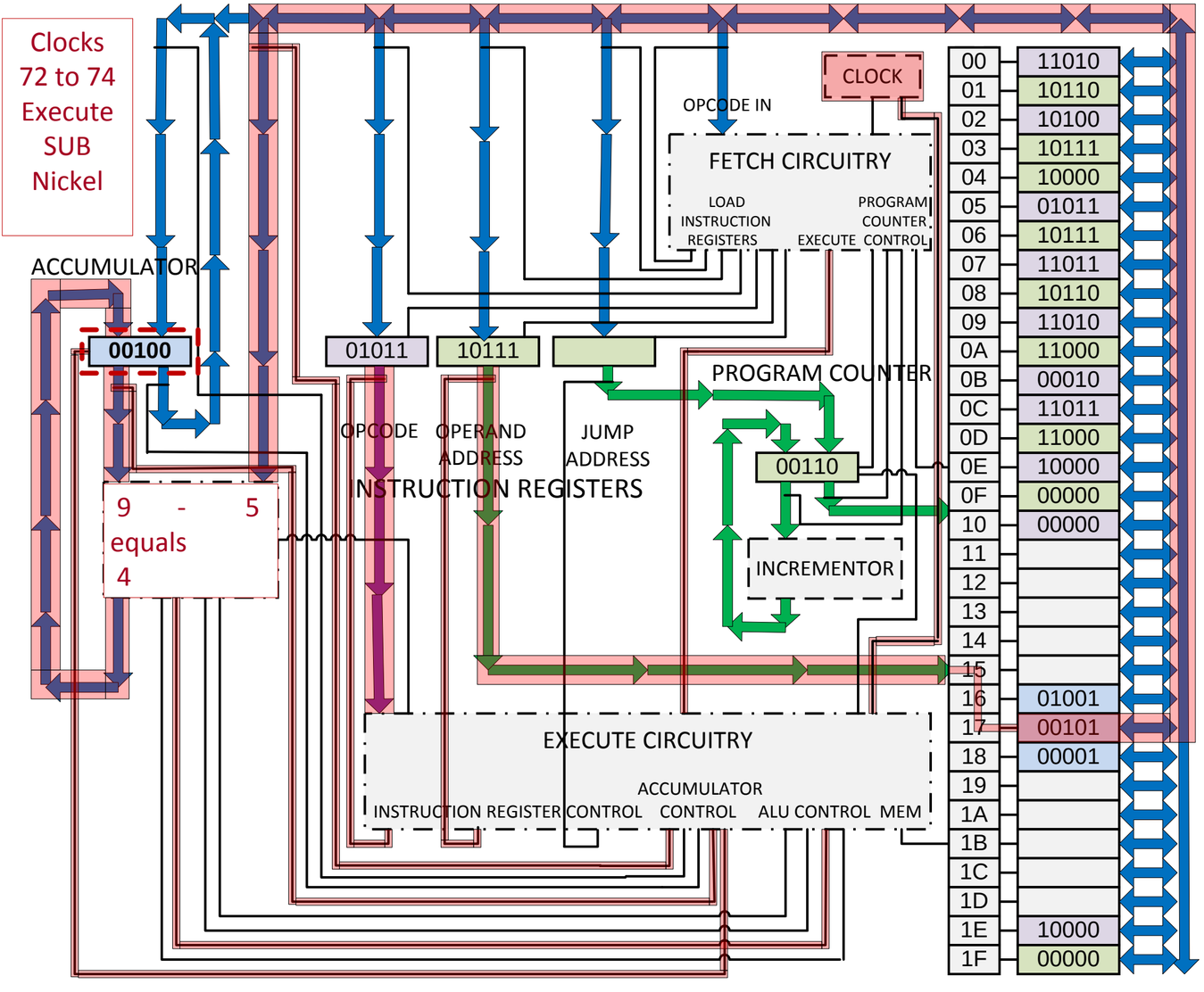
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



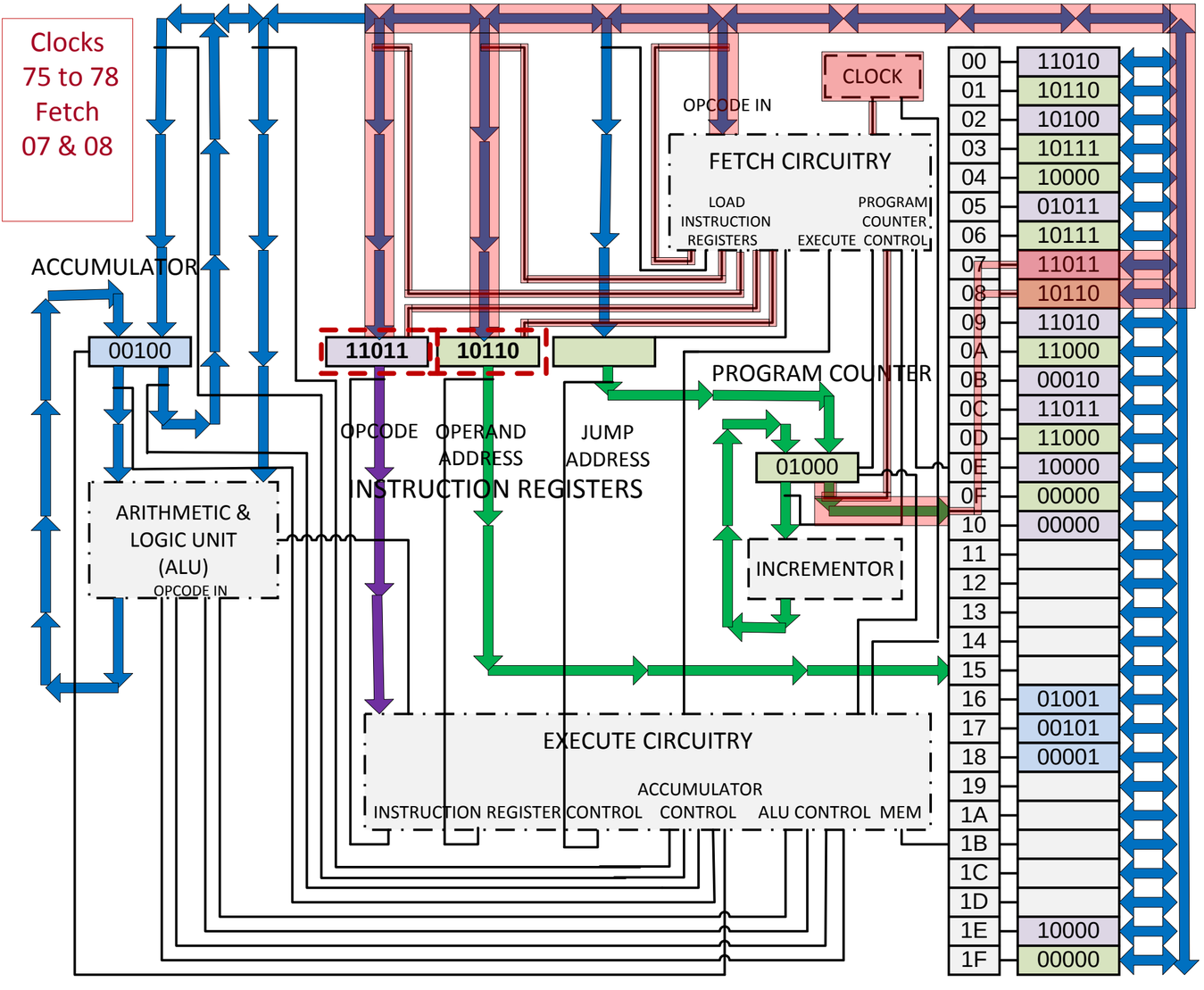
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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



