

Assembly Language  
Addresses: Program:

Assembly Language	Addresses: Program:
Again:	LOAD
	cents
	JLT
	nickel
	Done:
	SUB
	nickel
	STOR
	cents
	LOAD
	count
	INC
	STOR
	count
	JMP
	Again:
Done:	HALT
cents	14
nickel	5
count	0
	JMP
	Again:

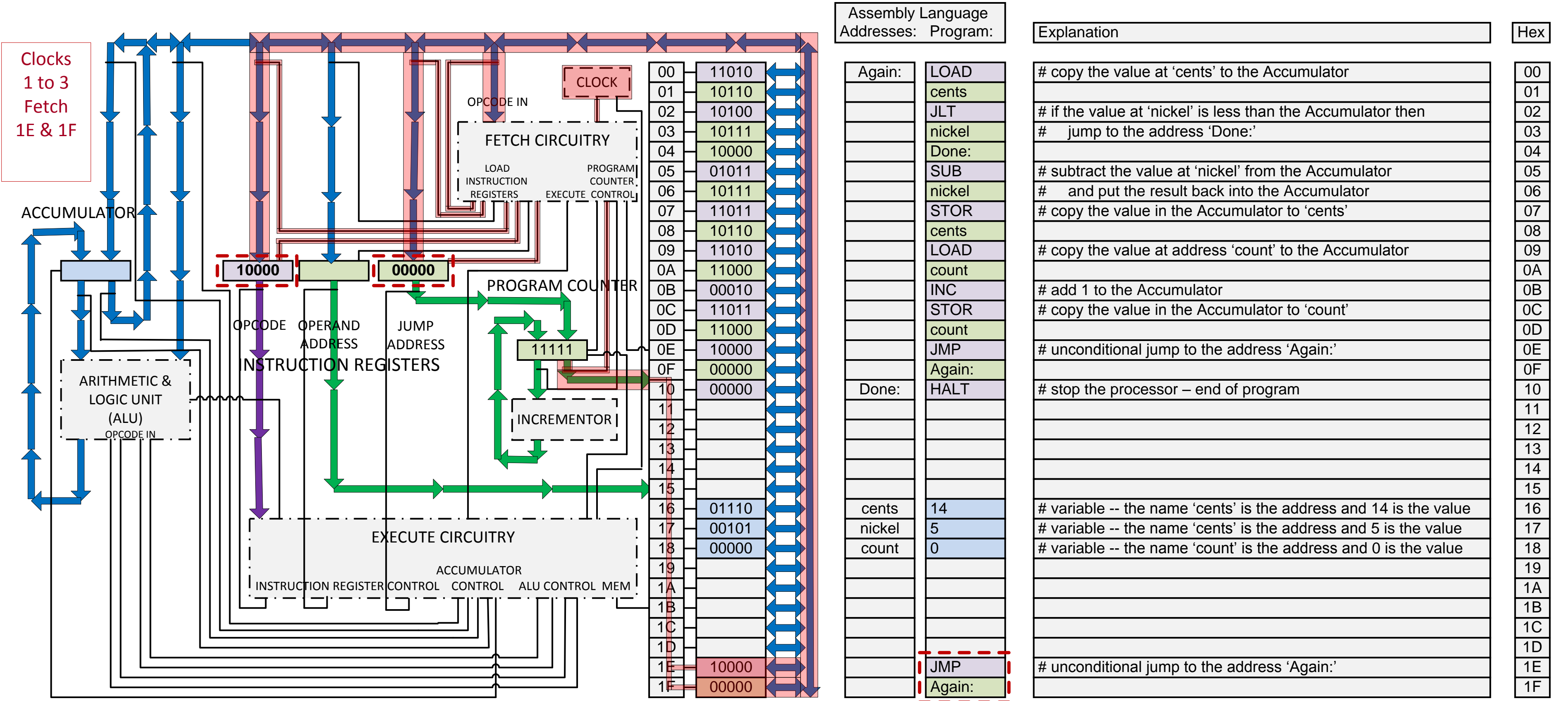
Explanation

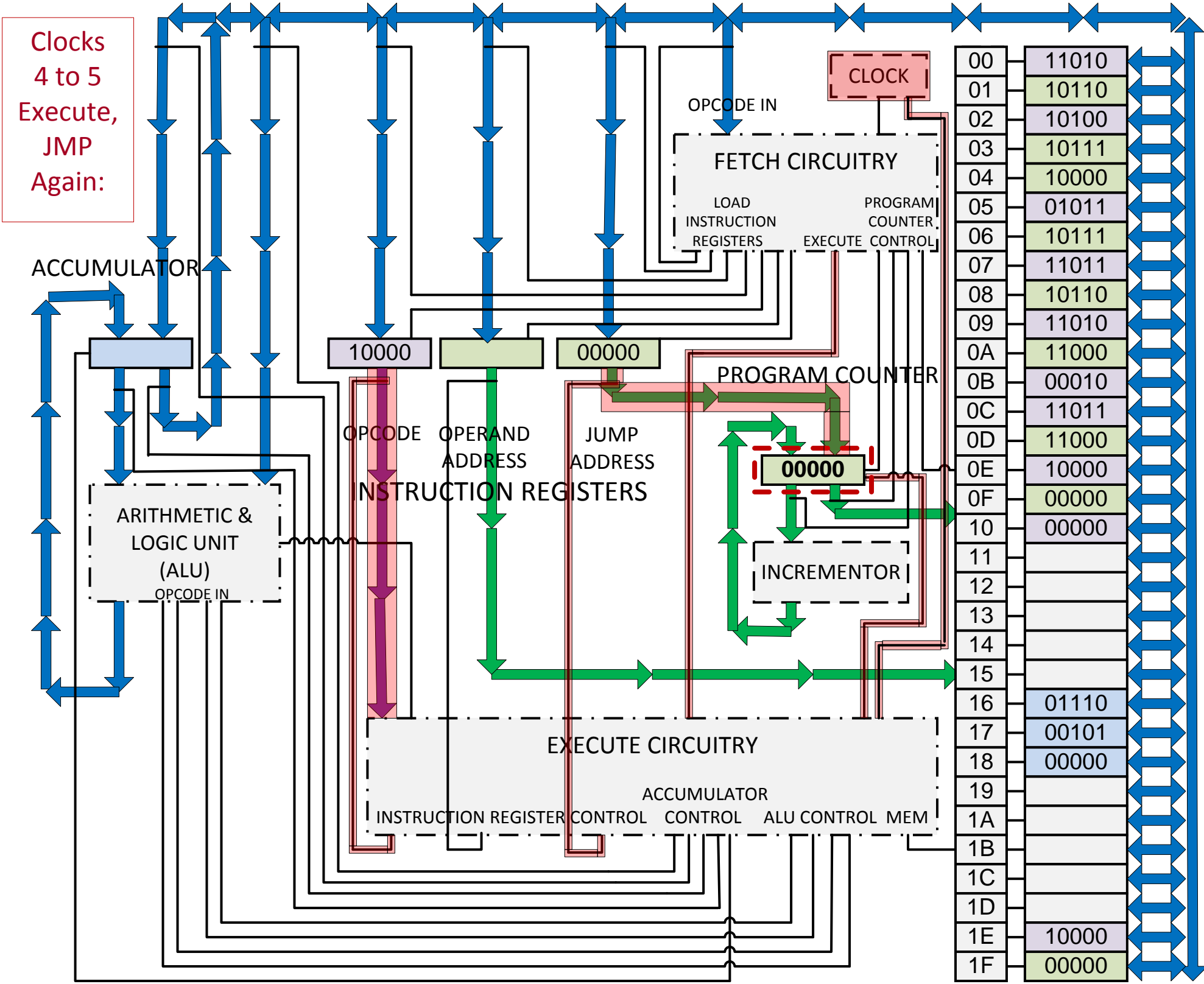
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then
# jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator
# and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

Hex

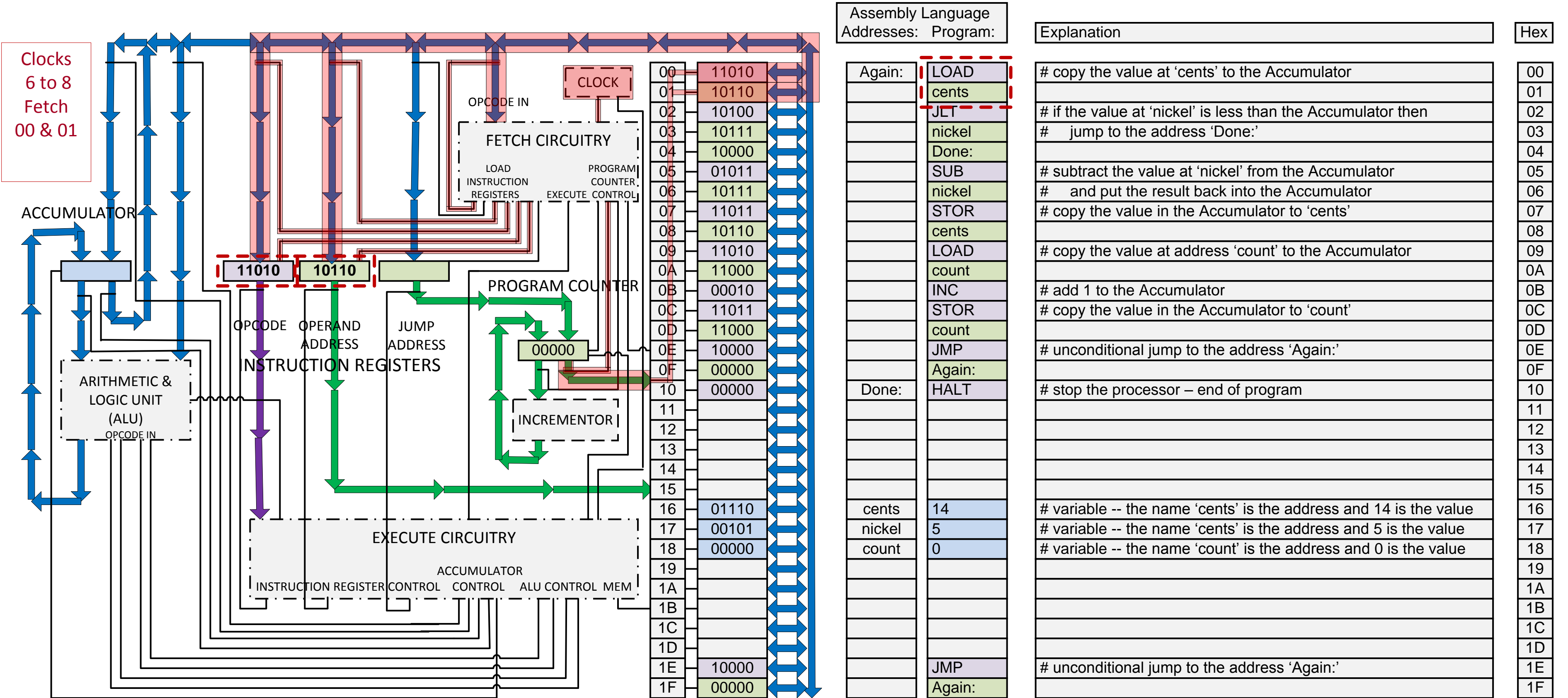
00
01
02
03
04
05
06
07
08
09
0A
0B
0C
0D
0E
0F
10
11
12
13
14
15
16
17
18
19
1A
1B
1C
1D
1E
1F







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



Clocks  
6 to 8  
Fetch  
00 & 01

Assembly Language  
Addresses: Program:

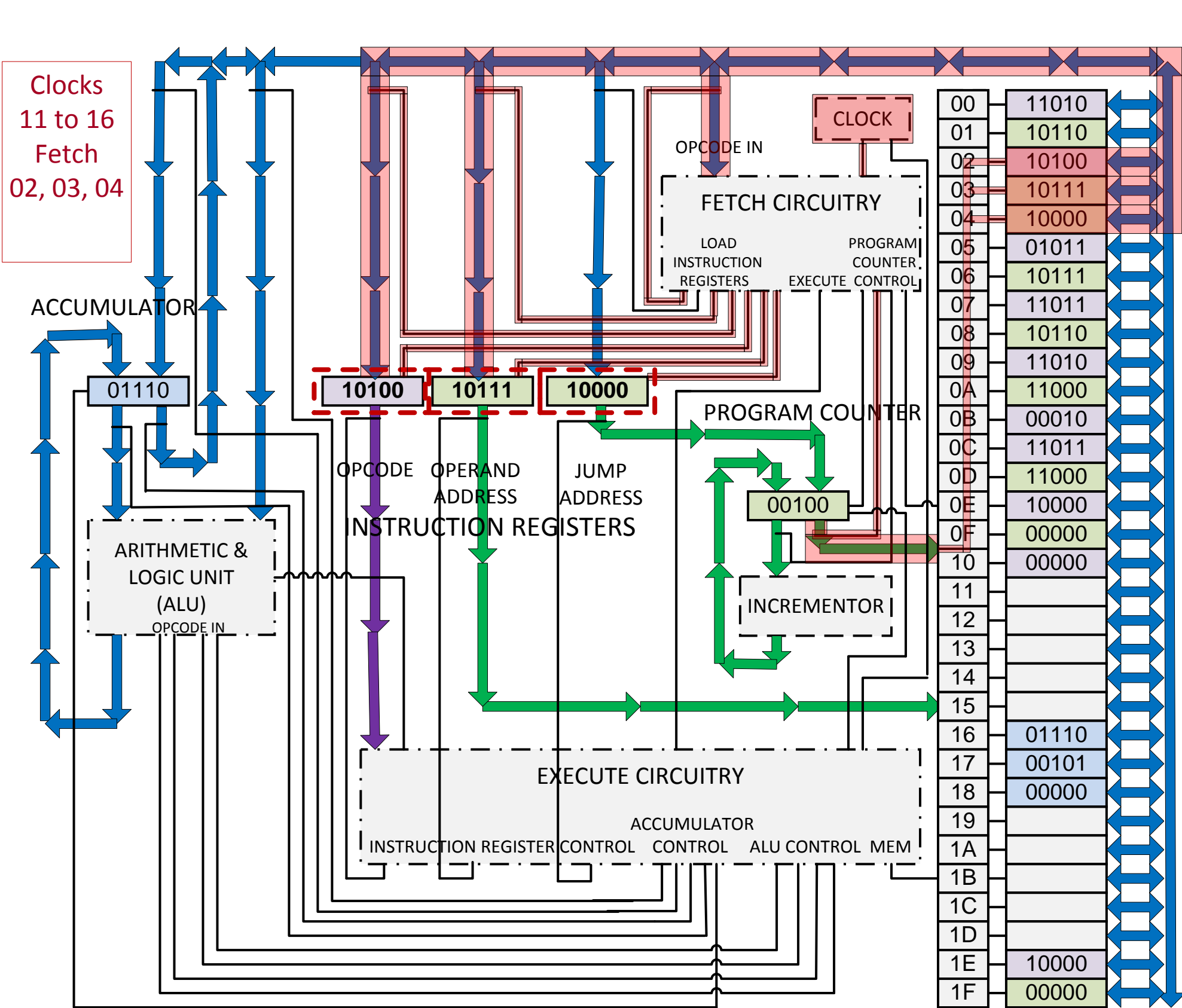
Explanation

Hex

Address	Hex	Binary	Instruction
00	00	11010	LOAD cents
01	01	10110	JLT nickel
02	02	10100	Done:
03	03	10111	nickel
04	04	10000	Done:
05	05	01011	SUB nickel
06	06	10111	nickel
07	07	11011	STOR cents
08	08	10110	cents
09	09	11010	LOAD count
0A	0A	11000	count
0B	0B	00010	INC
0C	0C	11011	STOR count
0D	0D	11000	count
0E	0E	10000	JMP Again:
0F	0F	00000	Again:
10	10	00000	Done:
11	11		
12	12		
13	13		
14	14		
15	15		
16	16	01110	cents 14
17	17	00101	nickel 5
18	18	00000	count 0
19	19		
1A	1A		
1B	1B		
1C	1C		
1D	1D		
1E	1E	10000	JMP Again:
1F	1F	00000	Again:

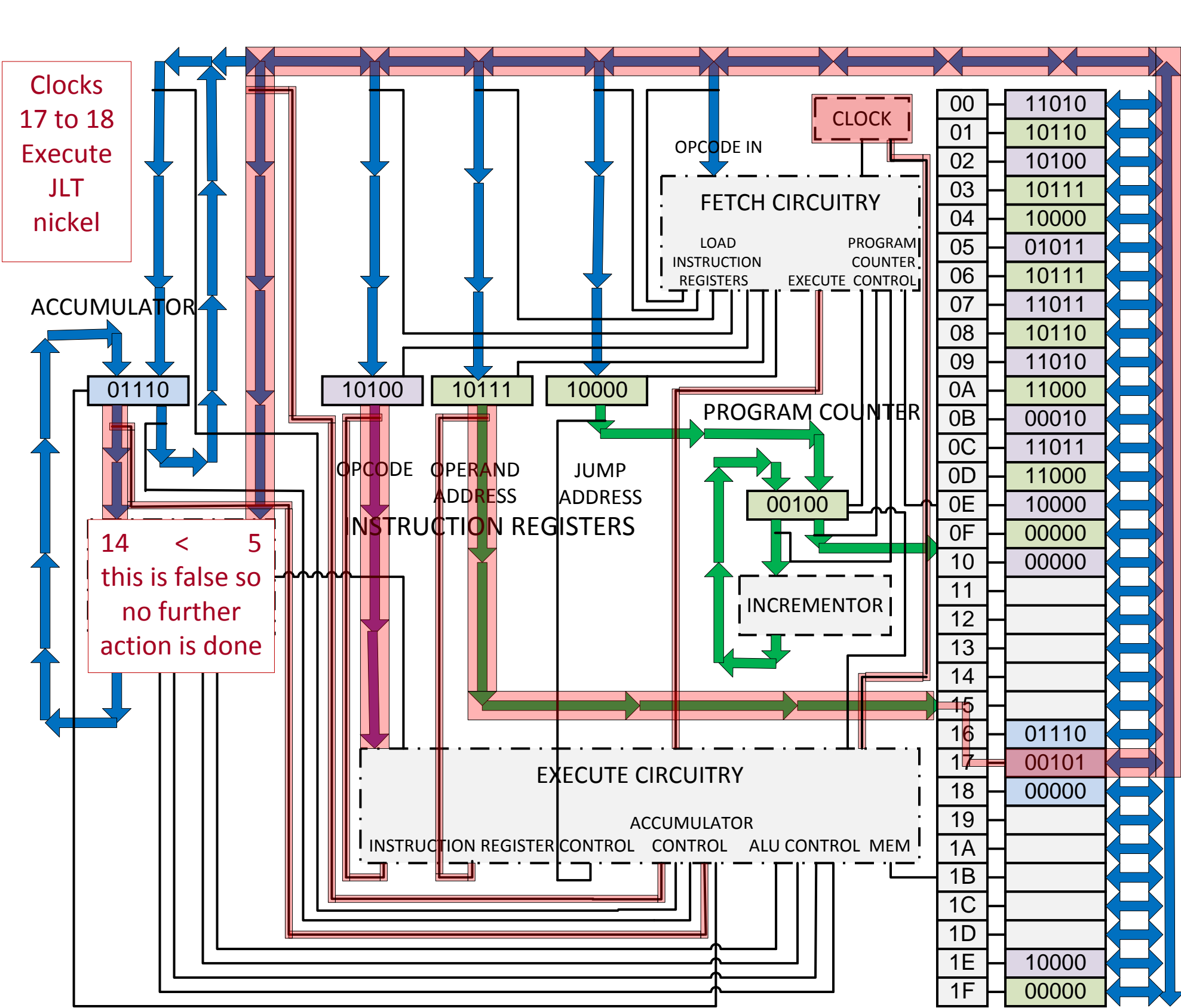
Address	Hex	Binary	Instruction	Explanation
00	00	11010	LOAD cents	# copy the value at 'cents' to the Accumulator
01	01	10110	JLT nickel	# if the value at 'nickel' is less than the Accumulator then
02	02	10100	Done:	# jump to the address 'Done:'
03	03	10111	nickel	
04	04	10000	Done:	
05	05	01011	SUB nickel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





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

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

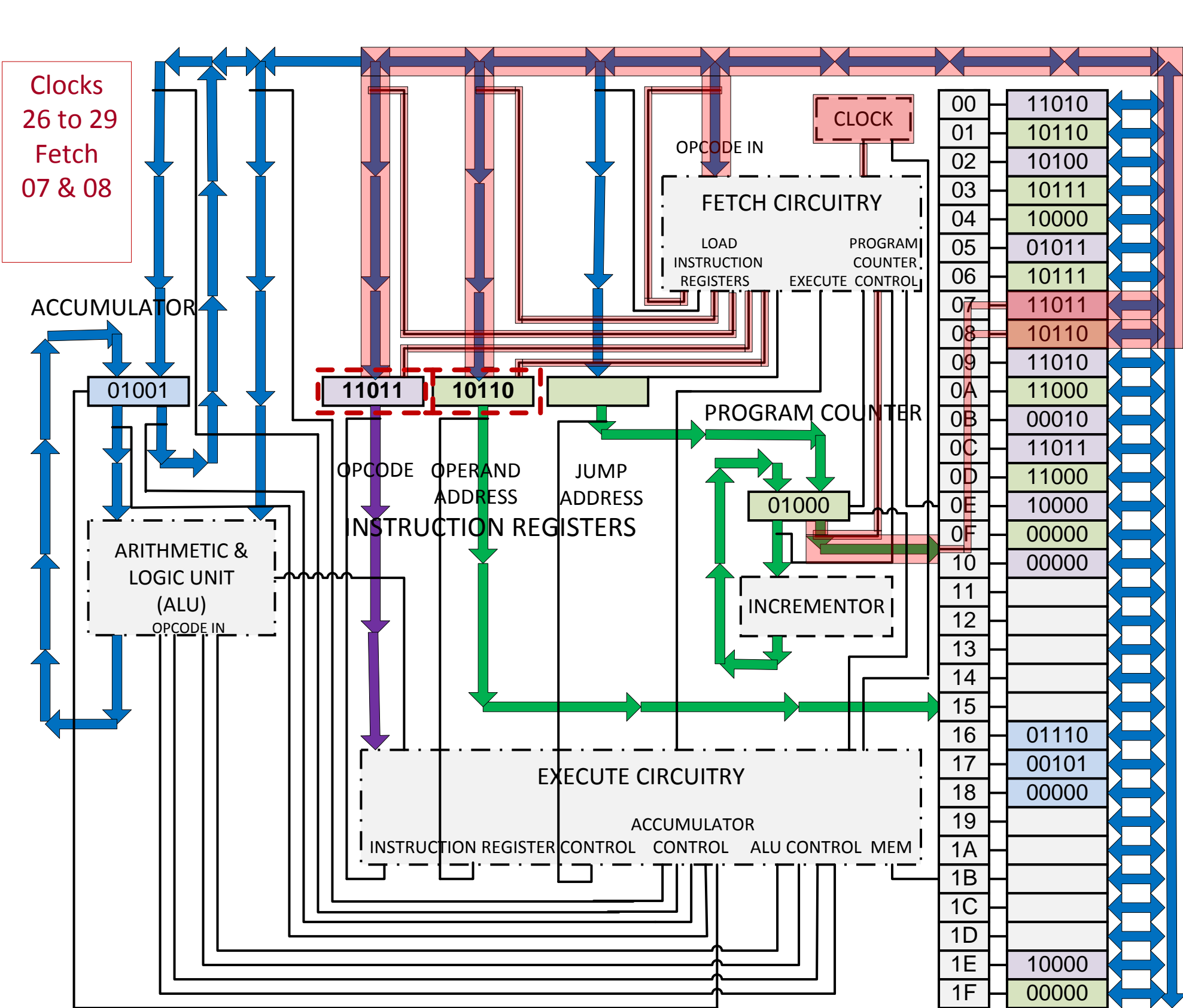


Assembly Language Addresses: Program:		Explanation	Hex
Again:	LOAD	# copy the value at 'cents' to the Accumulator	00
	<del>cents</del>		01
	<del>JLT</del>	# if the value at 'nickel' is less than the Accumulator then	02
	<del>nickel</del>	# jump to the address 'Done:'	03
	<del>Done:</del>		04
	<del>SUB</del>	# 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



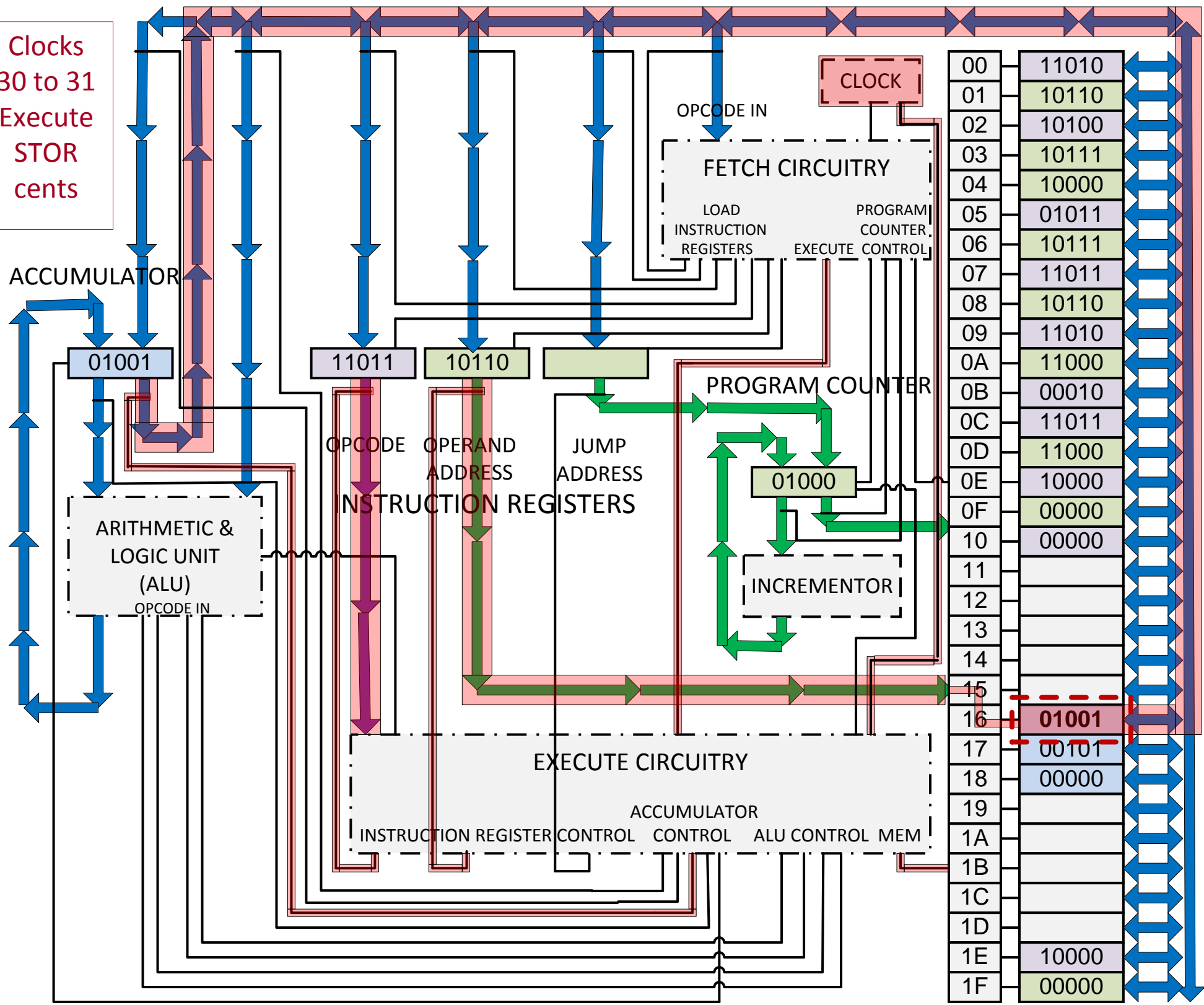






Assembly Language Addresses: Program:		Explanation	Hex
Again:	LOAD	# copy the value at 'cents' to the Accumulator	00
	cents		01
	JLT	# if the value at 'nickel' is less than the Accumulator then	02
	nickel	# jump to the address 'Done:'	03
	Done:		04
	SUB	# subtract the value at 'nickel' from the Accumulator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

