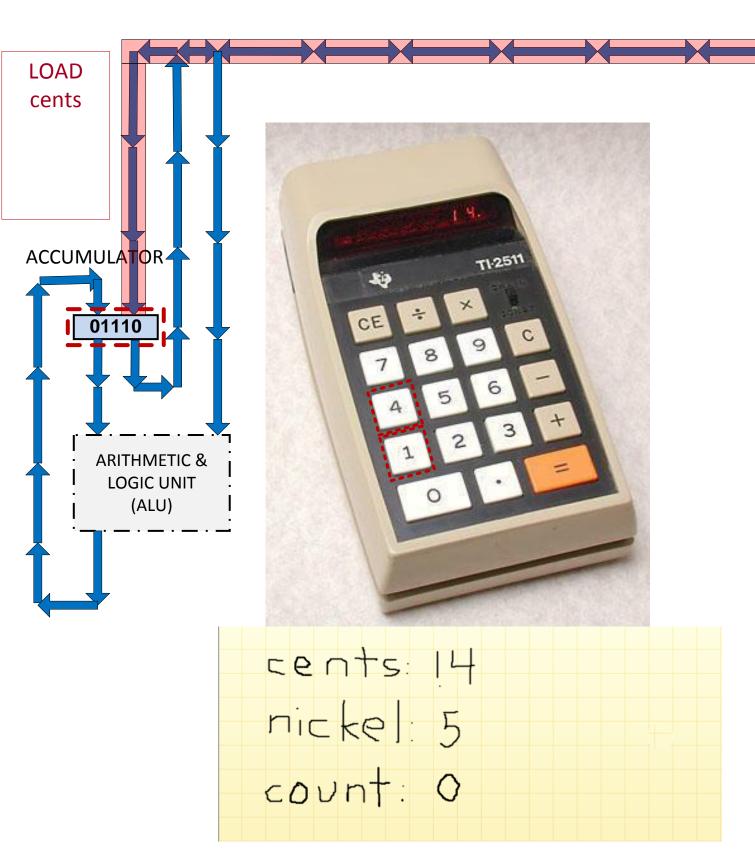


			, , , ,
00	Н	11010	
01	H	10110	
02	Н	10100	
03	Н	10111	
04	Н	10000	
05	Н	01011	
06	H	10111	
07	Н	11011	
08	Н	10110	
09	Н	11010	
0A	Н	11000	
0B	Н	00010	
0C	Н	11011	
0D	Н	11000	
0E	Н	10000	
0F	Н	00000	
10	Ц	00000	
11	H		
12	Ц		
13	H		
14	H		
15	H		
16	H	01110	
17	Ц	00101	
18	ᅡ	00000	
19	ᅡ		
1A	ᅡ		
1B	ᅡ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

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:

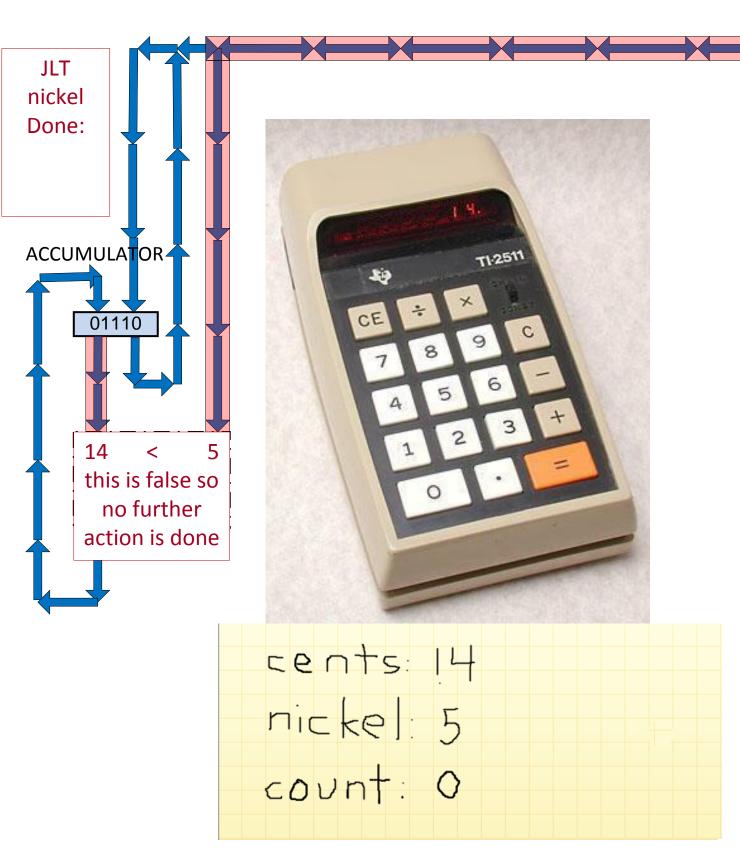
Explanation	He
# converted value of 'conto' to the Accumulator	
# copy the value at 'cents' to the Accumulator	00
# if the value at 'pickel' is less than the Assumulator than	01
# if the value at 'nickel' is less than the Accumulator then	02
# jump to the address 'Done:'	03
# subtract the value at 'nickel' from the Accumulator	04
	00
F 1 - F 2 -	07
# copy the value in the Accumulator to 'cents'	—
# converted value of address (sound) to the Accumulator	08
# copy the value at address 'count' to the Accumulator	09
# add 1 to the Accumulator	
# copy the value in the Accumulator to 'count'	00
	_
	1 1 / 11
# unconditional jump to the address 'Again:'	_
· · ·	OF
# unconditional jump to the address 'Again:' # stop the processor – end of program	OF 10
· · ·	OF 10
· · ·	OF 10 12
· · ·	0F 10 12 12
· · ·	0F 10 12 12 13 14
# stop the processor – end of program	OF 10 11 12 12 12 14 14 14
# stop the processor – end of program # variable the name 'cents' is the address and 14 is the value	0F 10 12 12 13 14 15 16
# 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	10 10 12 13 14 14 15
# stop the processor – end of program # variable the name 'cents' is the address and 14 is the value	10 10 11 12 13 14 14 15 16
# 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	OF 10 10 10 10 10 10 10 10 10 10 10 10 10
# 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	OF 10 10 12 12 12 12 12 12 12 12 12 12 12 12 12
# 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	OF 10 10 11 12 13 14 15 15 16 15 16 16 16 16 16 16 16 16 16 16 16 16 16
# 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	0E 0F 10 12 13 14 15 16 17 18 17 18
# 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	OF 10 12 13 14 14 15 16 17 18



Addresses: Program: LOAD Again: cents JET nickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator to 'cents' 80 09 # copy the value at address 'count' to the Accumulator 0A 0B 0C 0D # add 1 to the Accumulator # copy the value in the Accumulator to 'count' 0E # unconditional jump to the address 'Again:' 0F 10 # stop the processor – end of program 14 15 16 # variable -- the name 'cents' is the address and 14 is the value 17 # variable -- the name 'cents' is the address and 5 is the value 18 # variable -- the name 'count' is the address and 0 is the value 19 1A 1B 1C 1D 1E # unconditional jump to the address 'Again:'