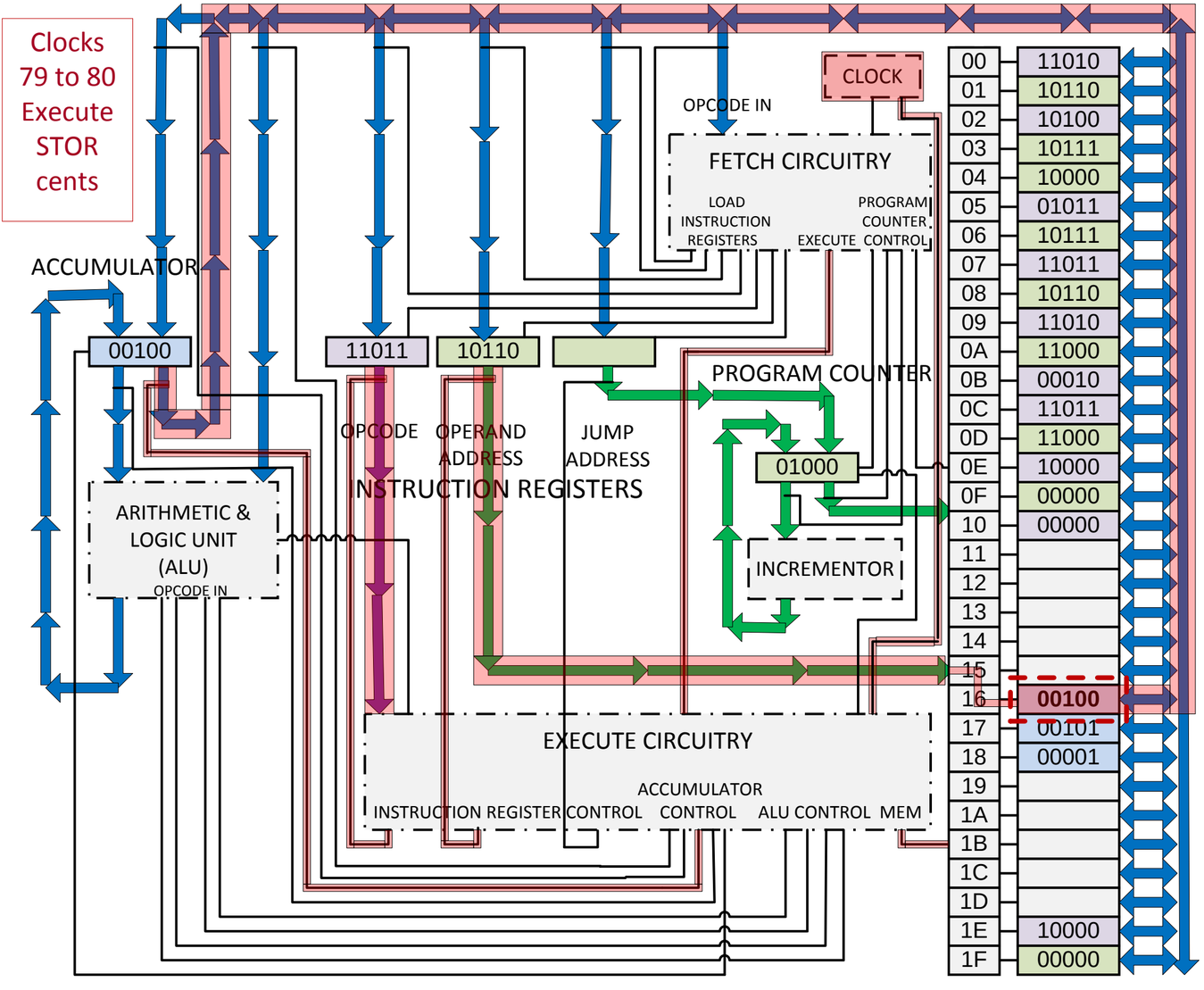
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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	9	# 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	1	# 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

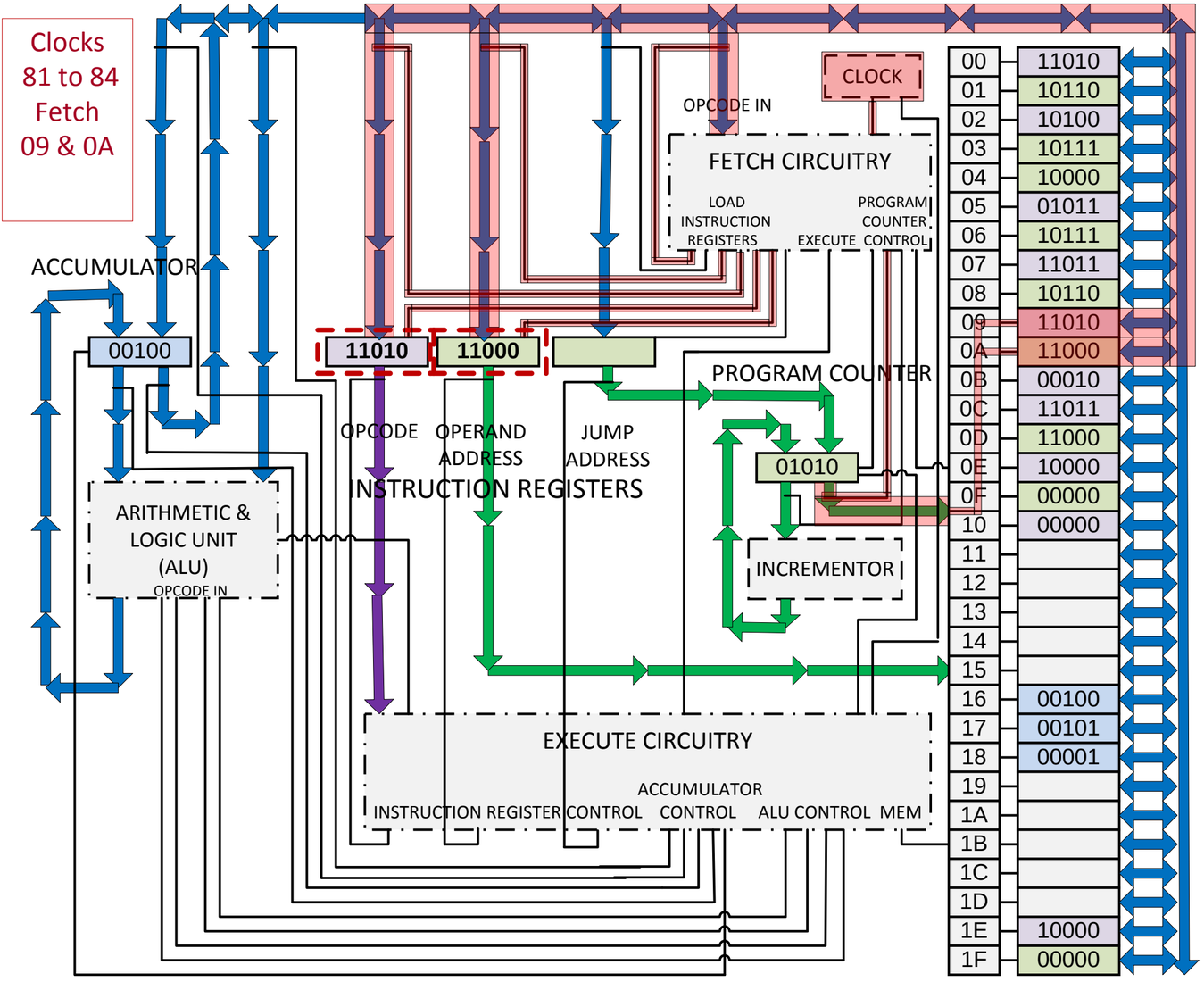


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	9	# 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	1	# 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