Clocks 30 to 31  
Execute  
STOR  
cents



Assembly Language Addresses: Program:

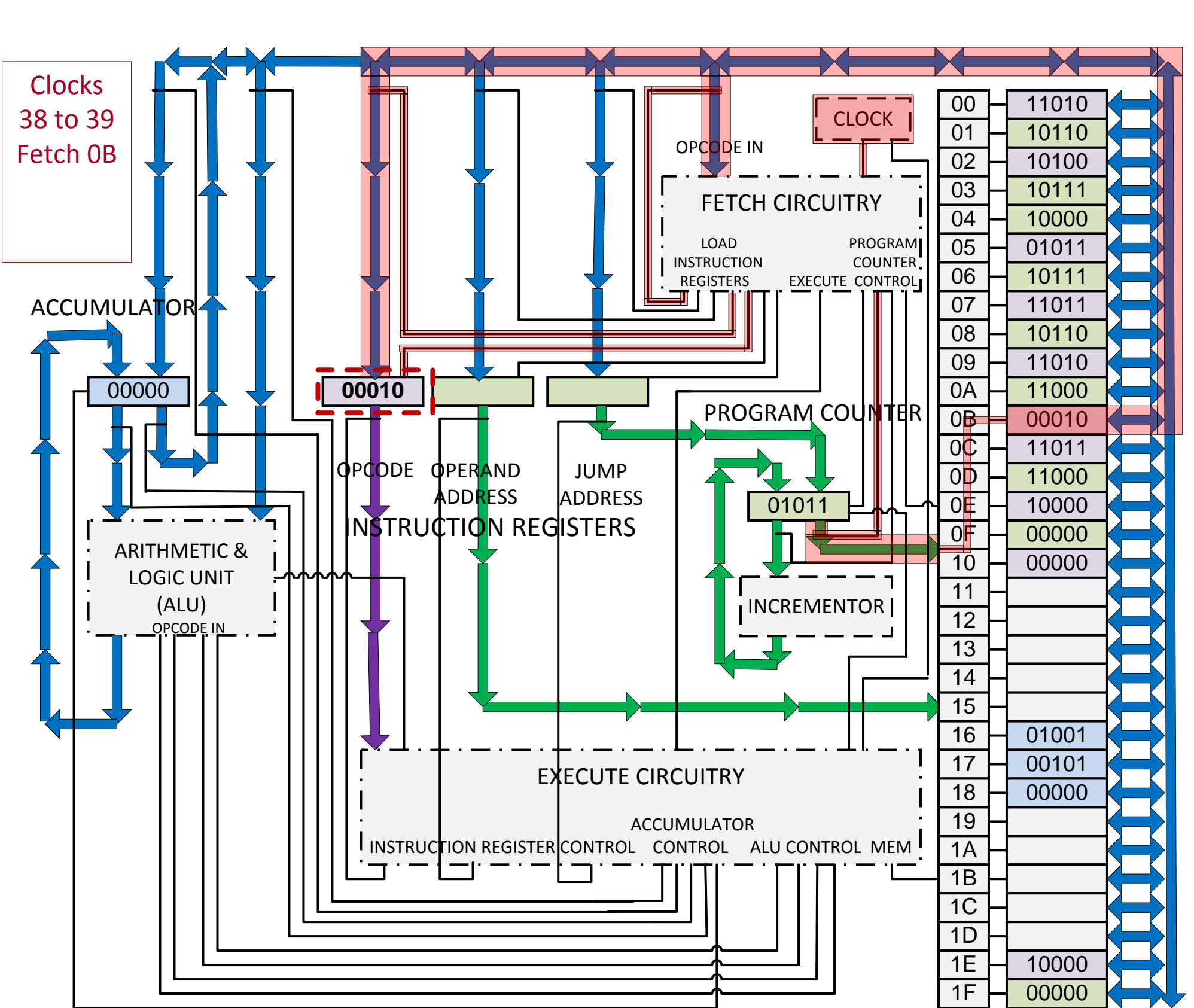
Address	Instruction	Explanation	Hex
00	LOAD	# copy the value at 'cents' to the Accumulator	00
01	cents		01
02	JLT	# if the value at 'nickel' is less than the Accumulator then	02
03	nickel	# jump to the address 'Done:'	03
04	Done:		04
05	SUB	# subtract the value at 'nickel' from the Accumulator	05
06	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	HALT	# stop the processor – end of program	10
11			11
12			12
13			13
14			14
15			15
16	01001	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	# variable -- the name 'count' is the address and 0 is the value	18
19			19
1A			1A
1B			1B
1C			1C
1D			1D
1E	10000	# unconditional jump to the address 'Again:'	1E
1F	00000		1F

Label	Address	Value
cents	9	14
nickel	5	5
count	0	0







Assembly Language Addresses: Program:

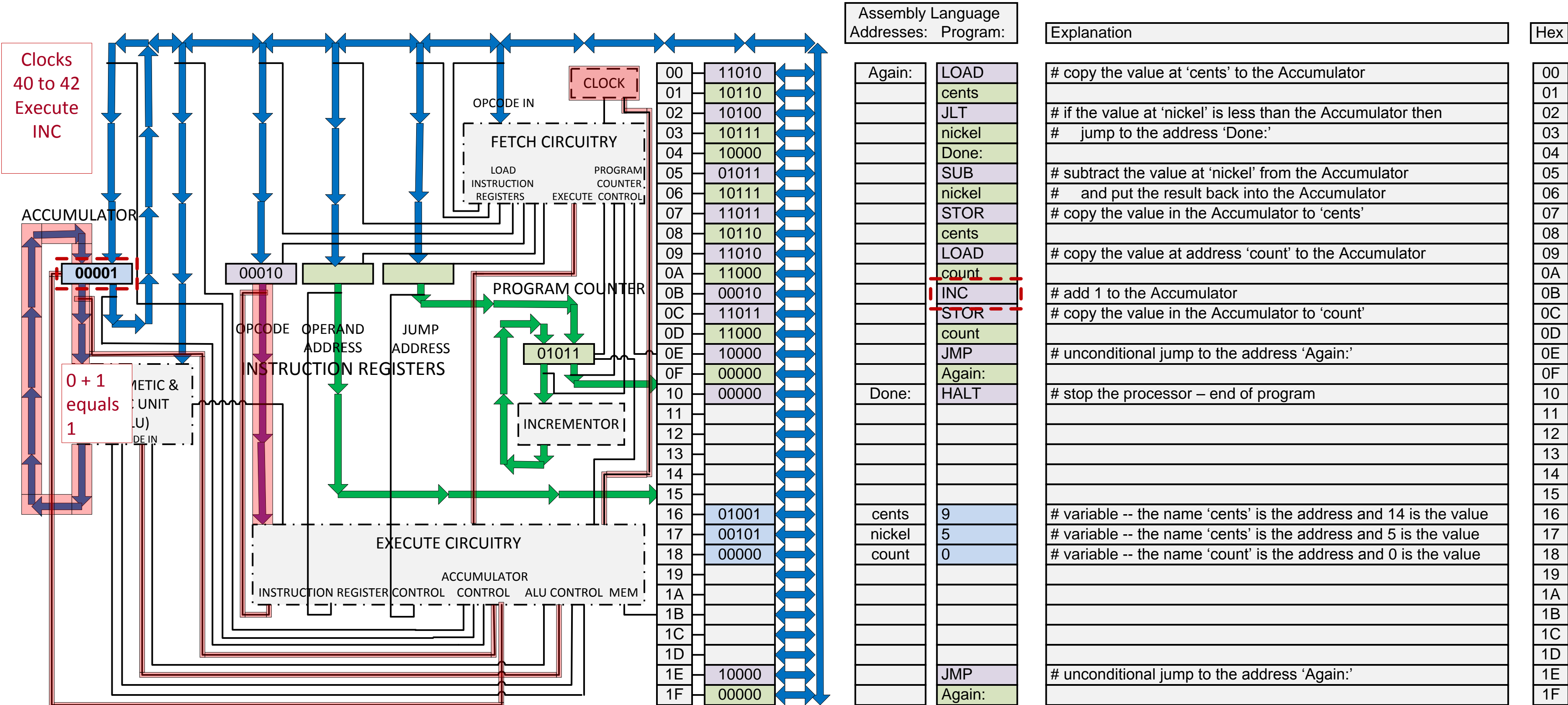
00	LOAD
01	cents
02	JLT
03	nickel
04	Done:
05	SUB
06	nickel
07	STOR
08	cents
09	LOAD
0A	count
0B	INC
0C	STOR
0D	count
0E	JMP
0F	Again:
10	HALT
11	
12	
13	
14	
15	
16	cents 9
17	nickel 5
18	count 0
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

Explanation

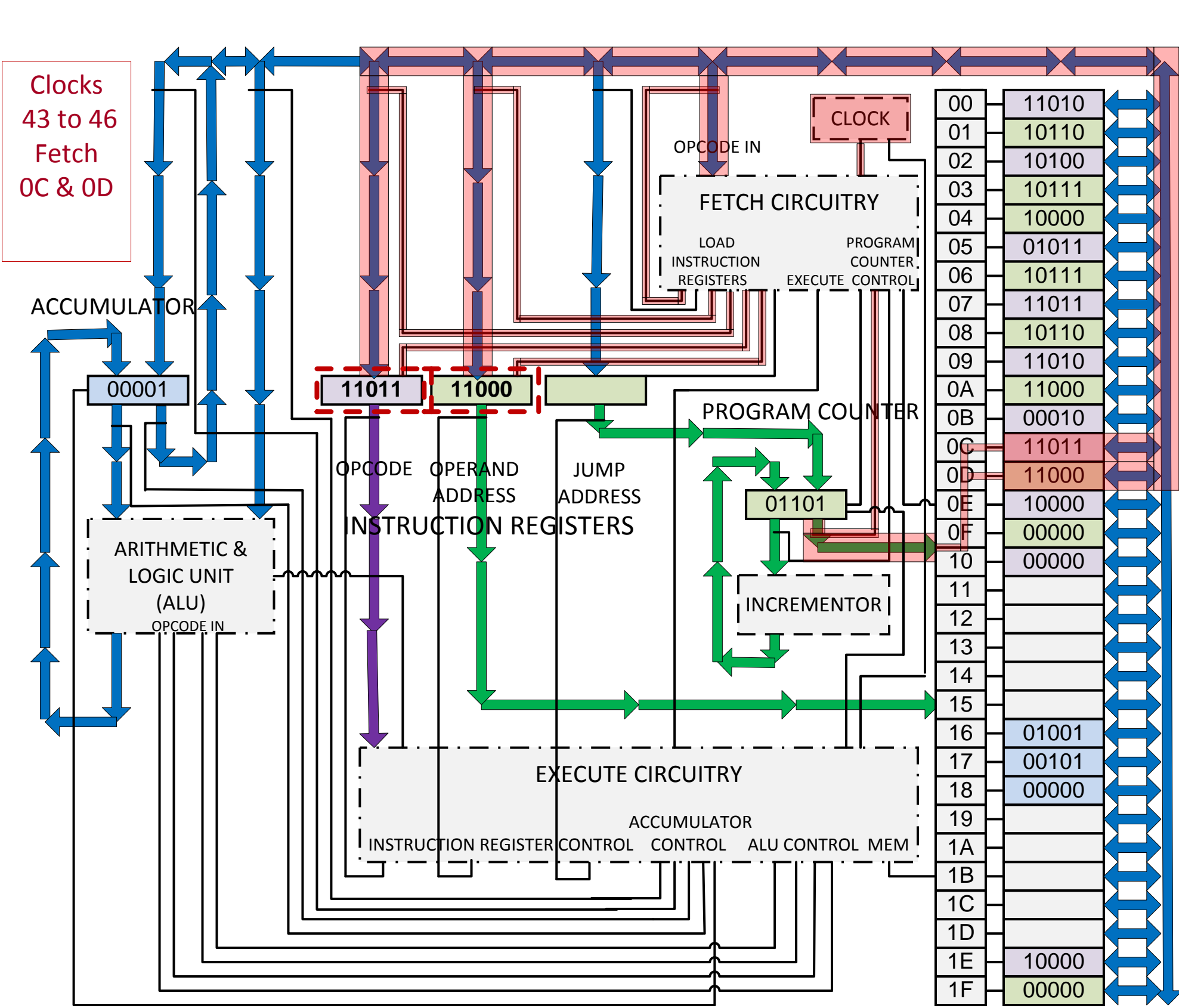
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then # jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator # and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