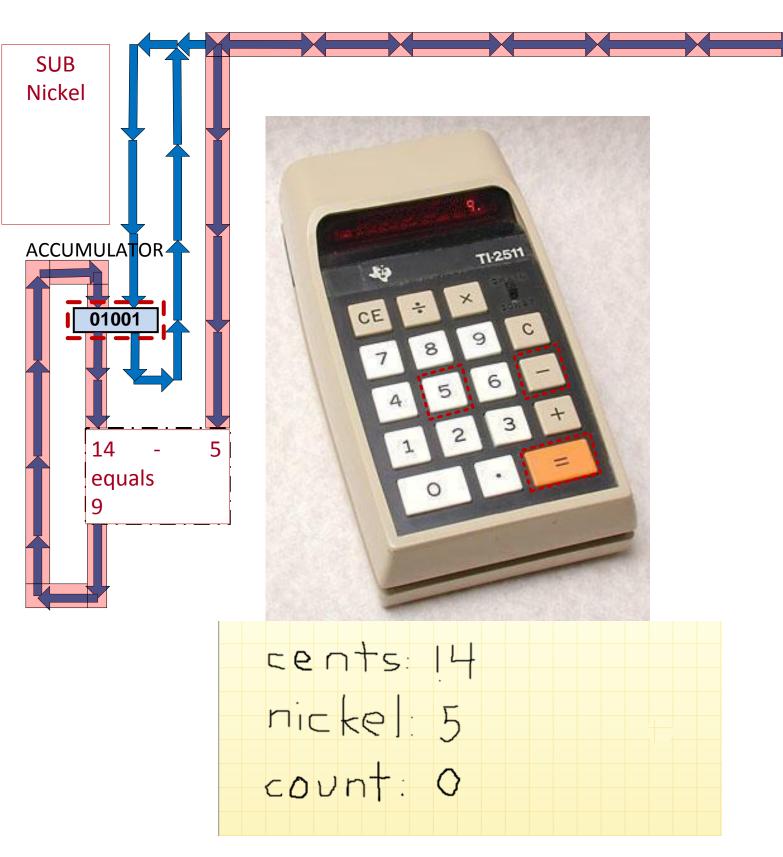


Addresses: Program: LOAD Again: cents Inickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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 1D 1E # unconditional jump to the address 'Again:'

07 80 09 0A 0B 0C 0D 0E 0F 10 14 15 16 17 18 19 1A 1B 1C



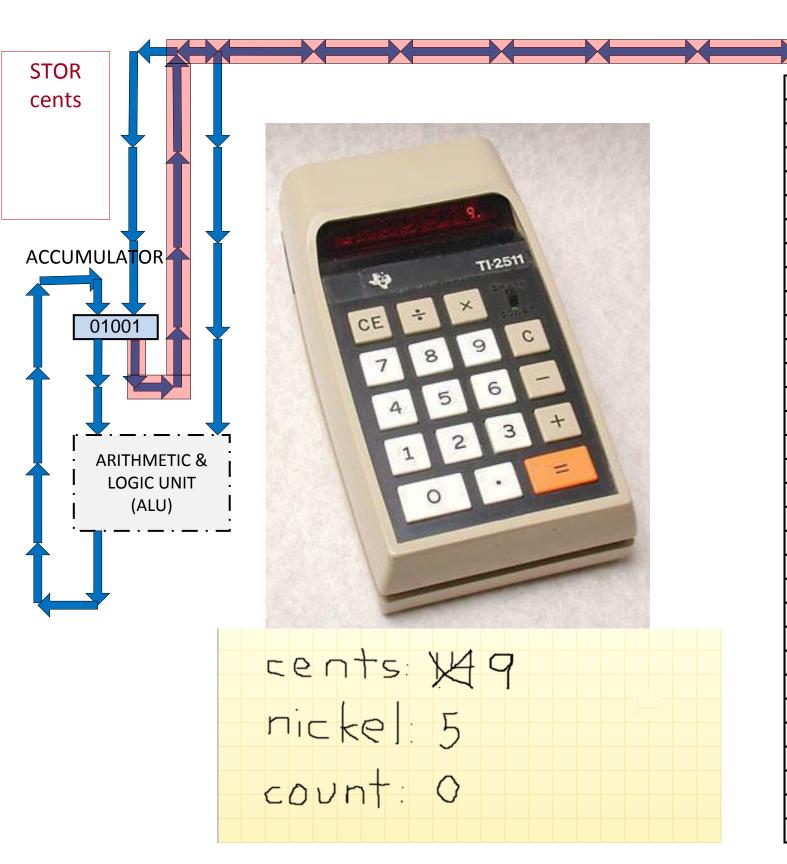
X =			
	1 5	11010	
00	Н	11010	
01	Н	10110	
02	Н	10100	
03	Н	10111	
04	Н	10000	
05	Н	01011	
06	Н	10111	
07	Н	11011	
08	Н	10110	
09	Н	11010	
0A	Н	11000	
0B	Н	00010	
0C	Н	11011	
0D	Н	11000	
0E	Н	10000	
0F	Н	00000	
10	Н	00000	
11	Н		
12	Н		
13	Н		
14	H		
15	H		
16	Н	01110	
17	Н	00101	
18	H	00000	
19	H		
1A	H		
1A 1B 1C	H		
1C	H		
1D 1E	H		
1E	H	10000	
1F	Н	00000	

Addresses: Program: LOAD Again: cents nickel LDone: _ SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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 13 14 15 16 17 18 19 1A 1B 1C 1D 1E