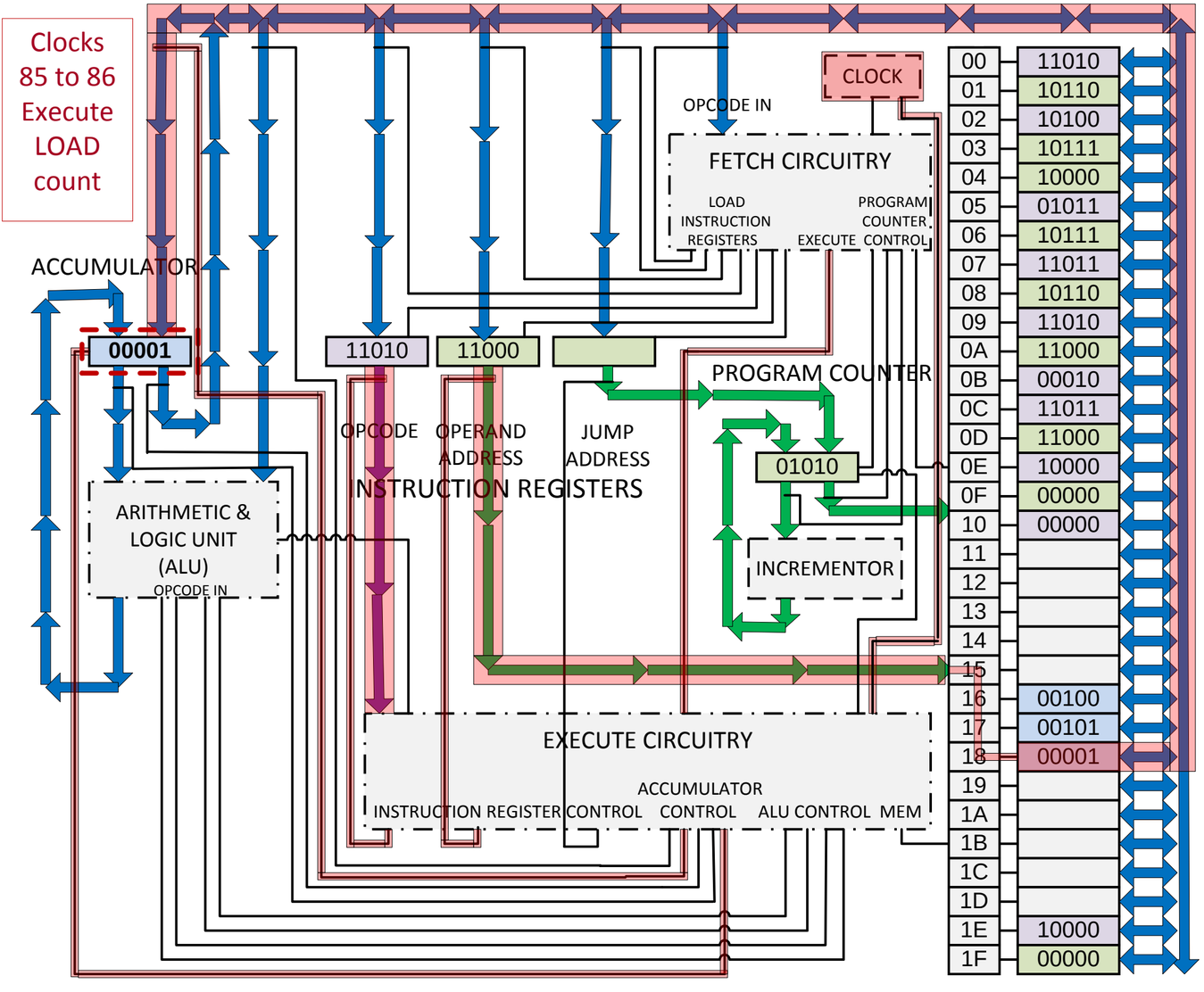


	Assembly Language Addresses:	Program:	Explanation	Hex
00	Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
01		cents		01
02		JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
03		nickel	# jump to the address ‘Done:’	03
04		Done:		04
05		SUB	# subtract the value at ‘nickel’ from the Accumulator	05
06		nickel	# and put the result back into the Accumulator	06
07		STOR	# copy the value in the Accumulator to ‘cents’	07
08		cents		08
09		LOAD	# copy the value at address ‘count’ to the Accumulator	09
0A		count		0A
0B		INC	# add 1 to the Accumulator	0B
0C		STOR	# copy the value in the Accumulator to ‘count’	0C
0D		count		0D
0E		JMP	# unconditional jump to the address ‘Again:’	0E
0F		Again:		0F
10	Done:	HALT	# stop the processor – end of program	10
11				11
12				12
13				13
14				14
15				15
16	cents	4	# variable -- the name ‘cents’ is the address and 14 is the value	16
17	nickel	5	# variable -- the name ‘cents’ is the address and 5 is the value	17
18	count	1	# variable -- the name ‘count’ is the address and 0 is the value	18
19				19
1A				1A
1B				1B
1C				1C
1D				1D
1E		JMP	# unconditional jump to the address ‘Again:’	1E
1F		Again:		1F

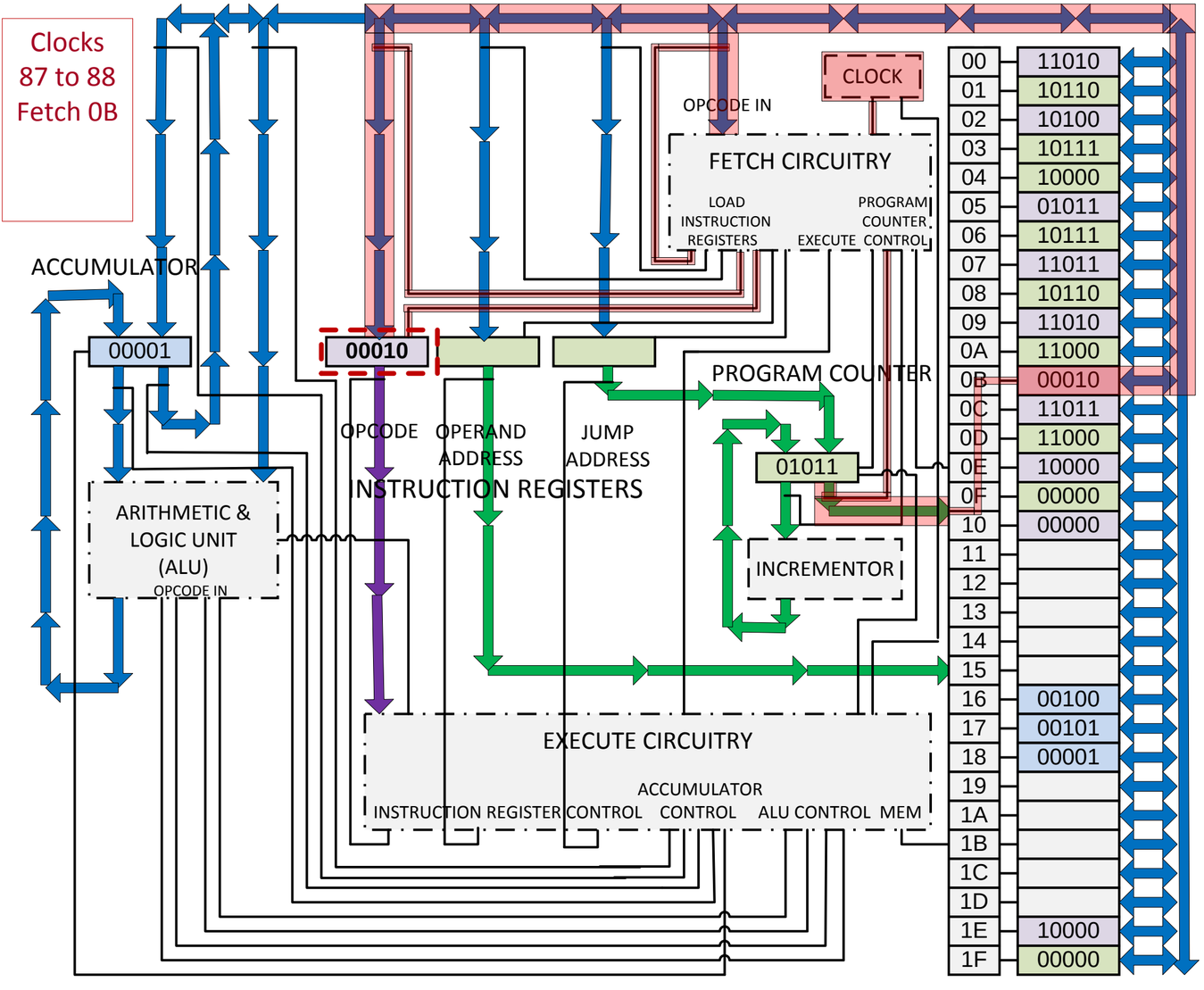
00	11010
01	10110
02	10100
03	10111
04	10000
05	01011
06	10111
07	11011
08	10110
09	11010
0A	11000
0B	00010
0C	11011
0D	11000
0E	10000
0F	00000
10	00000
11	
12	
13	
14	
15	
16	**00100**
17	00101
18	00001
19	
1A	
1B	
1C	
1D	
1E	10000
1F	00000



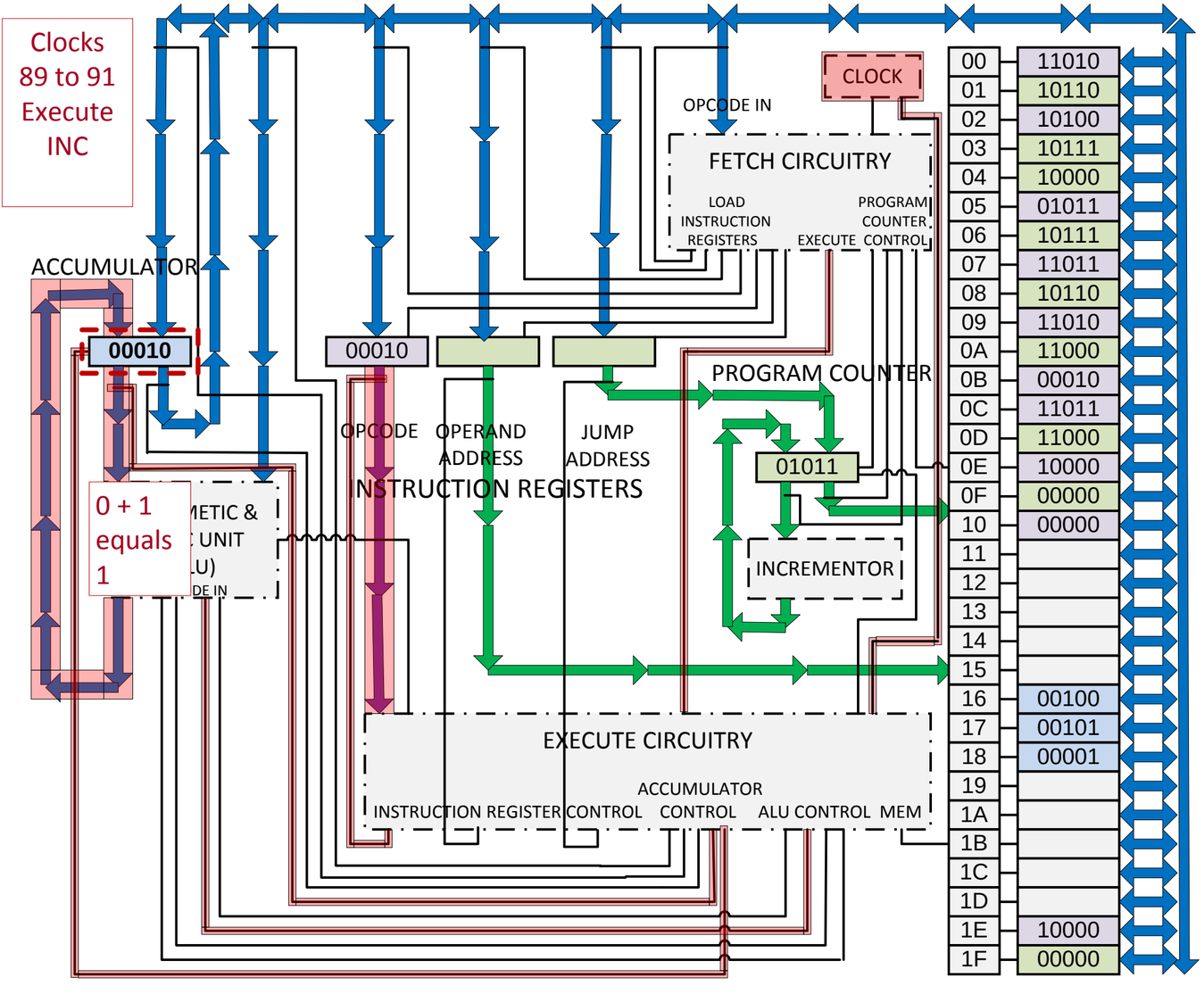
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	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	1	# 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