Hex

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







Assembly Language  
Addresses: Program:

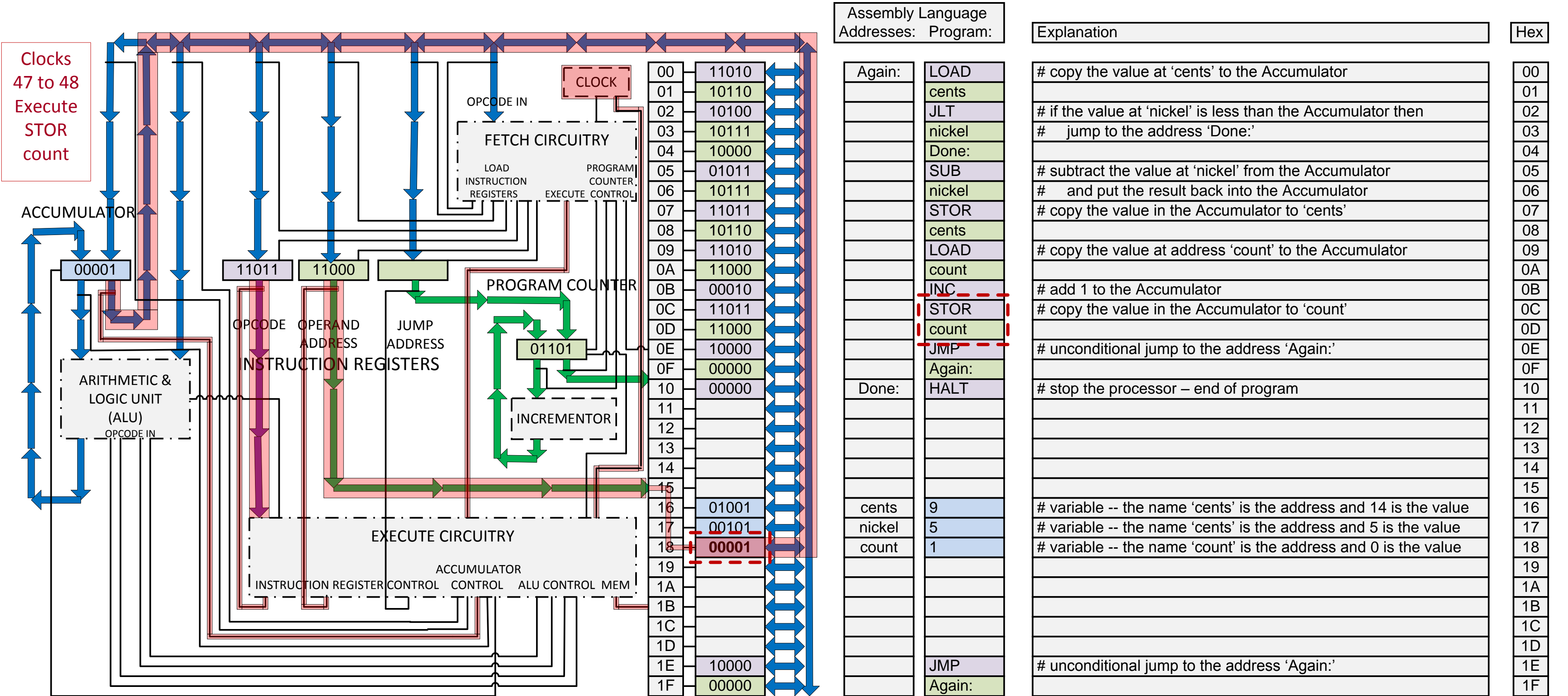
00	LOAD
01	cents
02	JLT
03	nickel
04	Done:
05	SUB
06	nickel
07	STOR
08	cents
09	LOAD
0A	count
0B	INC
0C	STOR
0D	count
0E	JMP
0F	Again:
10	HALT
11	
12	
13	
14	
15	
16	cents 9
17	nickel 5
18	count 0
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

Explanation

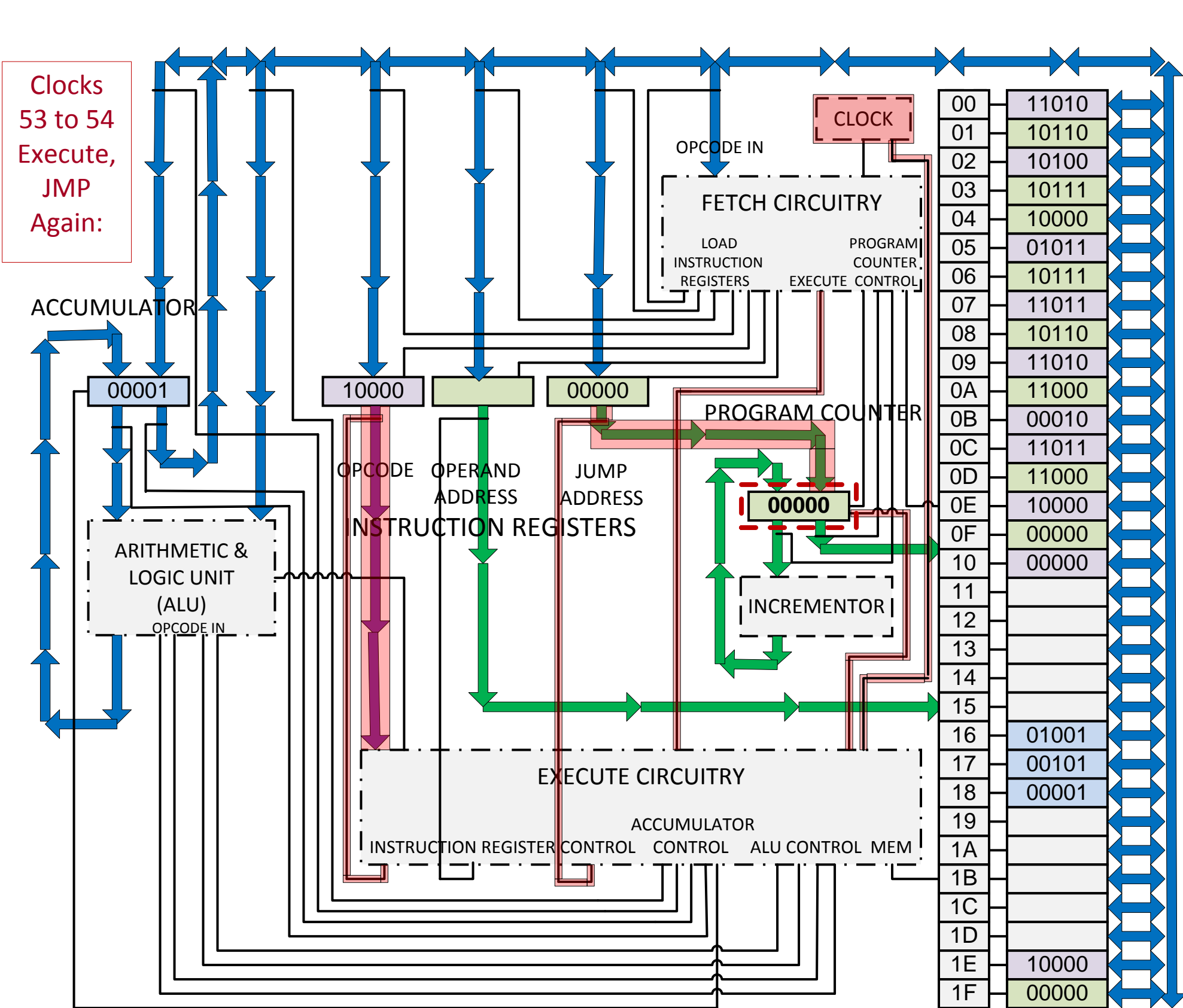
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then
# jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator
# and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

Hex

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







Assembly Language  
Addresses: Program:

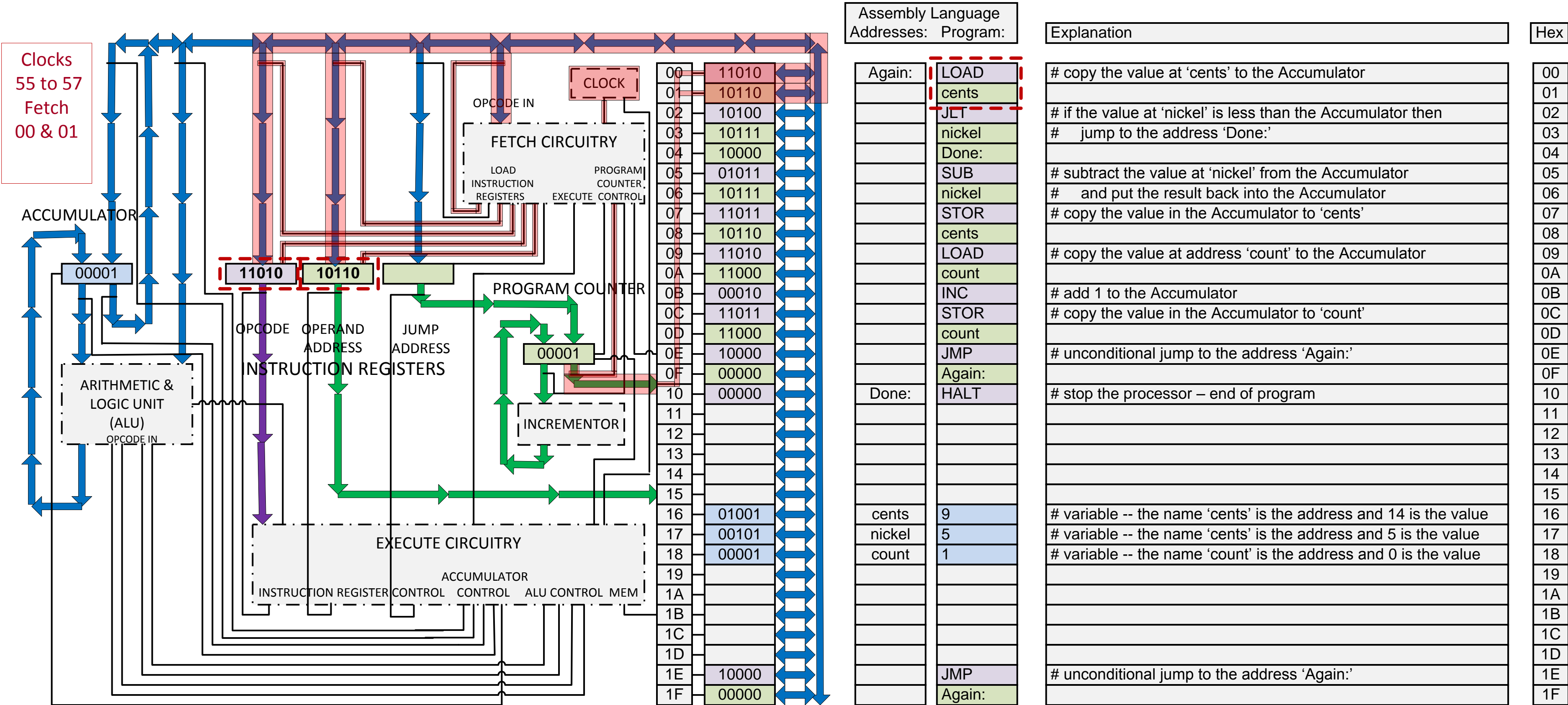
Address	Instruction
00	LOAD
01	cents
02	JLT
03	nickel
04	Done:
05	SUB
06	nickel
07	STOR
08	cents
09	LOAD
0A	count
0B	INC
0C	STOR
0D	count
0E	JMP
0F	Again:
10	HALT
11	
12	
13	
14	
15	
16	cents 9
17	nickel 5
18	count 1
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