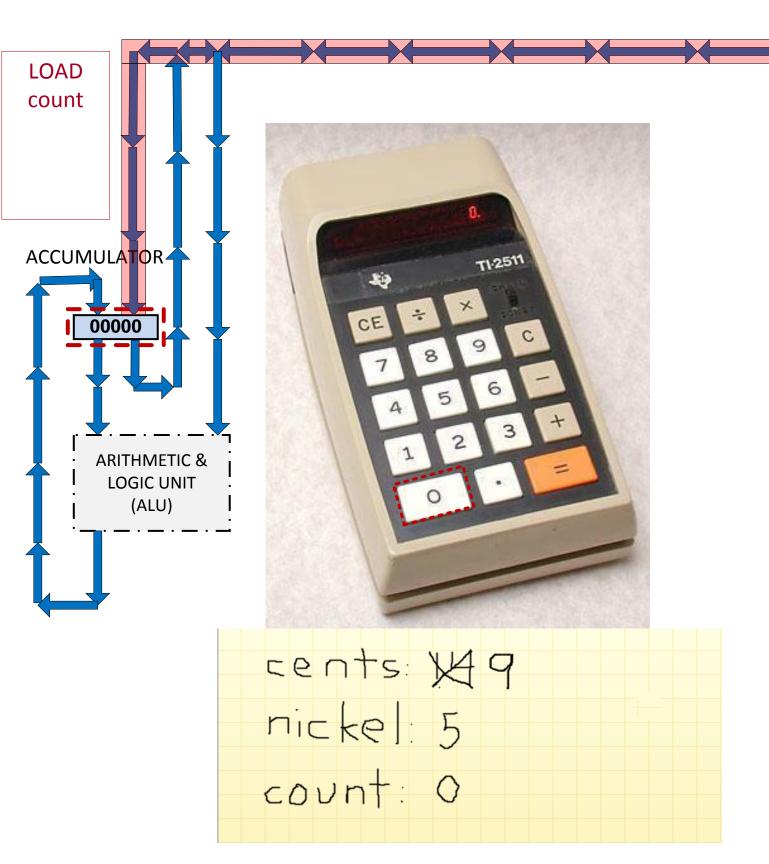
X		
00	11010	
01	10110	
02	10100	
03	10111	
04	10000	
05	01011	
06	10111	
07	11011	
08	10110	
09	11010	
0A	11000	
0B	00010	
0C	11011	
0D	11000	
0E	10000	
0F	00000	
10	00000	
11		
12		
13		
14		
15		
16	01001	
17	7 00101	
18	00000	
19		
1A		
1B		
1C		
1D		
1E	10000	
1F	00000	

Assembly Language Addresses: Program: Again: LOAD cents nickel Done: SUB Lnickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count 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:'

05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 13 14 15 16 17 18 19 1A 1B 1C 1D

1E



Addresses: Program: LOAD 11010 Again: 10110 cents 10100 03 10111 nickel 04 10000 Done: 05 SUB 01011 06 nickel STOR 11011 80 10110 Lcents 09 11010 LOAD 0A 11000 I count INC 00010 0C 11011 STOR 0D 11000 count JMP 10000 00000 Again: 10 00000 HALT Done: 13 15 16 01001 cents 00101 nickel 18 00000 count 19 1B 1D 1E 10000 JMP Again:

Assembly Language

Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator to 'cents' 80 09 # copy the value at address 'count' to the Accumulator 0A 0B 0C 0D # add 1 to the Accumulator # copy the value in the Accumulator to 'count' 0E # unconditional jump to the address 'Again:' 0F # 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 1A 1B 1C 1D 1E # unconditional jump to the address 'Again:'