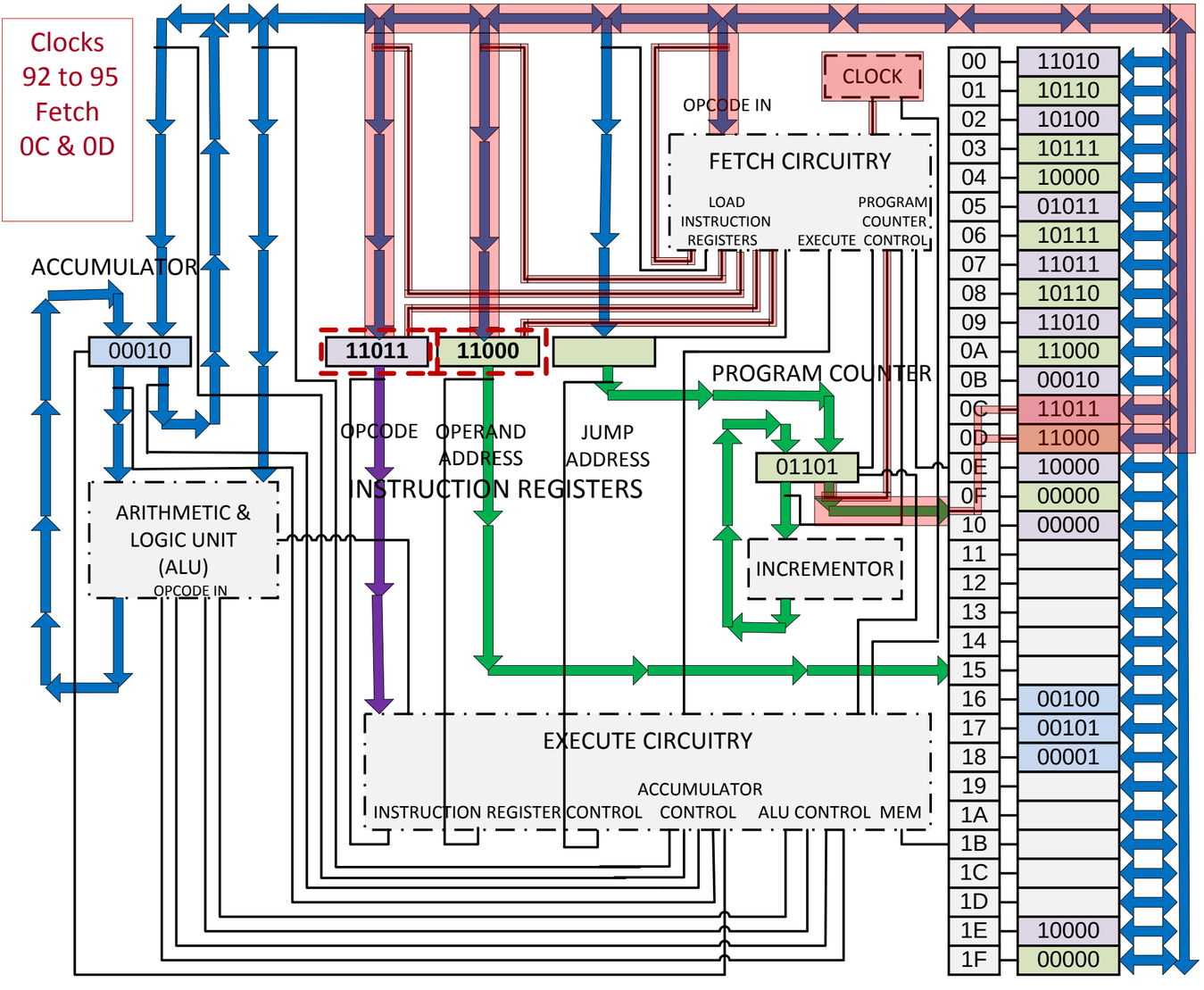
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	1	# 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



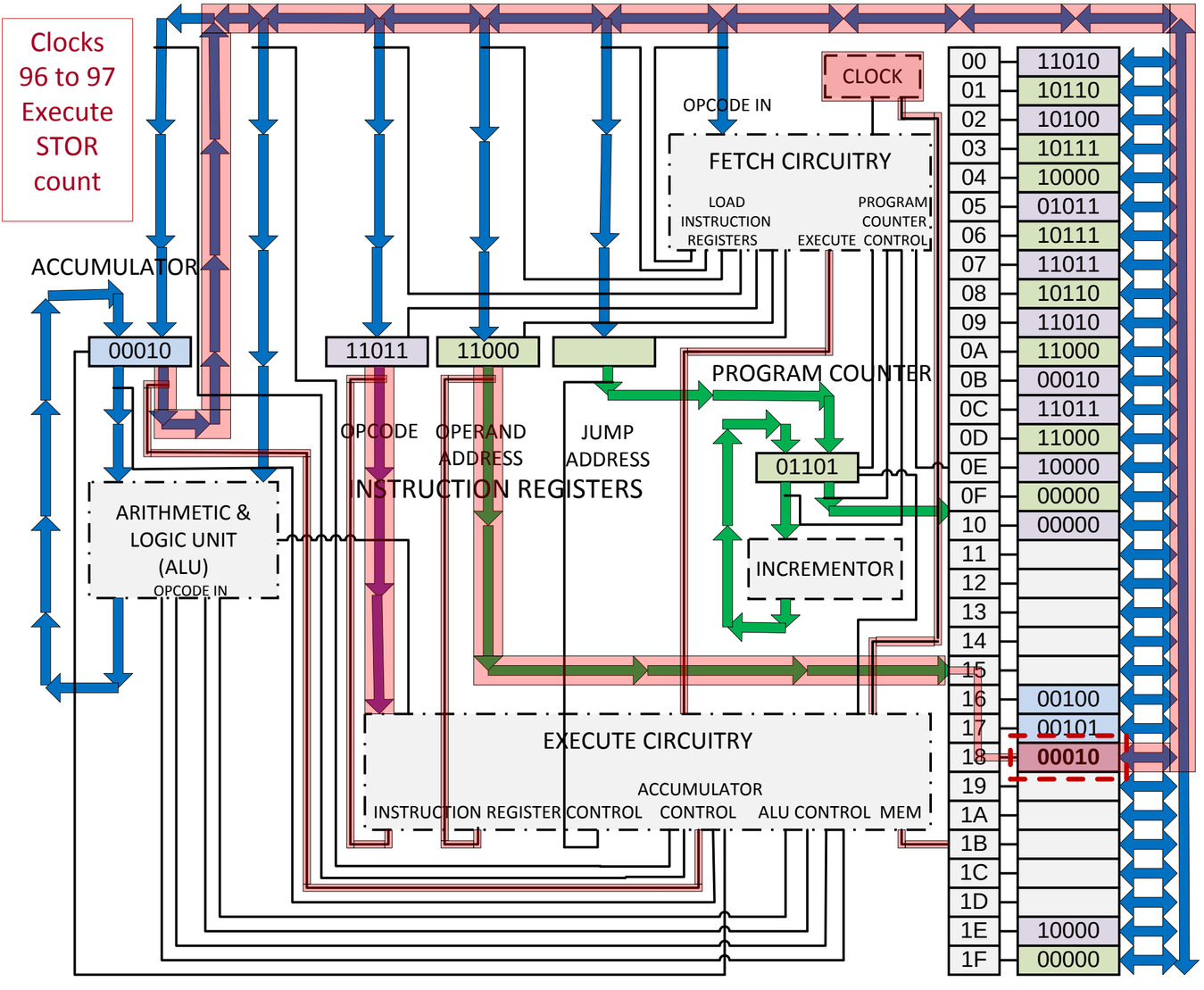
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	1	# 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