Explanation

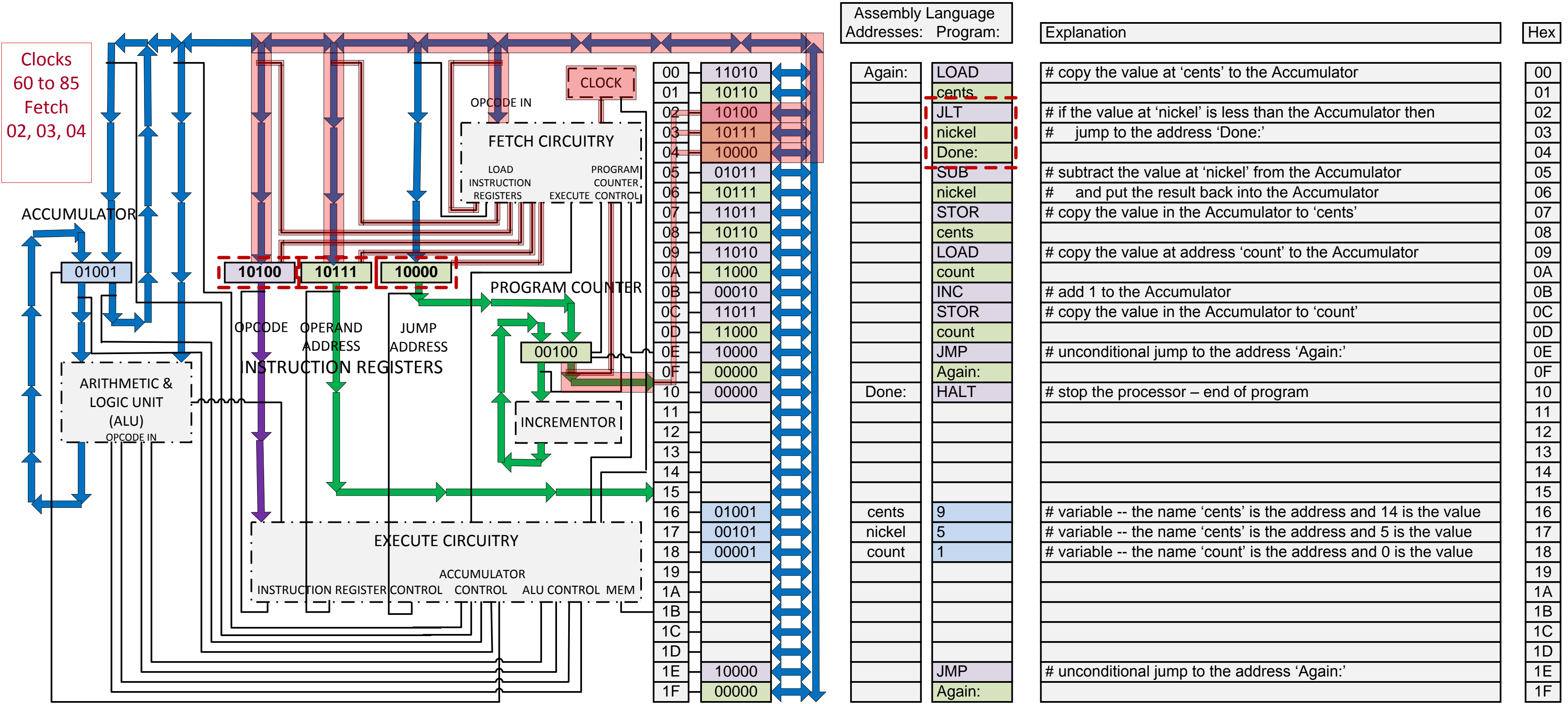
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then
# jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator
# and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

Hex

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

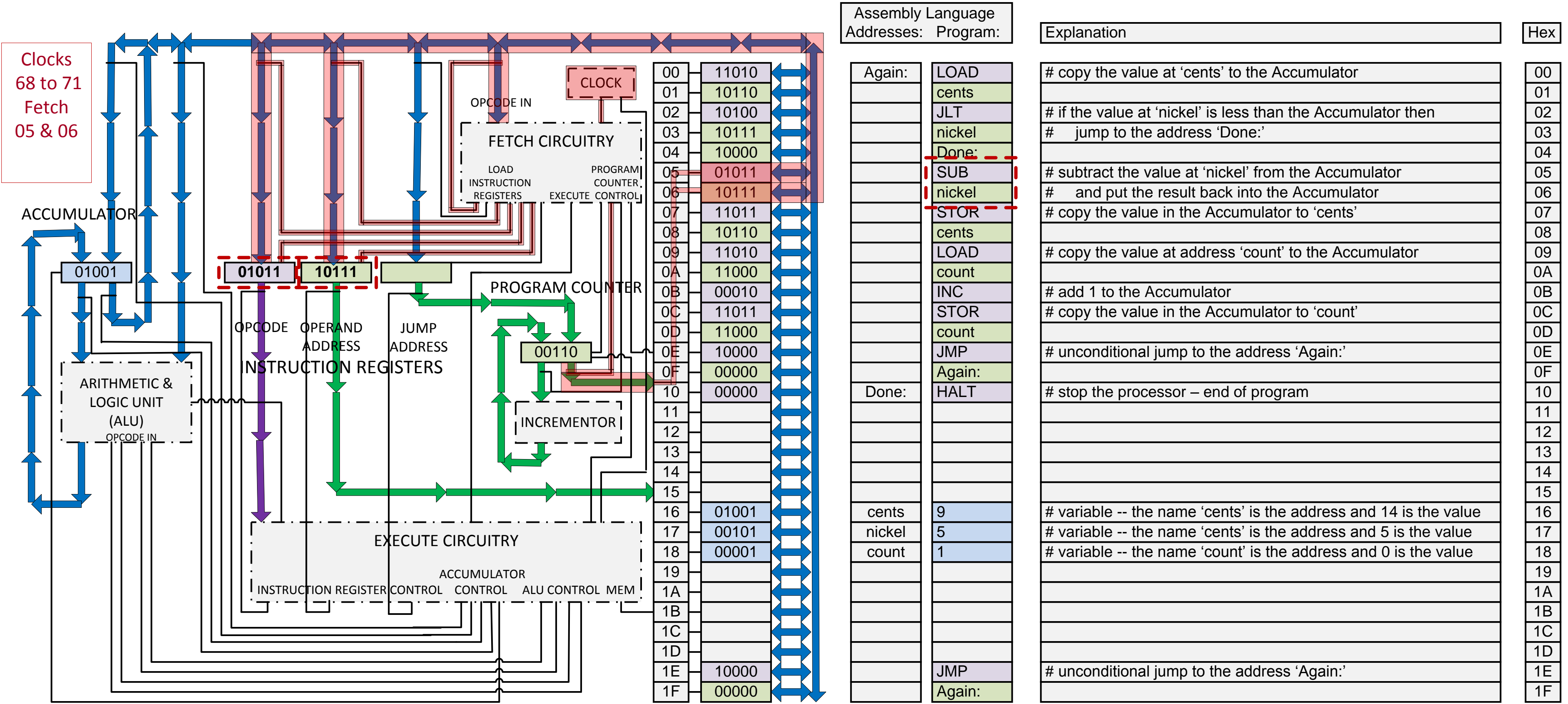




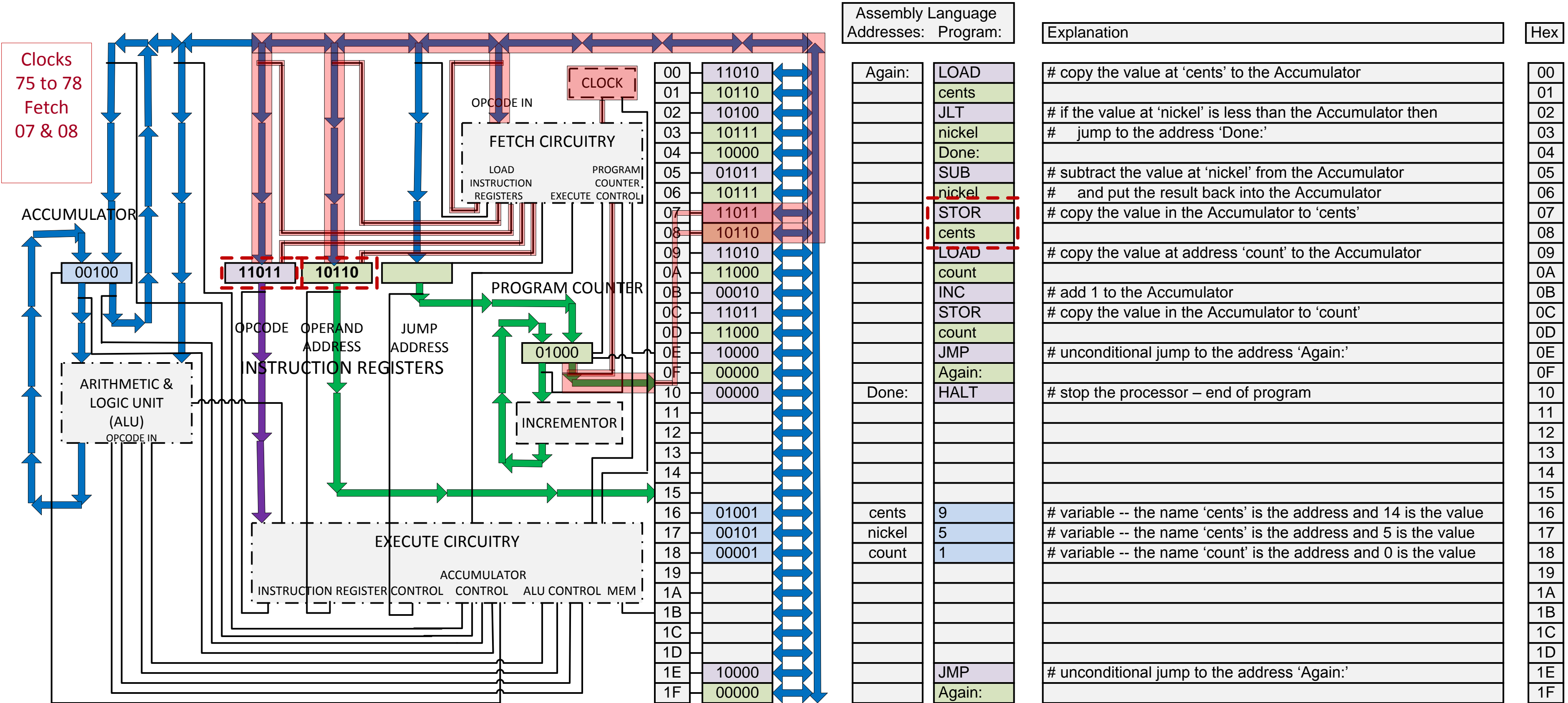












Clocks 75 to 78 Fetch 07 & 08

Assembly Language Addresses: Program:

Explanation

Hex

ACCUMULATOR

00100

11011

10110

PROGRAM COUNTER

01000

ARITHMETIC & LOGIC UNIT (ALU)

INCREMENTOR

INSTRUCTION REGISTERS

EXECUTE CIRCUITRY

ACCUMULATOR

INSTRUCTION REGISTER CONTROL CONTROL ALU CONTROL MEM

FETCH CIRCUITRY

PROGRAM COUNTER, EXECUTE CONTROL

CLOCK

Again:

LOAD

cents

JLT

nickel

Done:

SUB

nickel

STOR

cents

LOAD

count

INC

STOR

count

JMP

Again:

Done:

HALT

cents

9

nickel

5

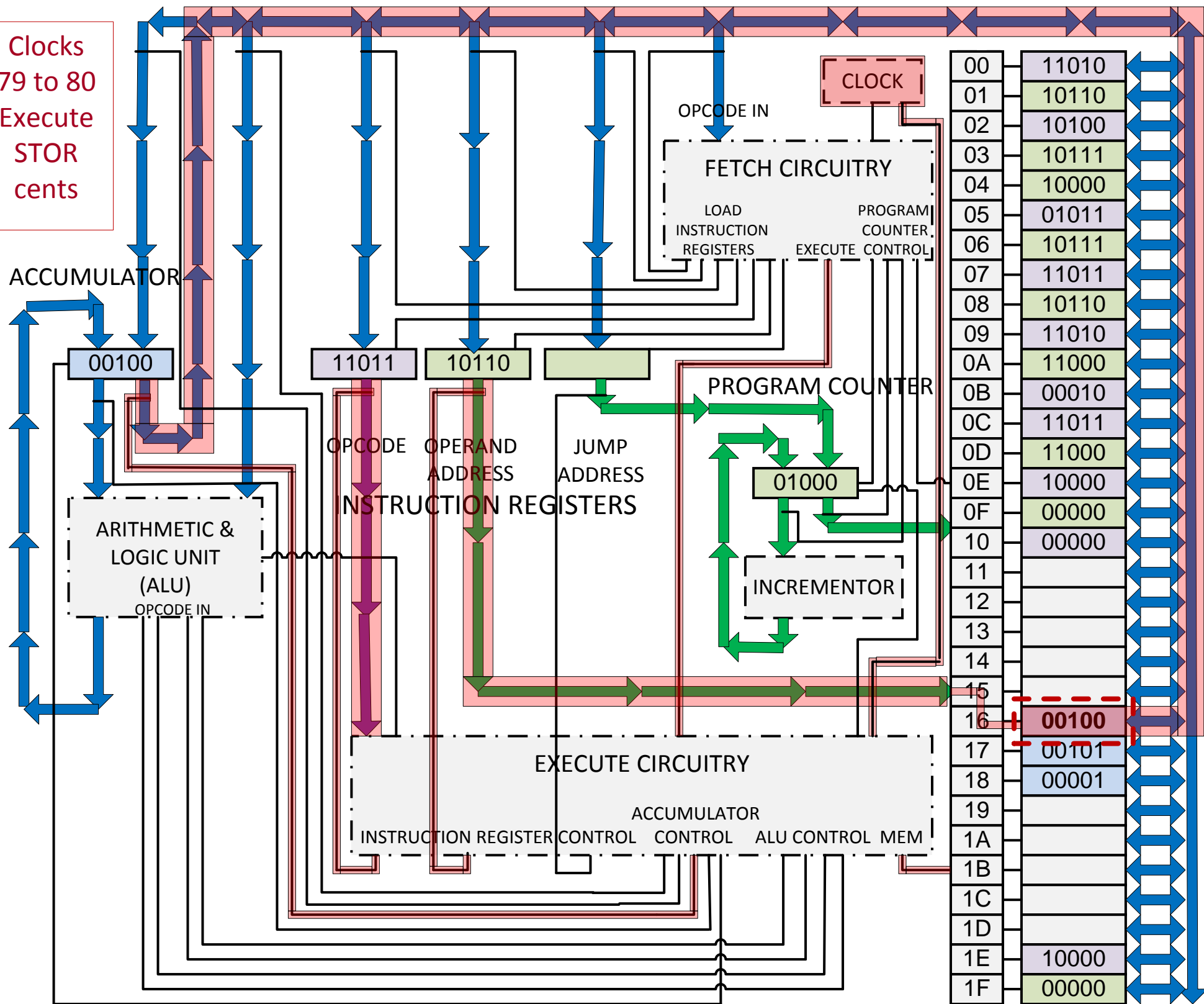
count

1

JMP

Again:

Clocks 79 to 80  
Execute  
STOR  
cents



Assembly Language  
Addresses: Program:

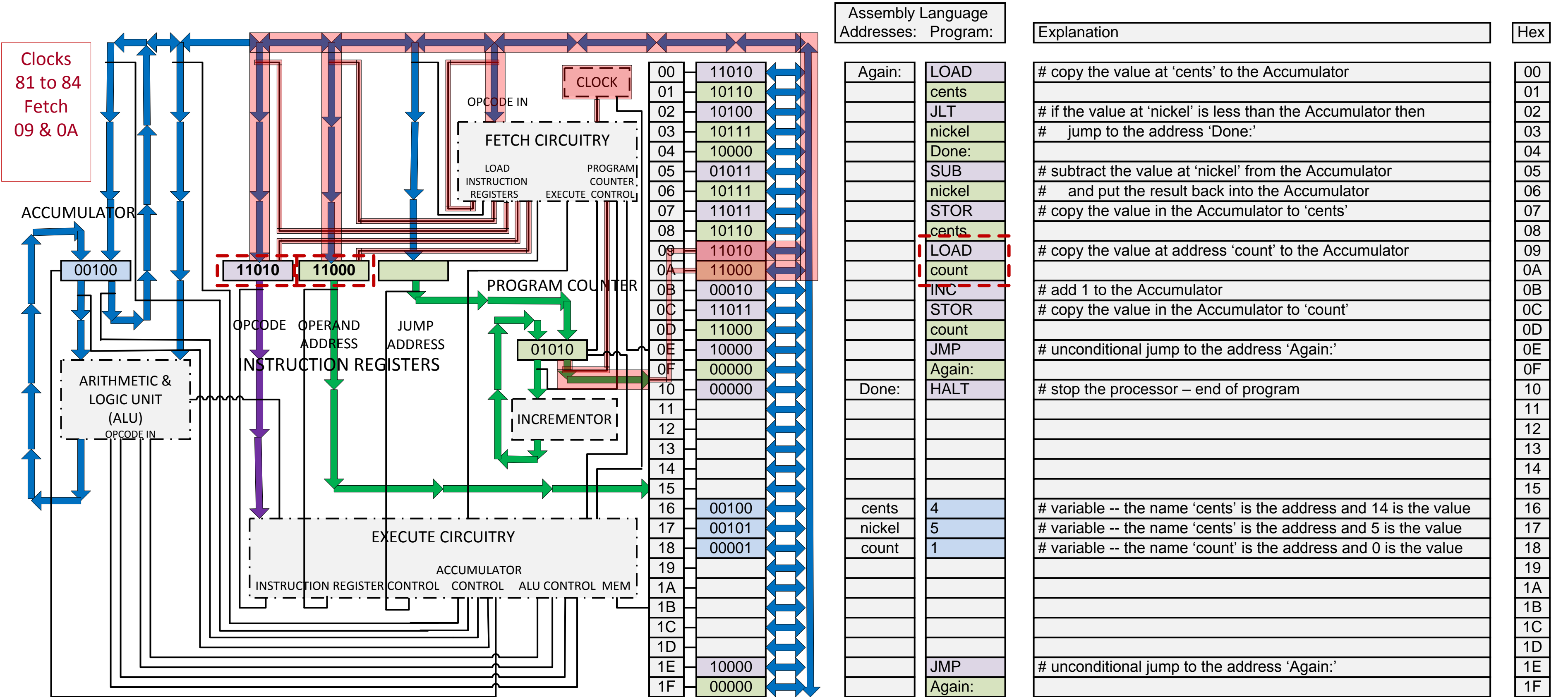
00	11010	Again:	LOAD
01	10110		cents
02	10100		JLT
03	10111		nickel
04	10000		Done:
05	01011		SUB
06	10111		nickel
07	11011		STOR
08	10110		cents
09	11010		LOAD
0A	11000		count
0B	00010		INC
0C	11011		STOR
0D	11000		count
0E	10000		JMP
0F	00000		Again:
10	00000	Done:	HALT
11			
12			
13			
14			
15			
16	00100	cents	4
17	00101	nickel	5
18	00001	count	1
19			
1A			
1B			
1C			
1D			
1E	10000		JMP
1F	00000		Again:

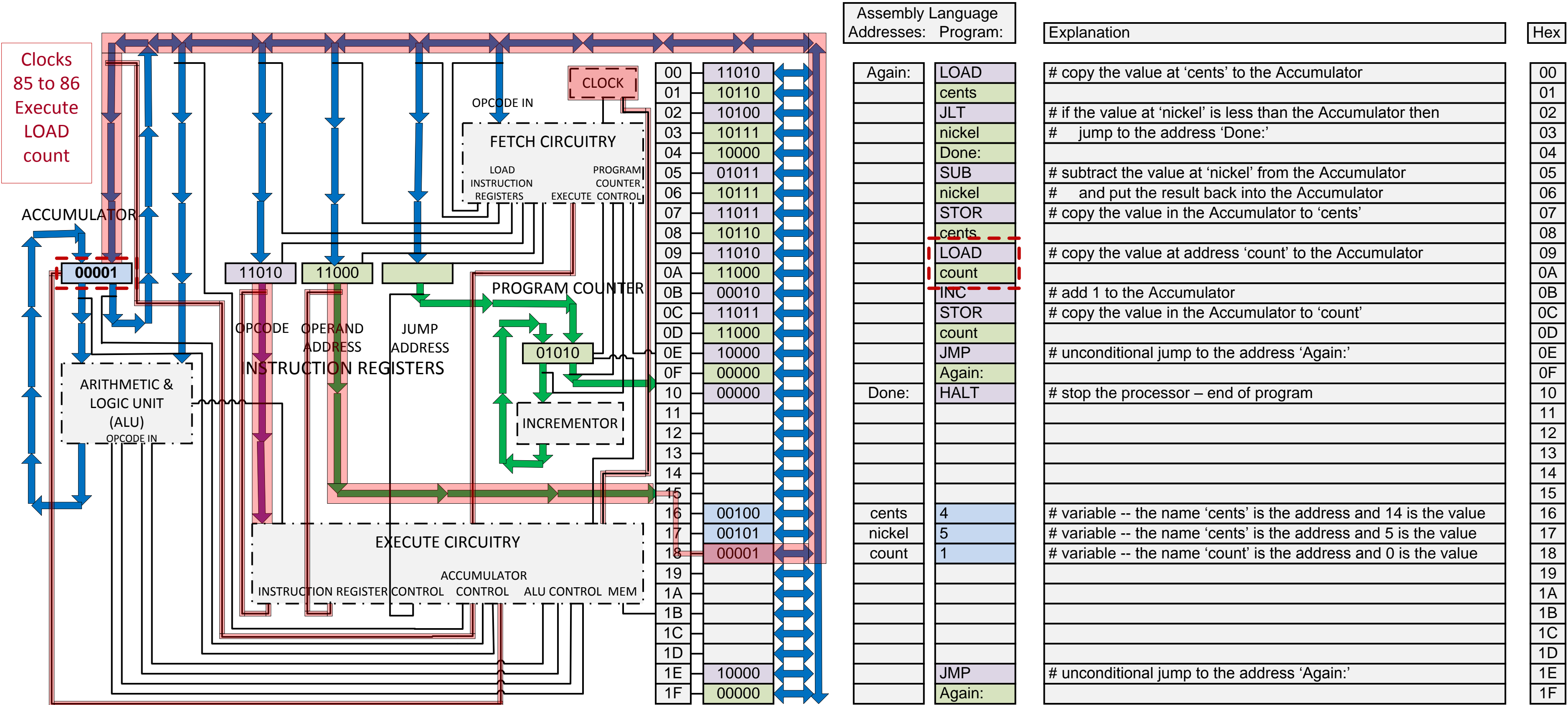
Explanation

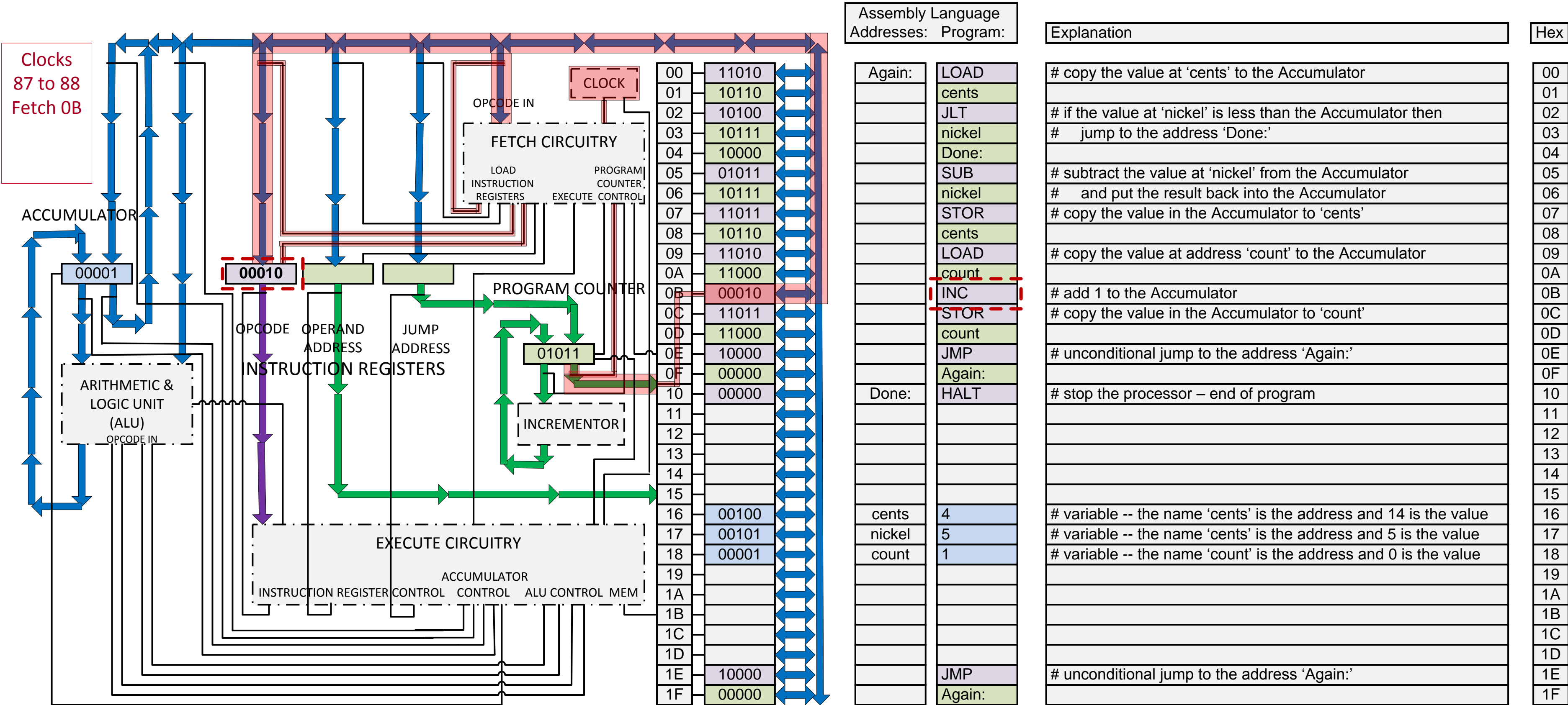
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then # jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator # and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