10

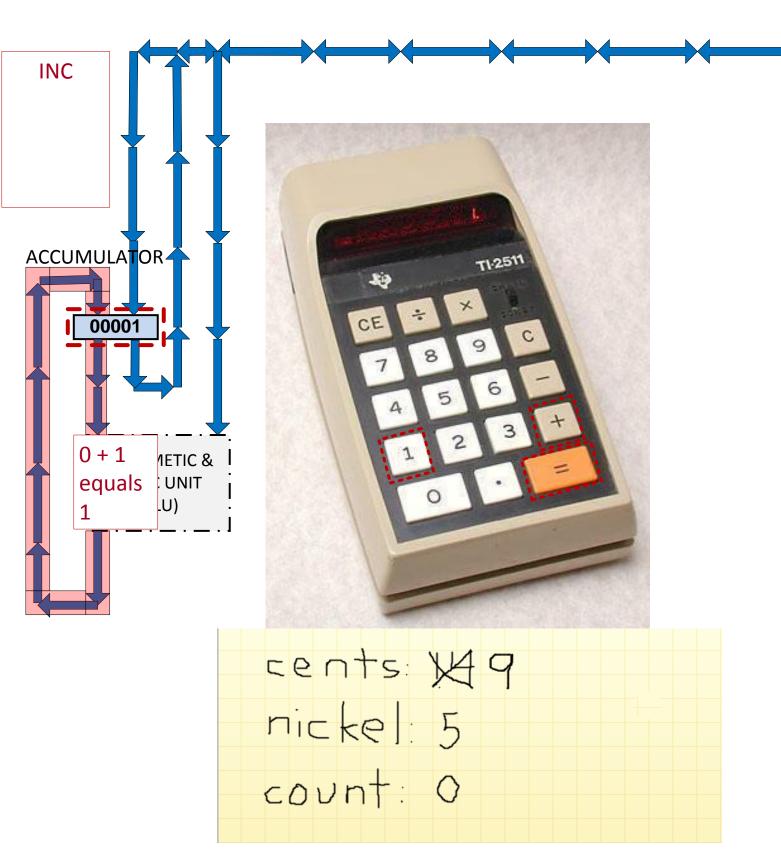
14 15

16

17

18

19



/				Addresses:		Program:
1			_			
00		11010		Again:		LOAD
01	\vdash	10110				cents
02	\vdash	10100				JLT
03	\vdash	10111				nickel
04	H	10000				Done:
05	H	01011				SUB
06	\vdash	10111				nickel
07	H	11011				STOR
80	H	10110				cents
09	H	11010				LOAD
0A		11000			_	count _
0B		00010				INC
OC		11011				STOR
OD		11000				count
0E		10000				JMP
0F		00000				Again:
10		00000		Done:		HALT
11	H					
12	H					
13	H					
14	H					
15						
16		01001		cents		9
17		00101		nickel		5
18		00000		count		0
19						
1A						
1B	H					
1C	\vdash					
1D	\vdash					
1E		10000				JMP
1F	\vdash	00000	-			Again:

Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 # subtract the value at 'nickel' from the Accumulator and put the result back into the Accumulator 07 # copy the value in the Accumulator to 'cents' # copy the value at address 'count' to the Accumulator 0B 0C 0D # add 1 to the Accumulator # copy the value in the Accumulator to 'count' 0E # unconditional jump to the address 'Again:' 0F # 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 1D 1E # unconditional jump to the address 'Again:'