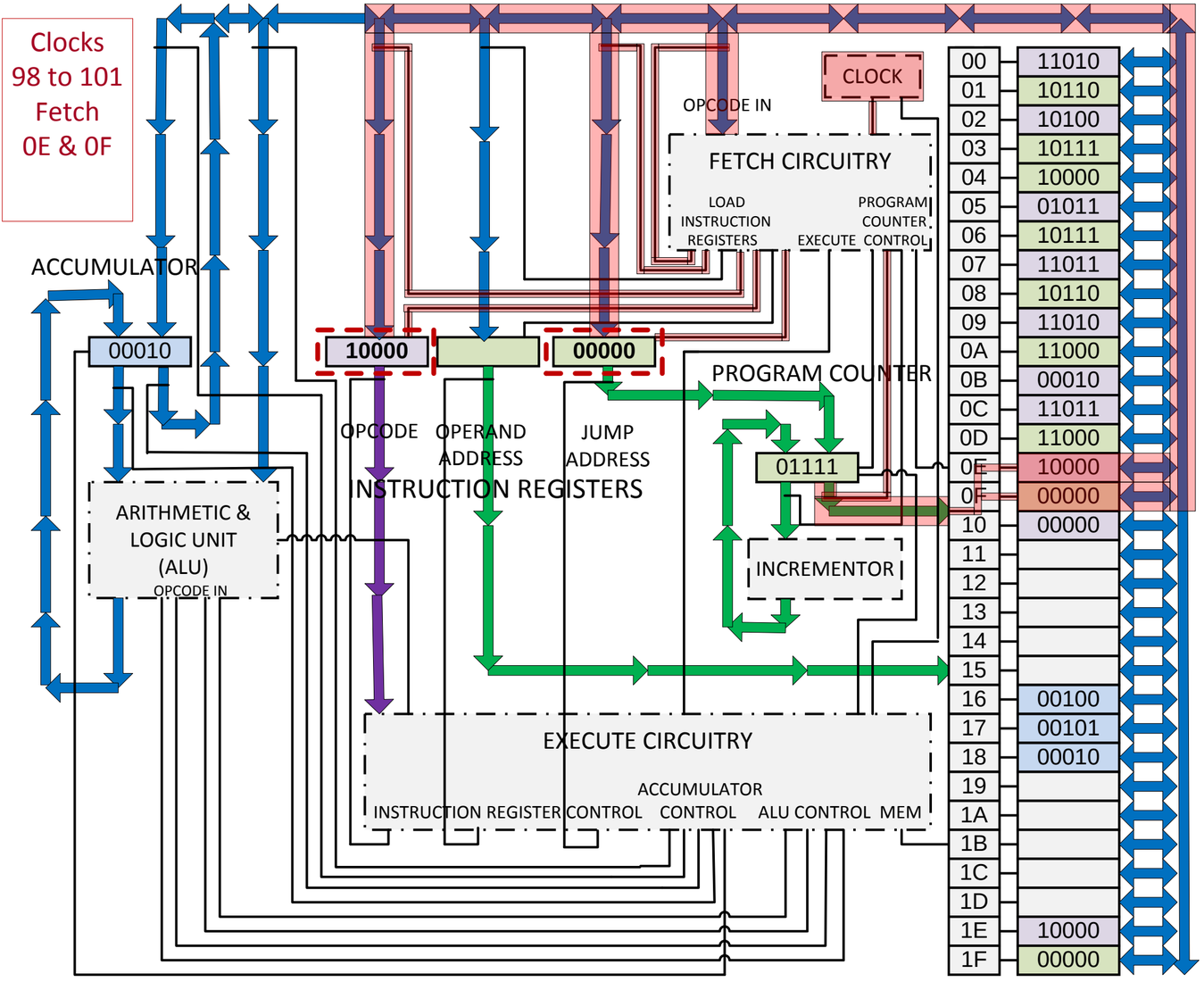
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	1	# 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



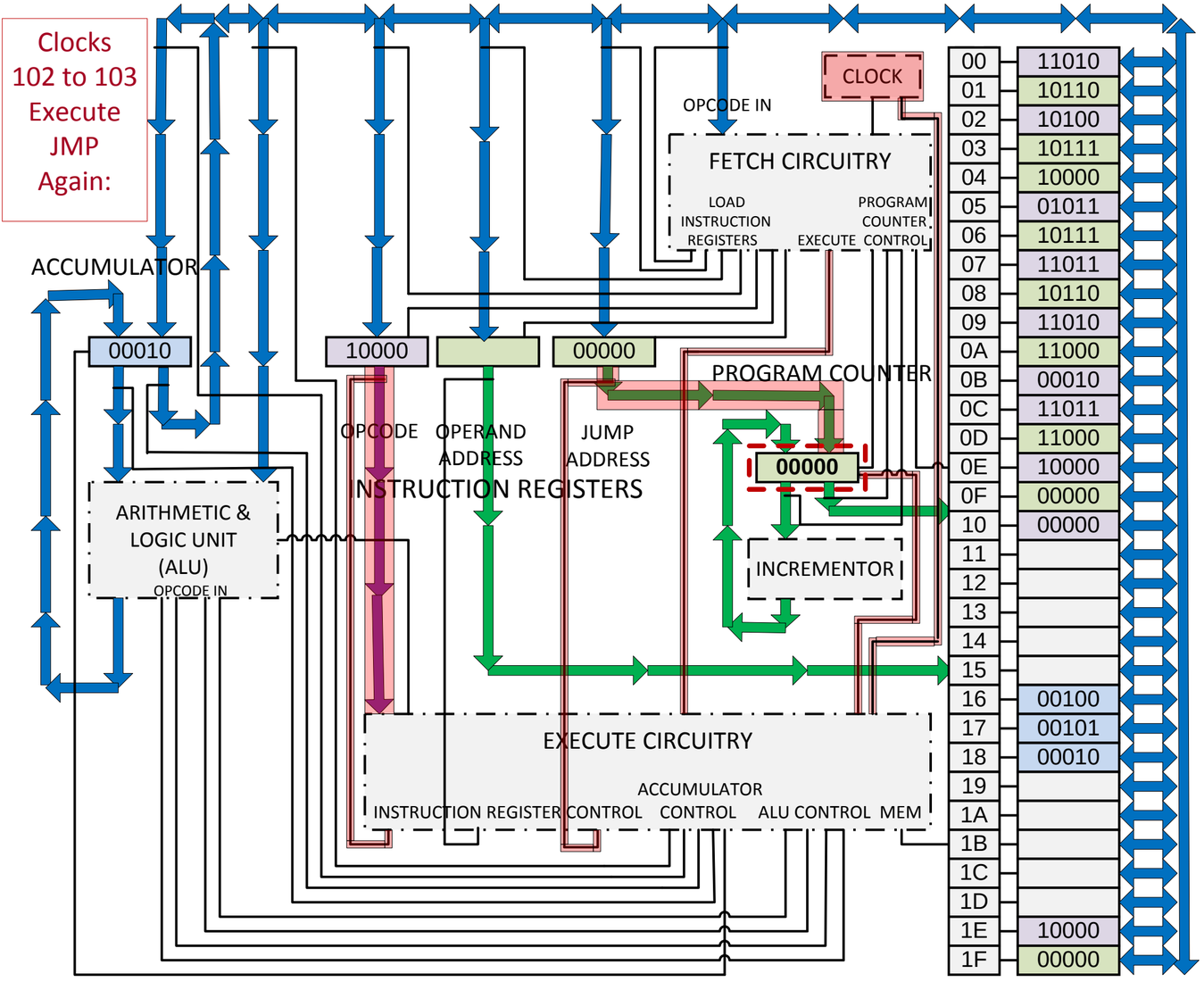
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator	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	1	# 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