Hex

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

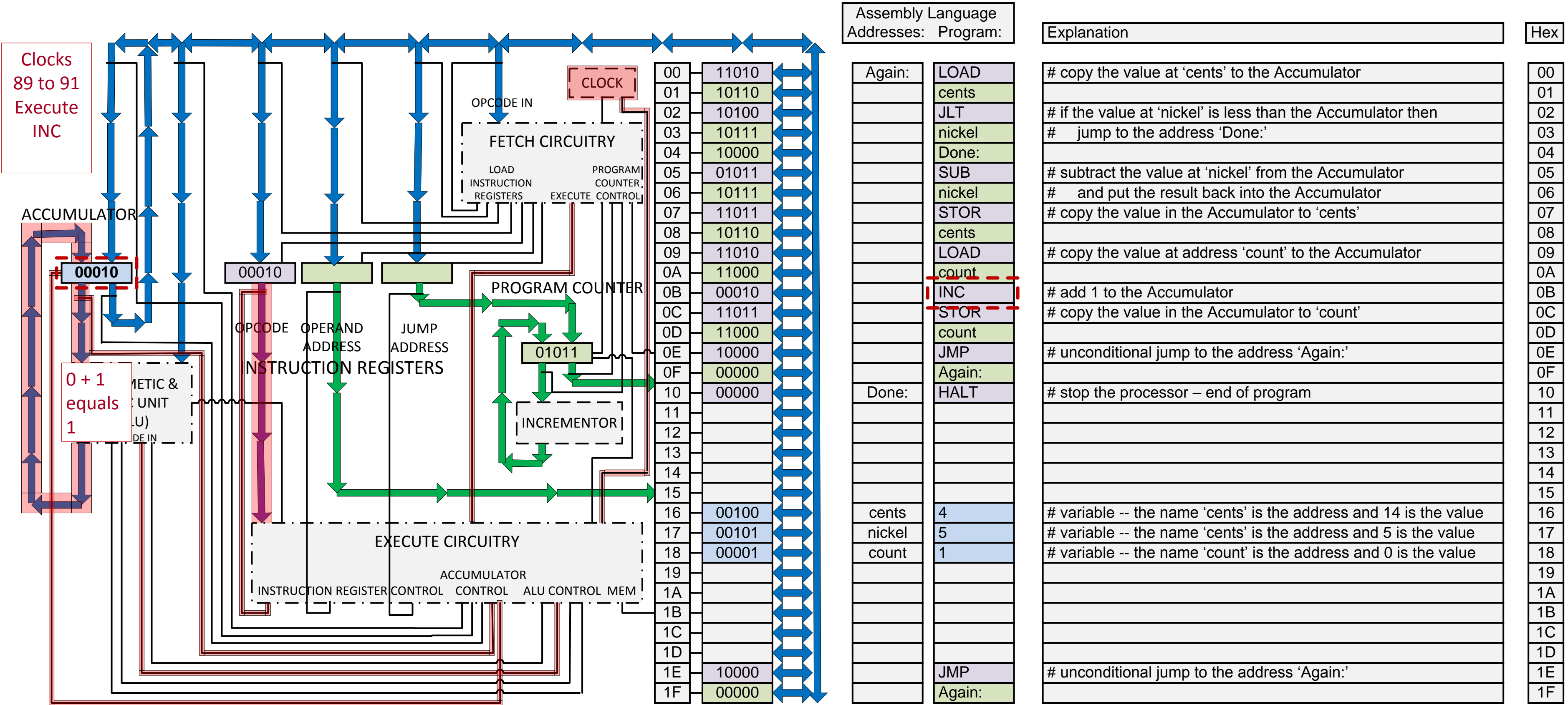




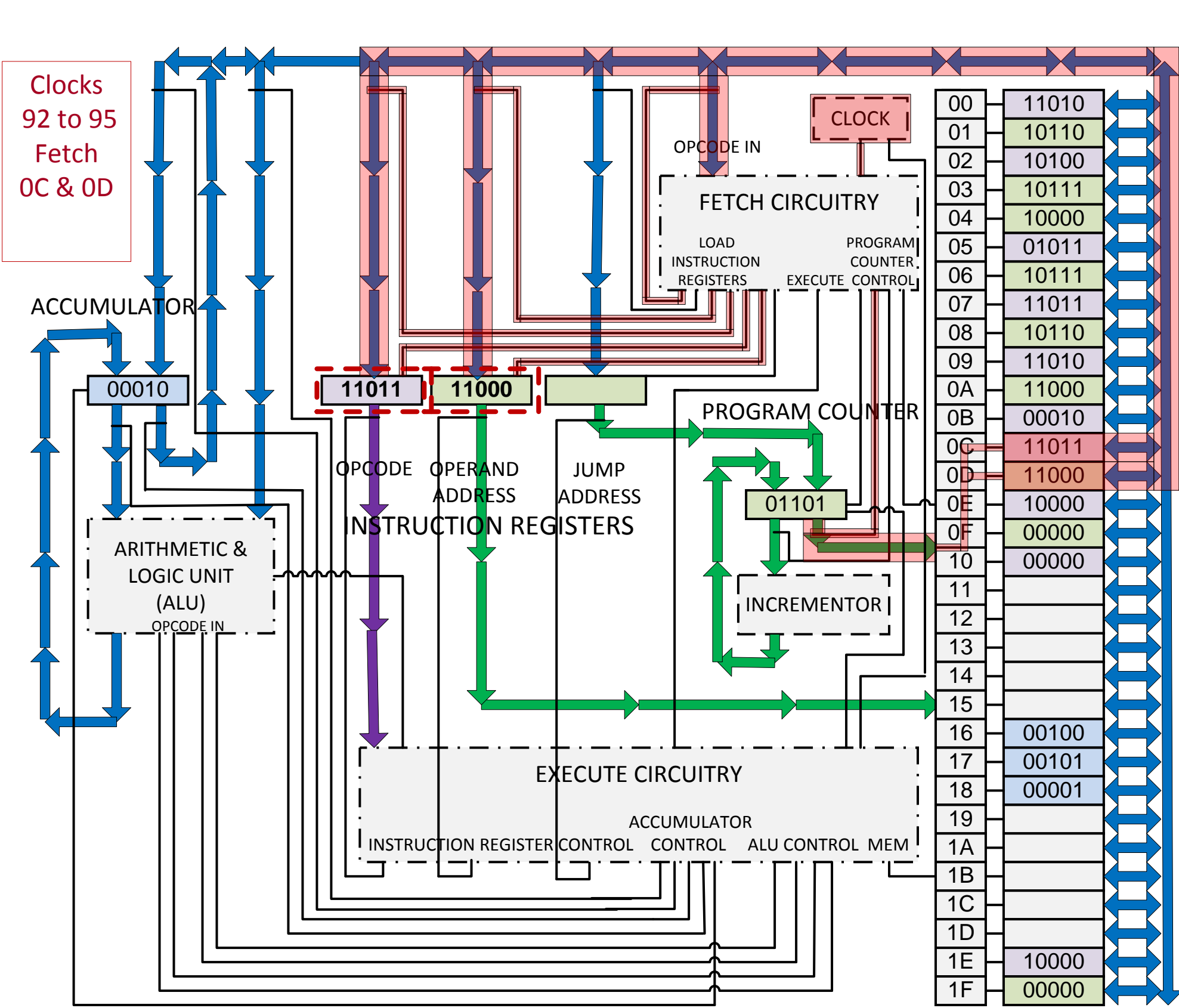


Assembly Language Addresses: Program:

Address	Instruction	Explanation	Hex
00	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	HALT	# stop the processor – end of program	10
11			11
12			12
13			13
14			14
15			15
16	cents	# variable -- the name 'cents' is the address and 14 is the value	16
17	nickel	# variable -- the name 'cents' is the address and 5 is the value	17
18	count	# 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







Assembly Language Addresses: Program:

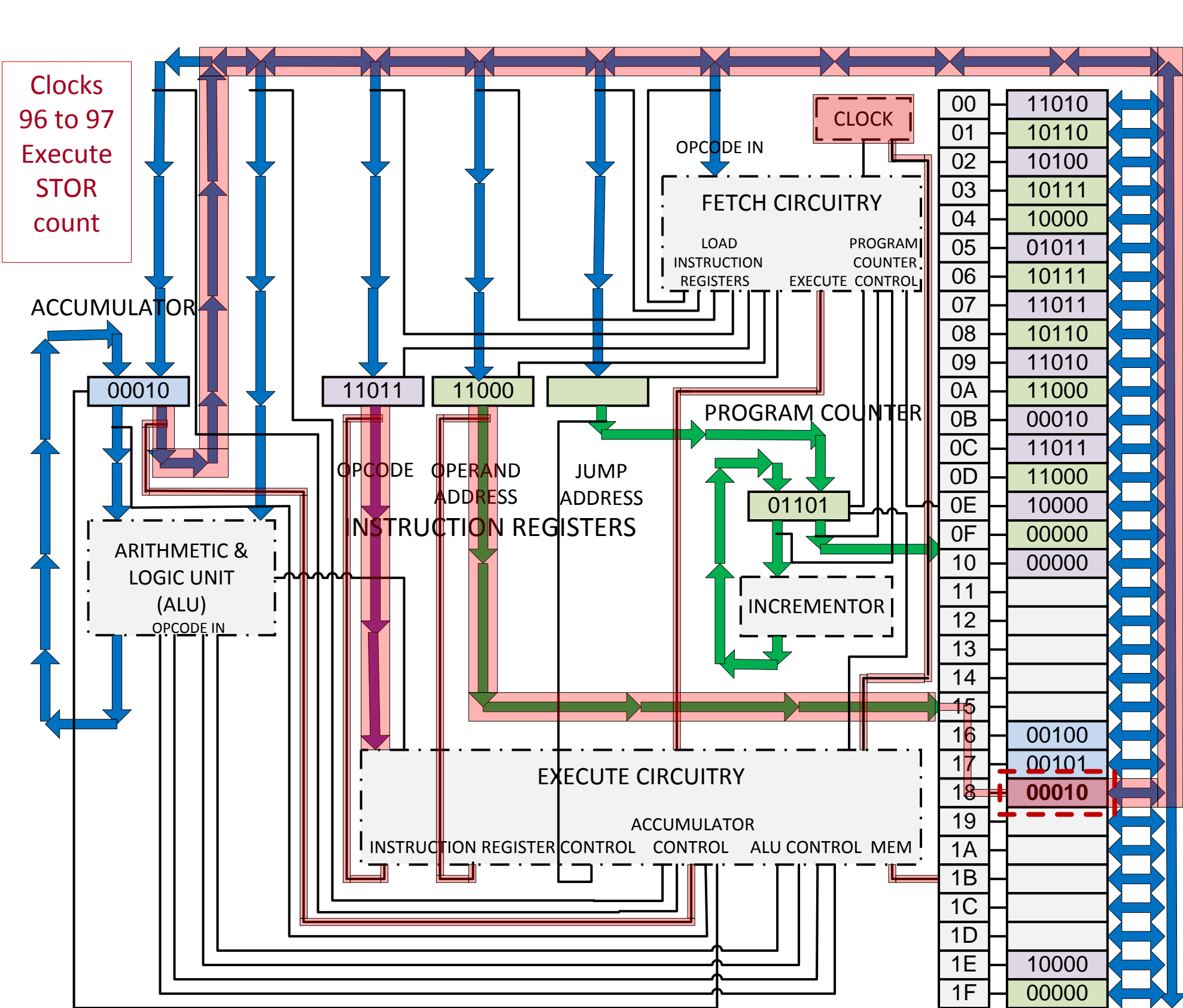
00	LOAD
01	cents
02	JLT
03	nickel
04	Done:
05	SUB
06	nickel
07	STOR
08	cents
09	LOAD
0A	count
0B	INC
0C	STOR
0D	count
0E	JMP
0F	Again:
10	HALT
11	
12	
13	
14	
15	
16	cents 4
17	nickel 5
18	count 1
19	
1A	
1B	
1C	
1D	
1E	JMP
1F	Again:

Explanation

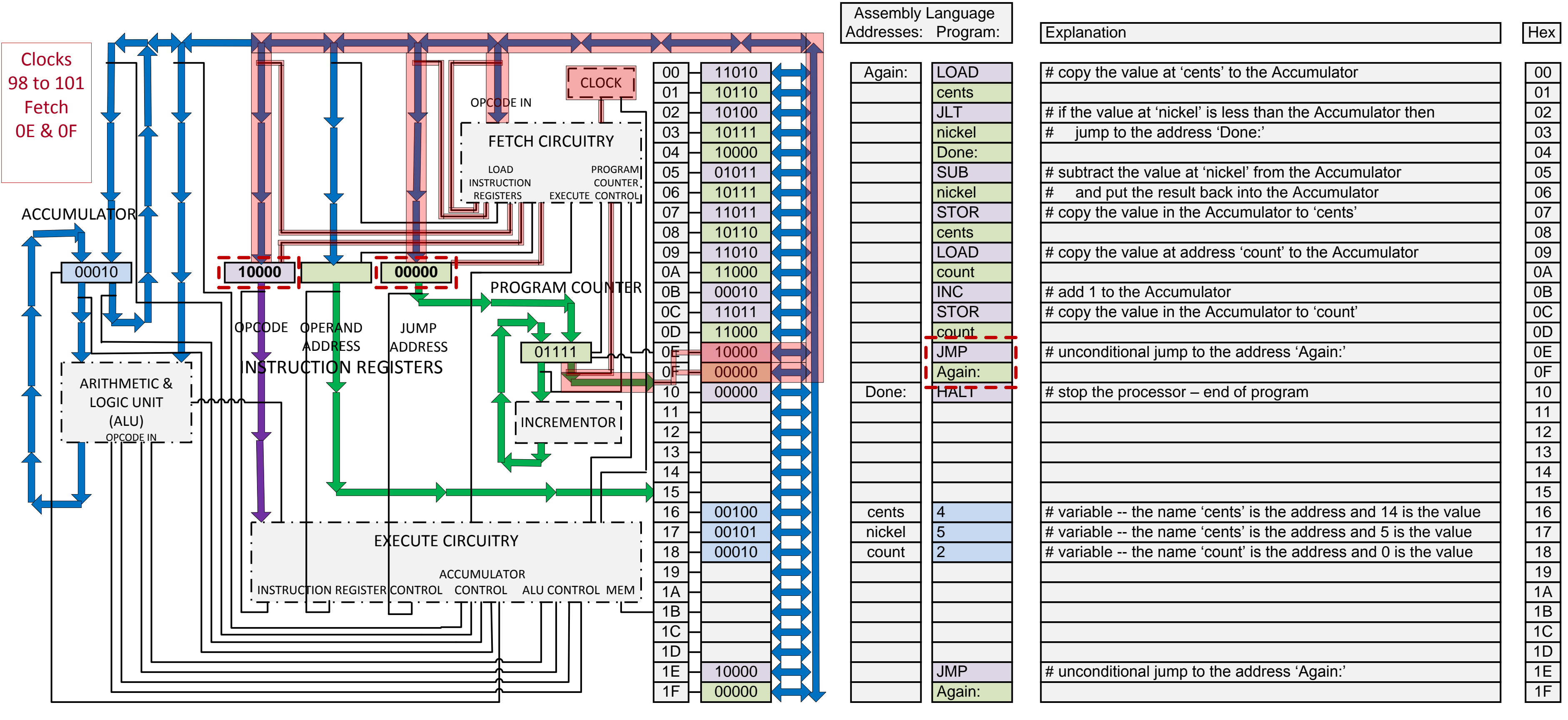
# copy the value at 'cents' to the Accumulator
# if the value at 'nickel' is less than the Accumulator then
# jump to the address 'Done:'
# subtract the value at 'nickel' from the Accumulator
# and put the result back into the Accumulator
# copy the value in the Accumulator to 'cents'
# copy the value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

