Before start

ACCUMULATOR

ARITHMETIC & LOGIC UNIT (ALU)

cents: 14
nickel: 5
count: 0

Addr	Binary	Asm Addr	Program	Explanation	Hex
			Assembly Language Addresses:	Program:	Explanation
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	01110	cents	14	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	count	0	# 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

LOAD cents

ACCUMULATOR

01110

ARITHMETIC & LOGIC UNIT (ALU)

cents: 14
nickel: 5
count: 0

Address	Binary		Assembly 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	01110		cents	14	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101		nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000		count	0	# 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

JLT
nickel
Done:

ACCUMULATOR

01110

14   <   5
this is false so
no further
action is done

cents: 14
nickel: 5
count: 0

Assembly Language

Address	Binary	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	01110	cents	14	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	count	0	# 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

SUB
Nickel

ACCUMULATOR

01001

14 - 5
equals
9

cents: 14
nickel: 5
count: 0

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	01110	cents 14	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel 5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	count 0	# 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

STOR
cents

ACCUMULATOR

01001

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ̶1̶4̶ 9
nickel: 5
count: 0

Address	Binary	Assembly 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	01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	count	0	# 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

LOAD count

ACCUMULATOR

00000

ARITHMETIC & LOGIC UNIT (ALU)

TI-2511

cents: ~~14~~ 9
nickel: 5
count: 0

Addr	Binary		Assembly Addresses	Program	Explanation	Hex
			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	01001		cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101		nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000		count	0	# 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

INC

ACCUMULATOR

00001

0 + 1 equals 1

METIC & UNIT (LU)

cents: ~~14~~ 9
nickel: 5
count: 0

Addr	Binary	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	01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00000	count	0	# 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

TI-2511

CE ÷ ×
7 8 9 C
4 5 6 −
1 2 3 +
0 . =

STOR
count

ACCUMULATOR

00001

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ~~14~~ 9
nickel: 5
count: ~~0~~ 1

Address	Memory		Assembly 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	01001		cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101		nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00001		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	10000			JMP	# unconditional jump to the address 'Again:'	1E
1F	00000			Again:		1F

JMP
Again:

ACCUMULATOR

00001

ARITHMETIC &
LOGIC UNIT
(ALU)
OPCODE IN

Handwritten note:
cents: ~~14~~ 9
nickel: 5
count: ~~0~~ 1

Address	Binary	Assembly 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	01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

LOAD
cents

ACCUMULATOR

01001

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ~~14~~ 9
nickel: 5
count: ~~0~~ 1

Address	Binary	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	01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

TI-2511

JLT
nickel
Done:

ACCUMULATOR

01001

9 < 5
this is false so
no further
action is done

cents: 1̶4̶ 9
nickel: 5
count: 0̶ 1

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 01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
		17 00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
		18 00001	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 10000		JMP	# unconditional jump to the address 'Again:'	1E
		1F 00000		Again:		1F

TI-2511

SUB
Nickel

ACCUMULATOR

00100

9 - 5

equals

4

cents: ~~1~~ ~~4~~ 9
nickel: 5
count: ~~0~~ 1

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	01001	cents	9	# variable -- the name 'cents' is the address and 14 is the value	16
17	00101	nickel	5	# variable -- the name 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

STOR
cents

ACCUMULATOR

00100

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ~~14~~ 4
nickel: 5
count: ~~0~~ 1

Assembly Language Addresses:	Program:	Explanation	Hex

Addr	Binary	Assembly Address	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 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

LOAD
count

ACCUMULATOR

00001

ARITHMETIC &
LOGIC UNIT
(ALU)

TI-2511

cents: ~~14~~ ~~7~~ 4
nickel: 5
count: ~~0~~ 1

Address	Binary	Label	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 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

INC

ACCUMULATOR

00010

1 + 1 equals 2

ARITHMETIC & LOGIC UNIT (ALU)

cents: 14 4

nickel: 5

count: 0 1

Address	Value	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 'cents' is the address and 5 is the value	17
18	00001	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	10000		JMP	# unconditional jump to the address 'Again:'	1E
1F	00000		Again:		1F

STOR
count

ACCUMULATOR

00010

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ~~14~~ 4
nickel: 5
count: ~~0~~ 2

Address			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 'cents' 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

JMP
Again:

ACCUMULATOR

00010

ARITHMETIC &
LOGIC UNIT
(ALU)
OPCODE IN

Handwritten note:
cents: ~~14~~ ~~14~~ 4
nickel: 5
count: ~~0~~ ~~0~~ 2

Address	Memory	Assembly Label / 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 'cents' 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

LOAD
cents

ACCUMULATOR

00100

ARITHMETIC &
LOGIC UNIT
(ALU)

cents: ~~1~~ ~~4~~ ~~1~~ 4
nickel: 5
count: ~~0~~ ~~1~~ 2

Assembly Language				
Address	Binary	Addresses:	Program:	Explanation	Hex

Addr	Binary	Labels	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 'cents' 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

ACCUMULATOR

00100

4   <   5
this is true so
JMP is
executed

cents: ~~14~~ ~~4~~ 4
nickel: 5
count: ~~0~~ ~~X~~ 2

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 'cents' 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

ACCUMULATOR

00100

ARITHMETIC &
LOGIC UNIT
(ALU)

TI-2511

cents: ~~14~~ ~~4~~ 4

nickel: 5

count: ~~0~~ ~~X~~ 2

Address	Memory	Assembly 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 'cents' 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

ACCUMULATOR

00100

ARITHMETIC &
LOGIC UNIT
(ALU)

HALT
Results in memory and on paper
"count of 2 nickels" and "4 cents remaining"

cents: 14 4
nickel: 5
count: 0 2

MEM

Assembly Language			
	Addresses:	Program:	Explanation
00	11010	Again: LOAD	# copy the value at 'cents' to the Accumulator
01	10110	cents	
02	10100	JLT	# if the value at 'nickel' is less than the Accumulator then
03	10111	nickel	# jump to the address 'Done:'
04	10000	Done:	
05	01011	SUB	# subtract the value at 'nickel' from the Accumulator
06	10111	nickel	# and put the result back into the Accumulator
07	11011	STOR	# copy the value in the Accumulator to 'cents'
08	10110	cents	
09	11010	LOAD	# copy the value at address 'count' to the Accumulator
0A	11000	count	
0B	00010	INC	# add 1 to the Accumulator
0C	11011	STOR	# copy the value in the Accumulator to 'count'
0D	11000	count	
0E	10000	JMP	# unconditional jump to the address 'Again:'
0F	00000	Again:	
10	00000	Done: HALT	# stop the processor – end of program
11			
12			
13			
14			
15			
16	00100	cents 4	# variable -- the name 'cents' is the address and 14 is the value
17	00101	nickel 5	# variable -- the name 'nickel' is the address and 5 is the value
18	00010	count 2	# variable -- the name 'count' is the address and 0 is the value
19			
1A			
1B			
1C			
1D			
1E	10000	JMP	# unconditional jump to the address 'Again:'
1F	00000	Again:	

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

TI-2511

End of Presentation