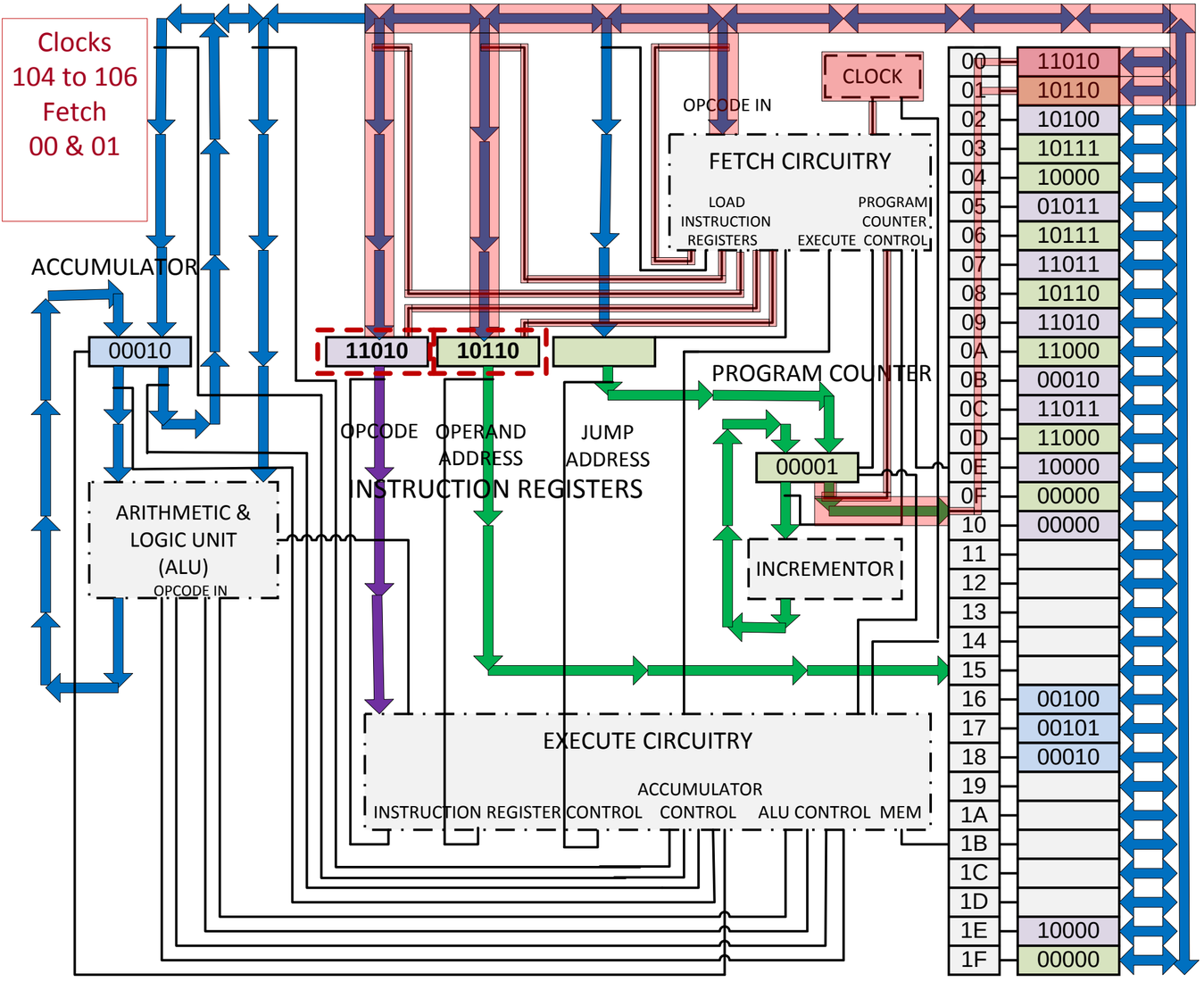
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



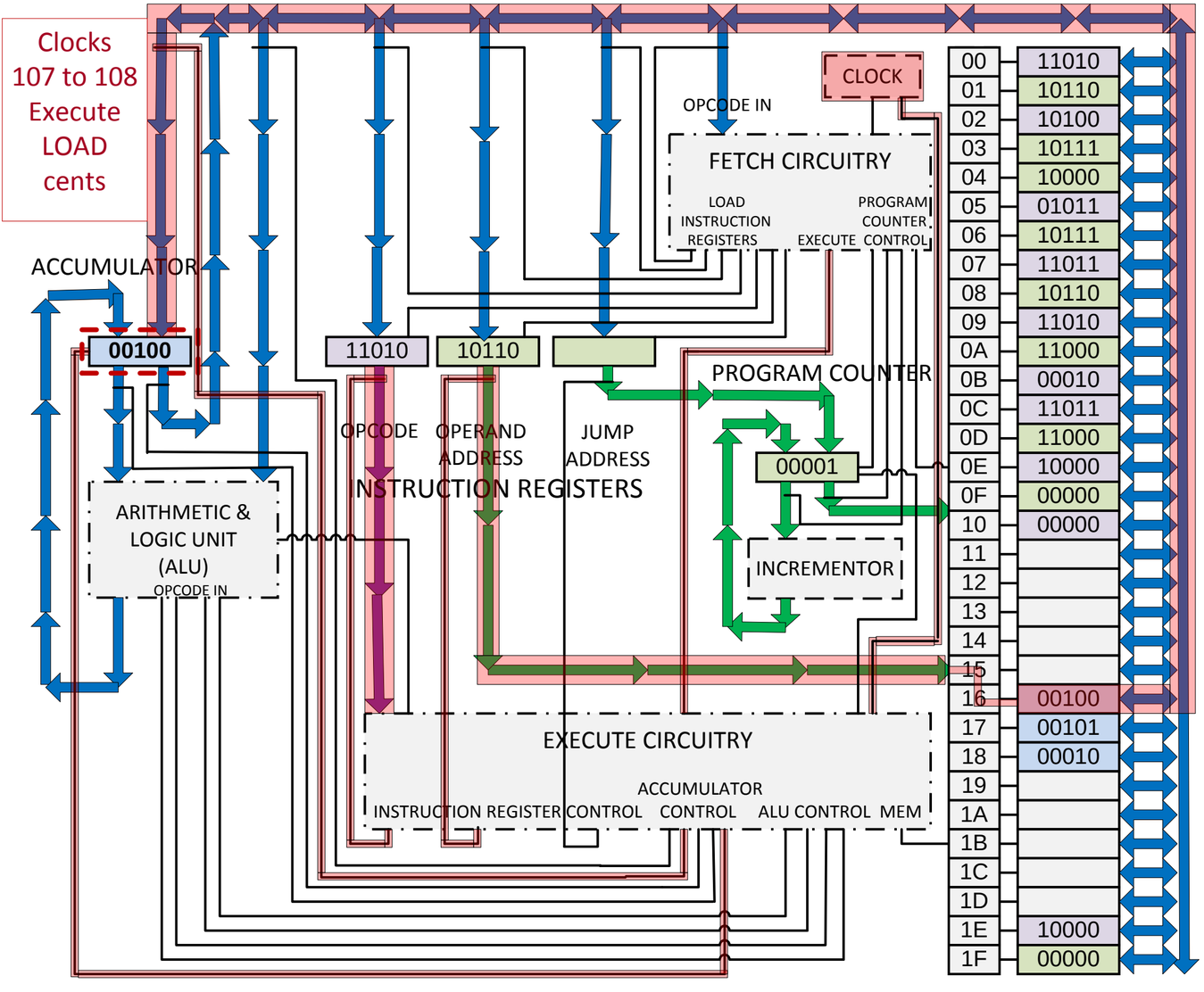
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



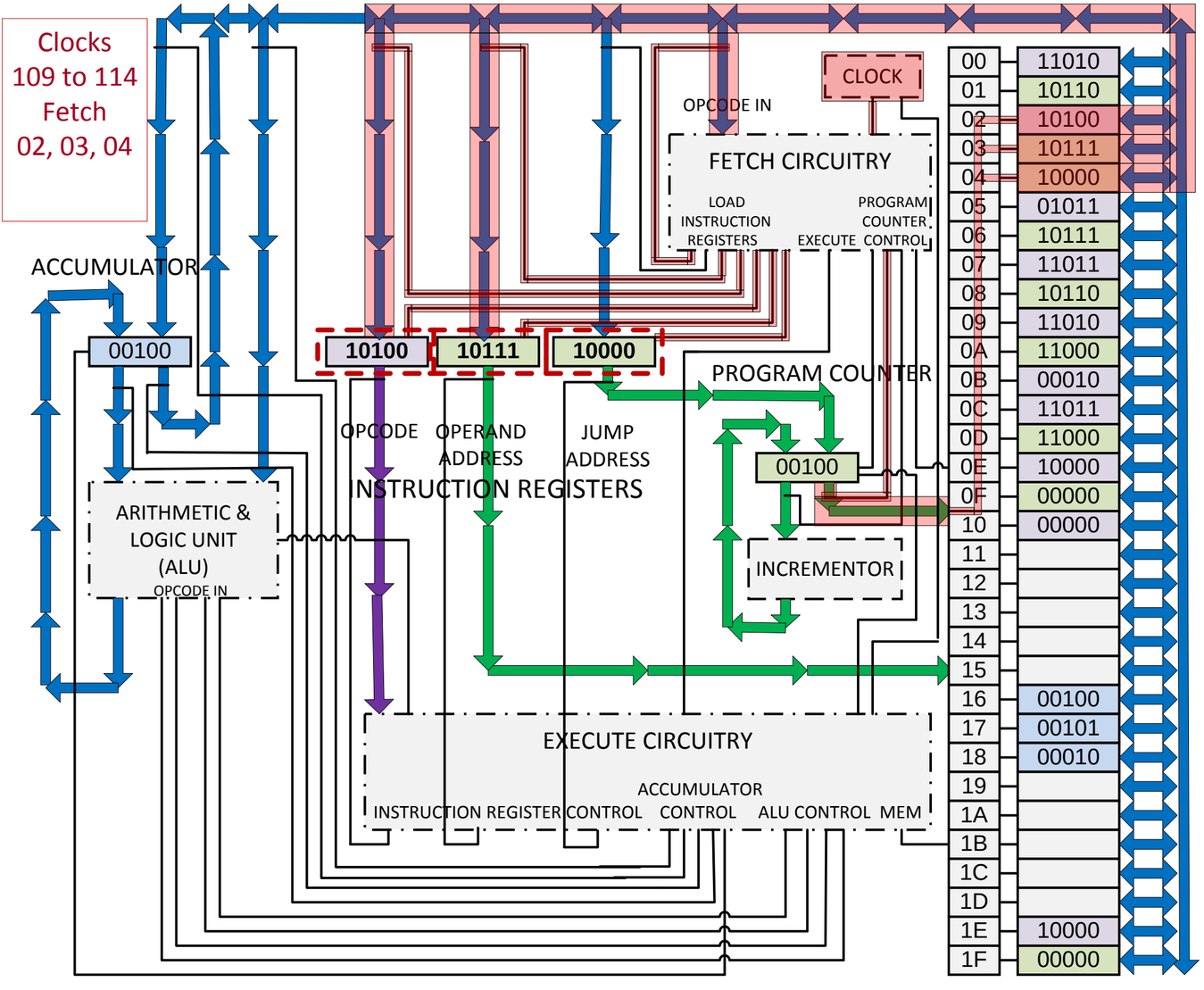
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