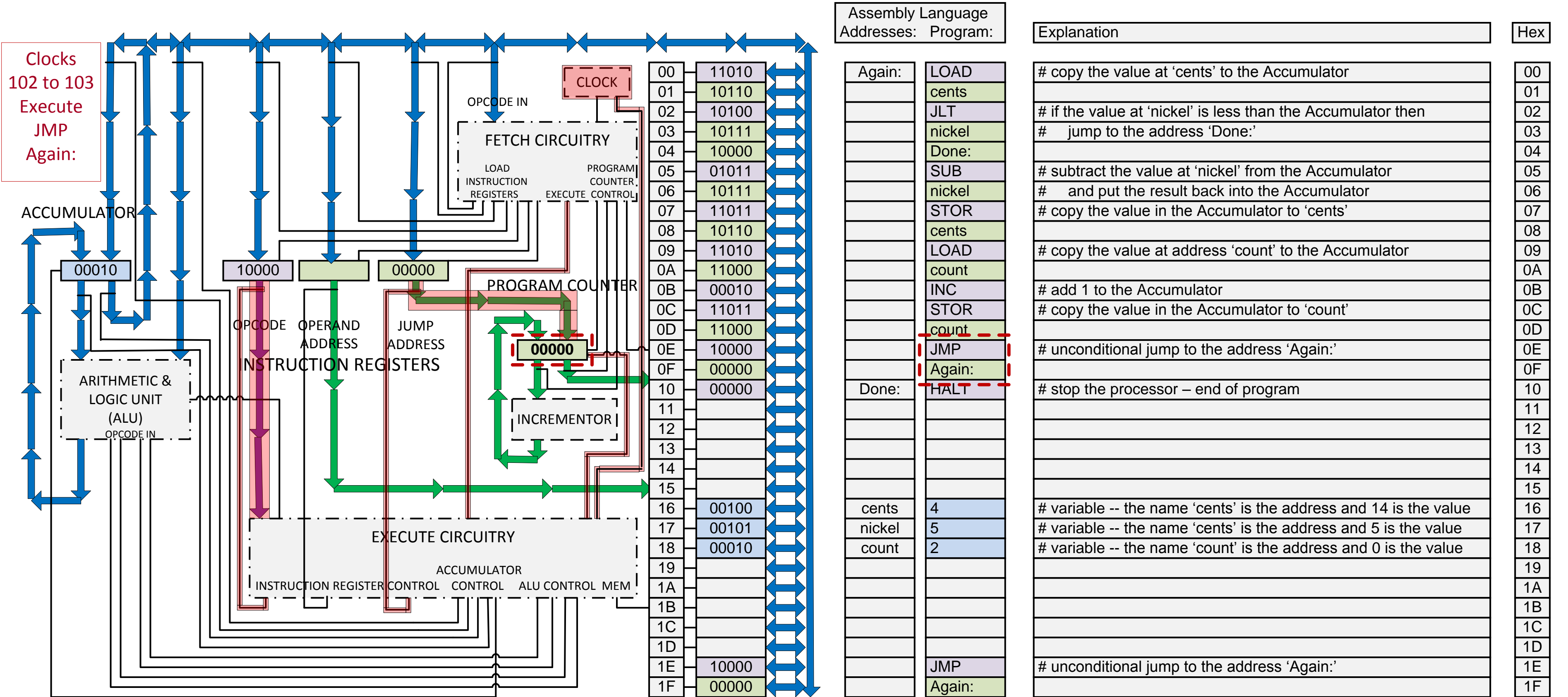
Hex

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

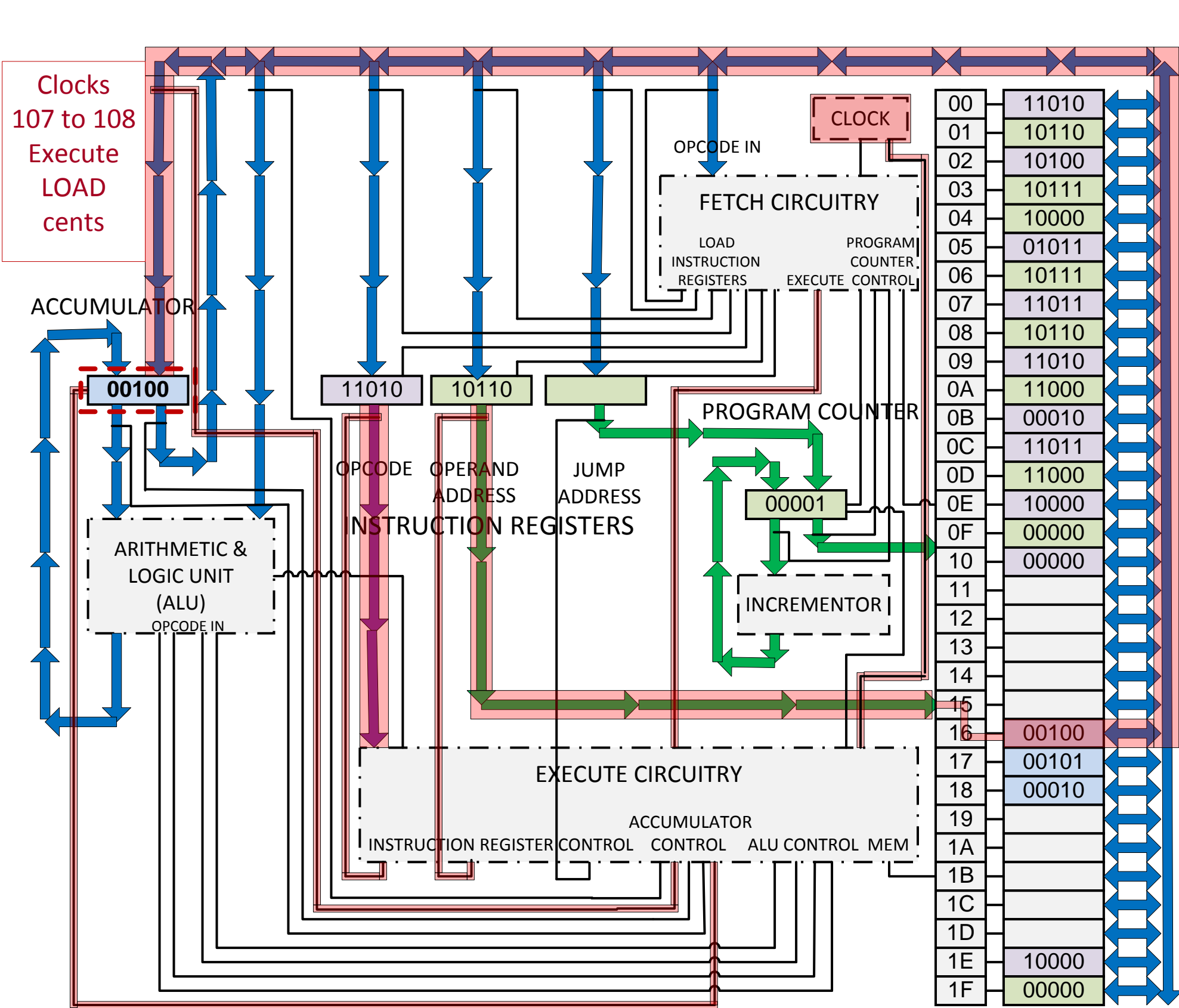


Assembly Language Addresses: Program:		Explanation	Hex
00	LOAD	# copy the value at 'cents' to the Accumulator	00
01	cents		01
02	JLT	# if the value at 'nickel' is less than the Accumulator then	02
03	nickel	# jump to the address 'Done:'	03
04	Done:		04
05	SUB	# subtract the value at 'nickel' from the 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	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 4 is the value	16
17	nickel 5	# variable -- the name 'cents' is the address and 5 is the value	17
18	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	JMP	# unconditional jump to the address 'Again:'	1E
1F	Again:		1F









Assembly Language Addresses: Program:

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

Explanation

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

Hex

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

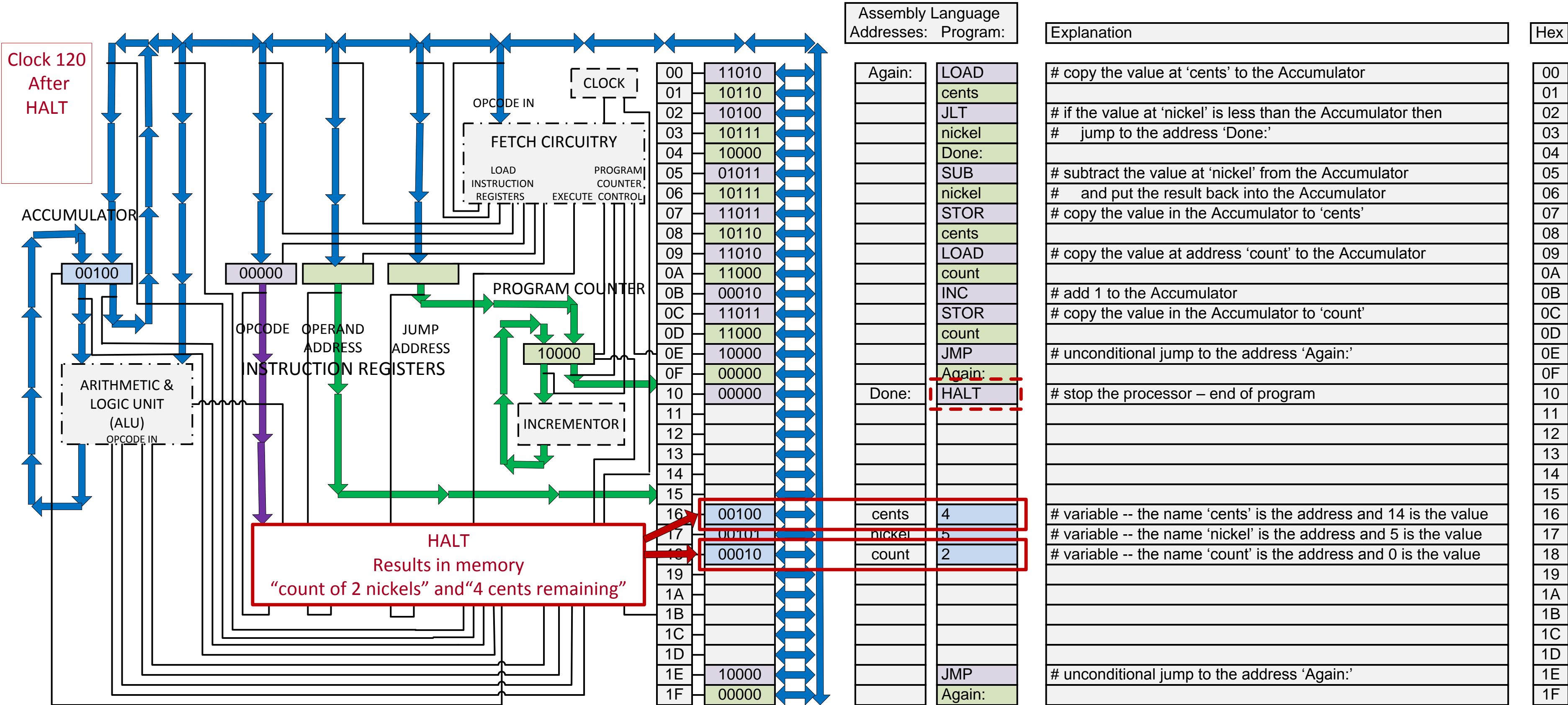












Next Presentation:  
Memory, ALU, and Control Circuitry

**End of Presentation**