05

06

08 09

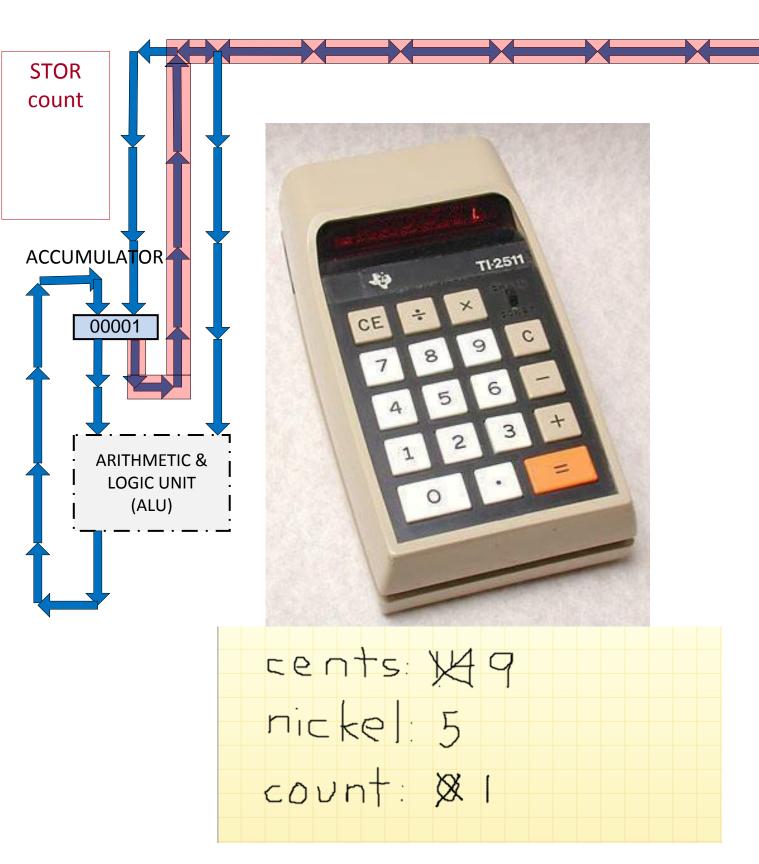
0A

15 16

18

19 1A

1B 1C

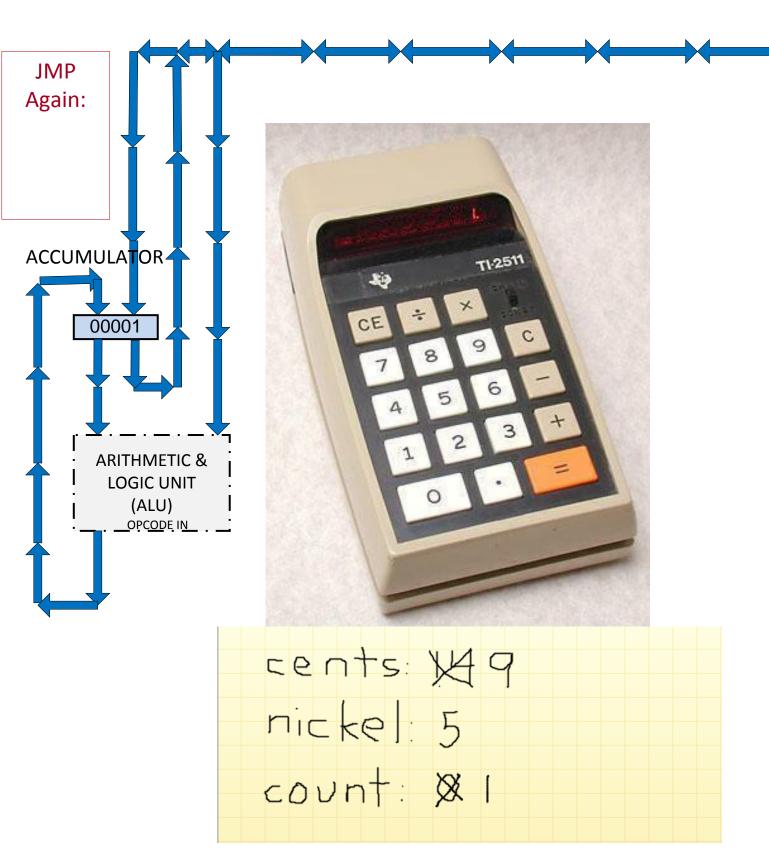


00	\vdash	11010	
01	H	10110	
02	H	10100	
03	\vdash	10111	
04	H	10000	
05	H	01011	
06	H	10111	
07	Н	11011	
80	H	10110	
09	H	11010	
0A	H	11000	
0B	H	00010	
0C	H	11011	
0D	H	11000	
0E	H	10000	
0F	H	00000	
10	H	00000	
11	Н		
12	Н		
13	Н		
14	Н		
15	Н		
16	Н	01001	
17	H	00101_	
18	H	00001	
19	H		
1A	H		
1B	Н		
1C	H		
1D	H		
1E	H	10000	
1F	\vdash	00000	

Addresses: Program: LOAD Again: cents nickel Done: SUB nickel STOR cents LOAD count INC_ STOR count JMP Again: HALT Done: cents nickel count JMP Again:

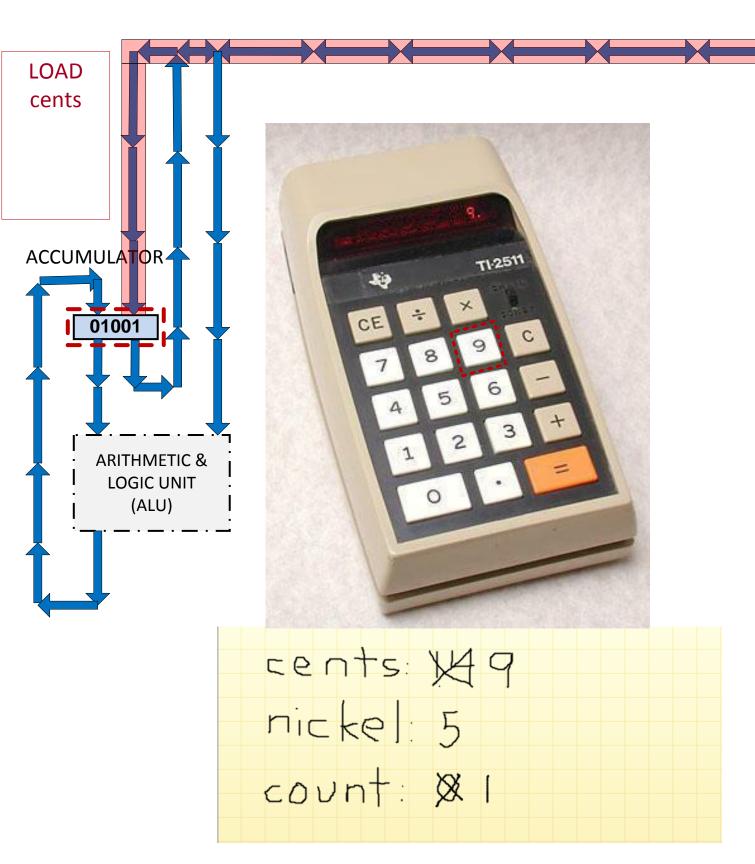
Assembly Language

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



<u> </u>					Α	Addresses	:	Progra
					_		_	
00	Н	11010				Again:		LOAD
01		10110						cents
02		10100			I			JLT
03		10111			I			nickel
04	H	10000						Done:
05	H	01011						SUB
06	Н	10111						nickel
07		11011			ı			STOR
08	Н	10110			1			cents
09		11010			1			LOAD
ΟA		11000			1			count
0B		00010			1			INC
C		11011			1			STOR
OD		11000			1			count
ΟE		10000			ı		I	JMP
0F		00000			ı			Again:
10		00000			ı	Done:	-	HALT
11					ı			
12					ı			
13					ı			
14					1			
15					1			
16		01001			1	cents		9
17		00101			1	nickel		5
18	Н	00001			1	count		1
19					1			
1A	Н				1			
1B					1			
1C	Н				j			
1D	H							
1E	H	10000						JMP
1F	H	00000			j			Again:
			- ' ' '	•			•	

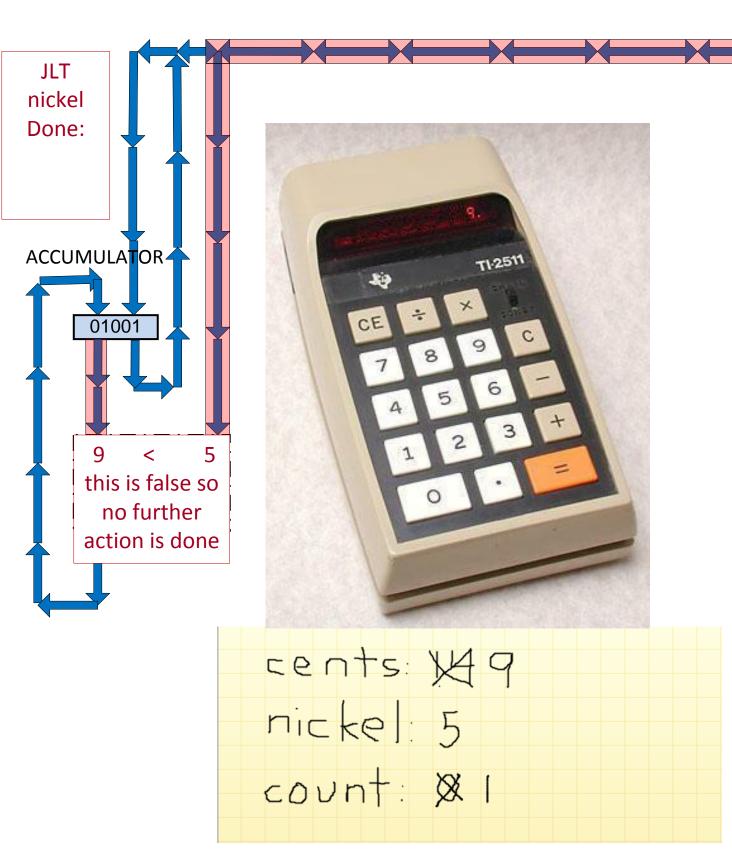
Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator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