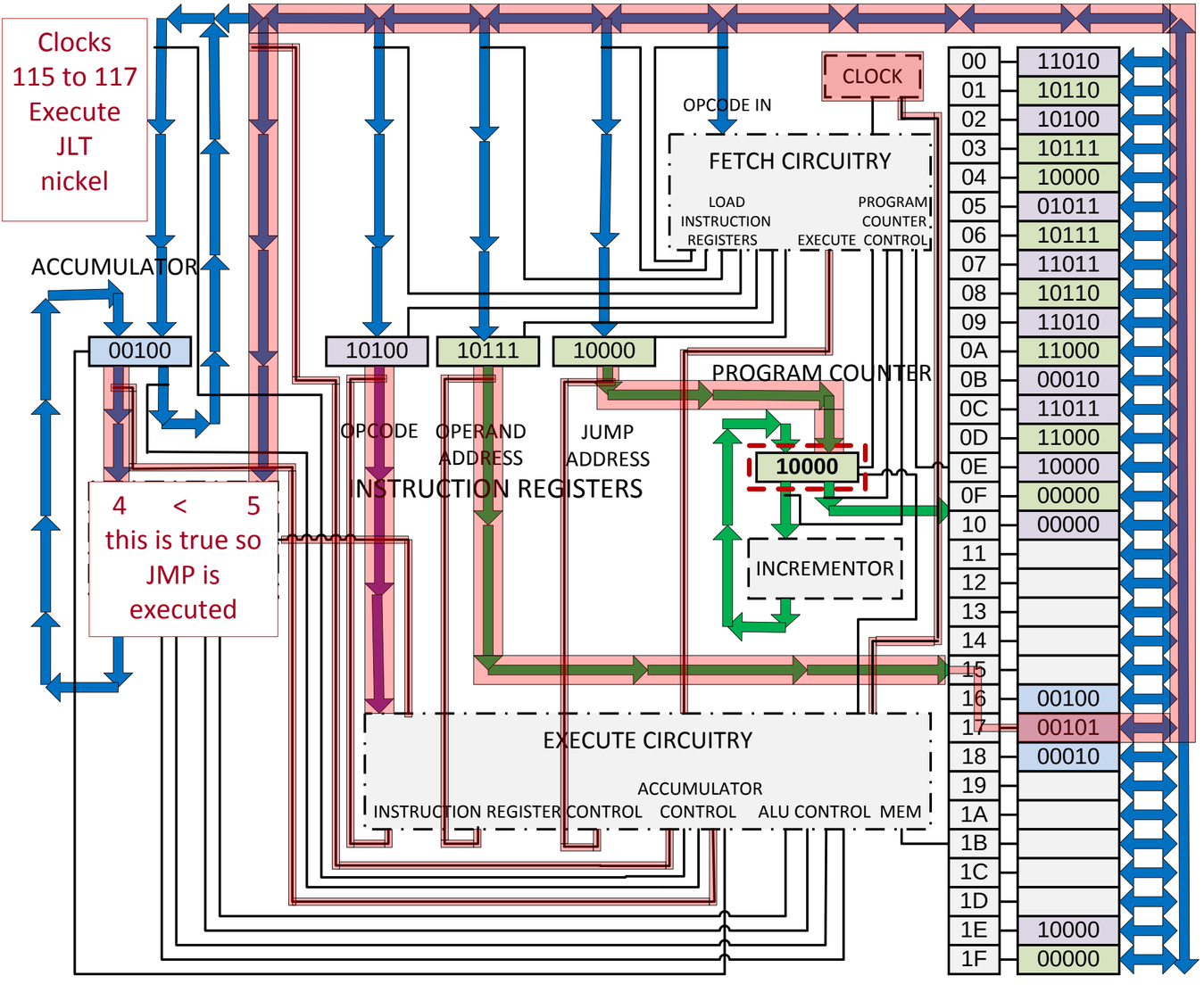
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



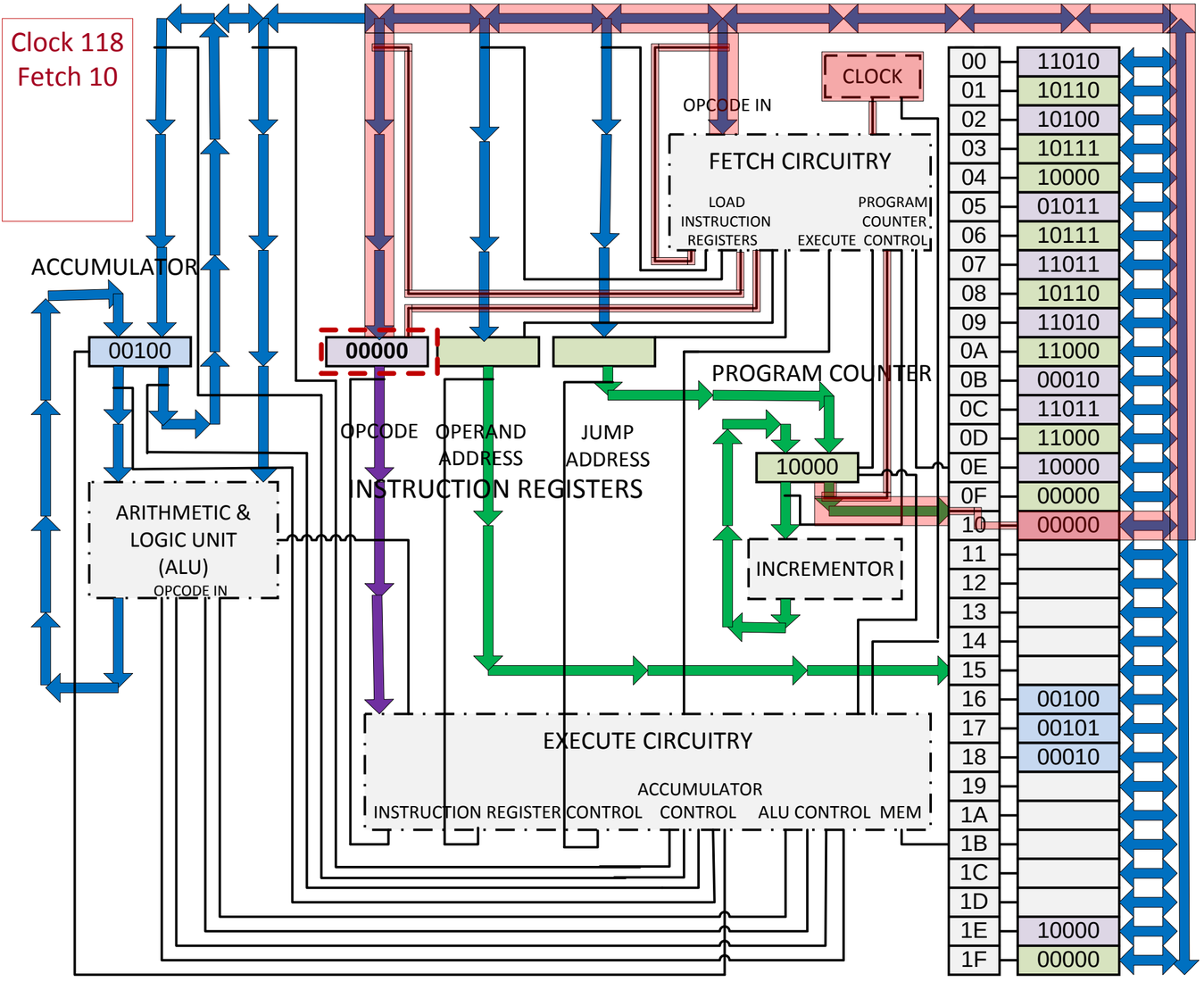
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



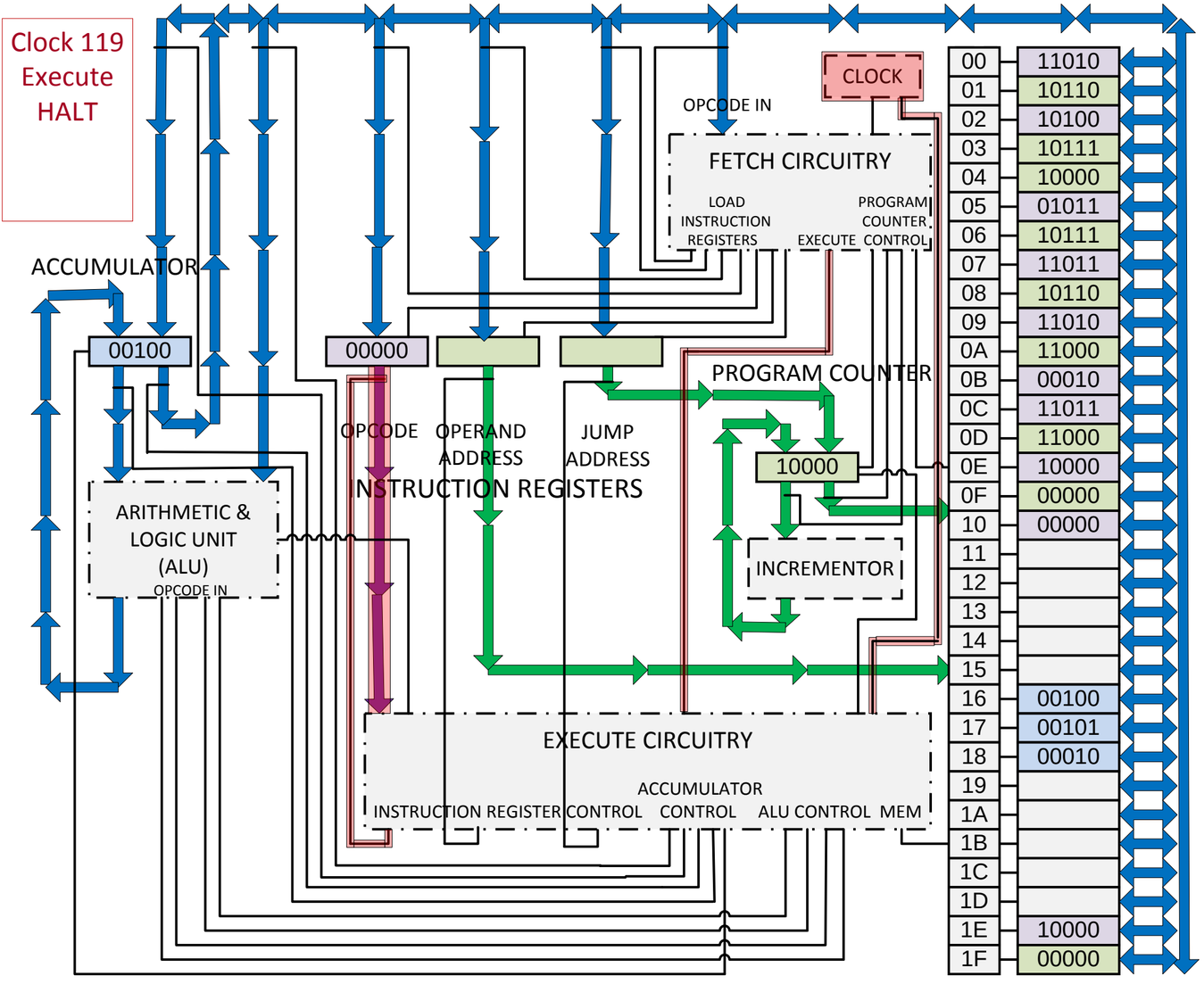
Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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

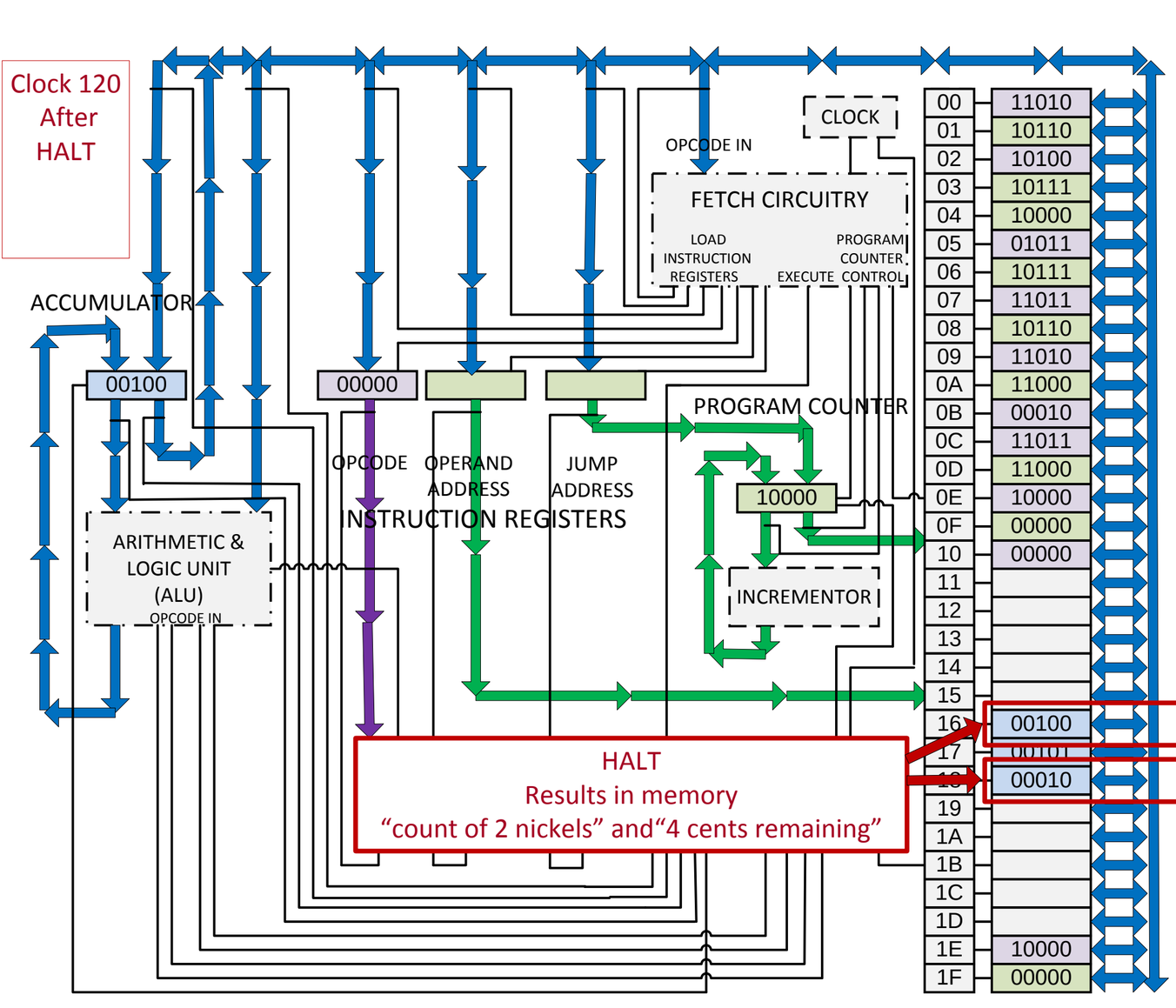


Assembly Language Addresses:	Program:	Explanation	Hex
Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
	cents		01
	JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
	nickel	# jump to the address ‘Done:’	03
	Done:		04
	SUB	# subtract the value at ‘nickel’ from the Accumulator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



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	

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
00	11010	Again:	LOAD	# copy the value at ‘cents’ to the Accumulator	00
01	10110		cents		01
02	10100		JLT	# if the value at ‘nickel’ is less than the Accumulator then	02
03	10111		nickel	# jump to the address ‘Done:’	03
04	10000		Done:		04
05	01011		SUB	# subtract the value at ‘nickel’ from the Accumulator	05
06	10111		nickel	# and put the result back into the Accumulator	06
07	11011		STOR	# copy the value in the Accumulator to ‘cents’	07
08	10110		cents		08
09	11010		LOAD	# copy the value at address ‘count’ to the Accumulator	09
0A	11000		count		0A
0B	00010		INC	# add 1 to the Accumulator	0B
0C	11011		STOR	# copy the value in the Accumulator to ‘count’	0C
0D	11000		count		0D
0E	10000		JMP	# unconditional jump to the address ‘Again:’	0E
0F	00000		Again:		0F
10	00000	Done:	HALT	# stop the processor – end of program	10
11					11
12					12
13					13
14					14
15					15
16	00100	cents	4	# variable -- the name ‘cents’ is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name ‘nickel’ is the address and 5 is the value	17
18	00010	count	2	# 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		JMP	# unconditional jump to the address ‘Again:’	1E
1F	00000		Again:		1F

Next Presentation:
Memory, ALU, and Control Circuitry

End of Presentation