Addresses: Program: LOAD Again: cents nickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator to 'cents' 80 09 # copy the value at address 'count' to the Accumulator 0A 0B # add 1 to the Accumulator 0C 0D # copy the value in the Accumulator to 'count' 0E # unconditional jump to the address 'Again:' 0F 10 # stop the processor – end of program 14 15 16 # variable -- the name 'cents' is the address and 14 is the value 17 # variable -- the name 'cents' is the address and 5 is the value 18 # variable -- the name 'count' is the address and 0 is the value 19 1A 1B 1C 1D 1E # unconditional jump to the address 'Again:'

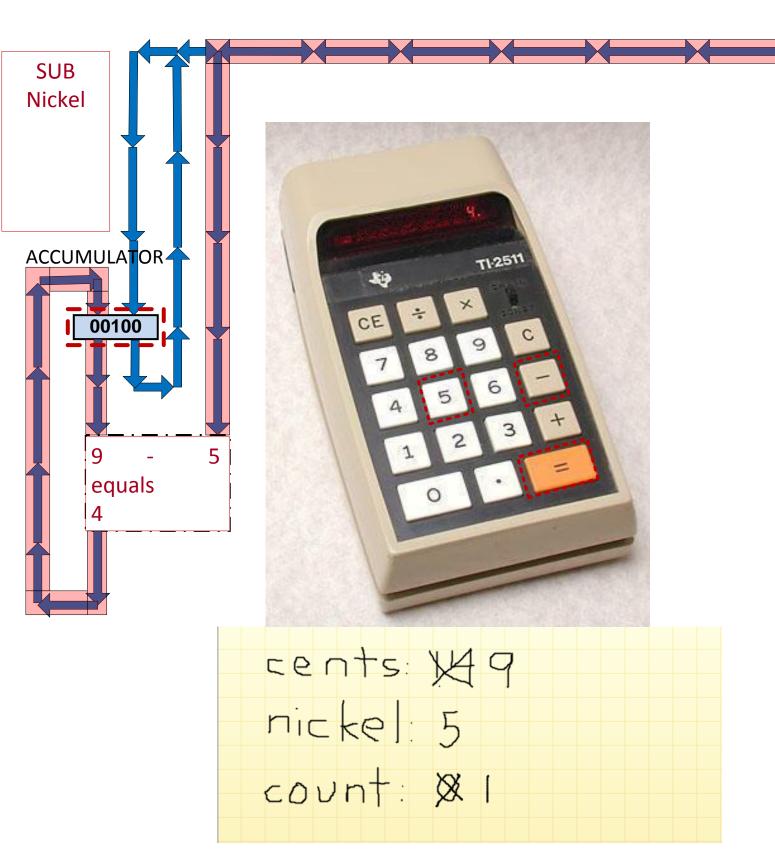


X			
00	Н	11010	
01	Н	10110	
02	Н	10100	
03	Ц	10111	
04	Н	10000	
05	Н	01011	
06	Н	10111	
07	Н	11011	
08	Н	10110	
09	Н	11010	
0A	Н	11000	
0B	Н	00010	
0C	Н	11011	
0D	H	11000	
0E	H	10000	
0F	Н	00000	
10	Н	00000	
11	Н		
12	Н		
13	Н		
14	Н		
15	Н		
16	Н	01001	
17	Н	00101	
18	Н	00001	
19	Н		
1A	Н		
1B	Н		
1C	Н		
1D	Н		
1E	Н	10000	
1F	Н	00000	

Addresses: Program: Again: LOAD cents Inickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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

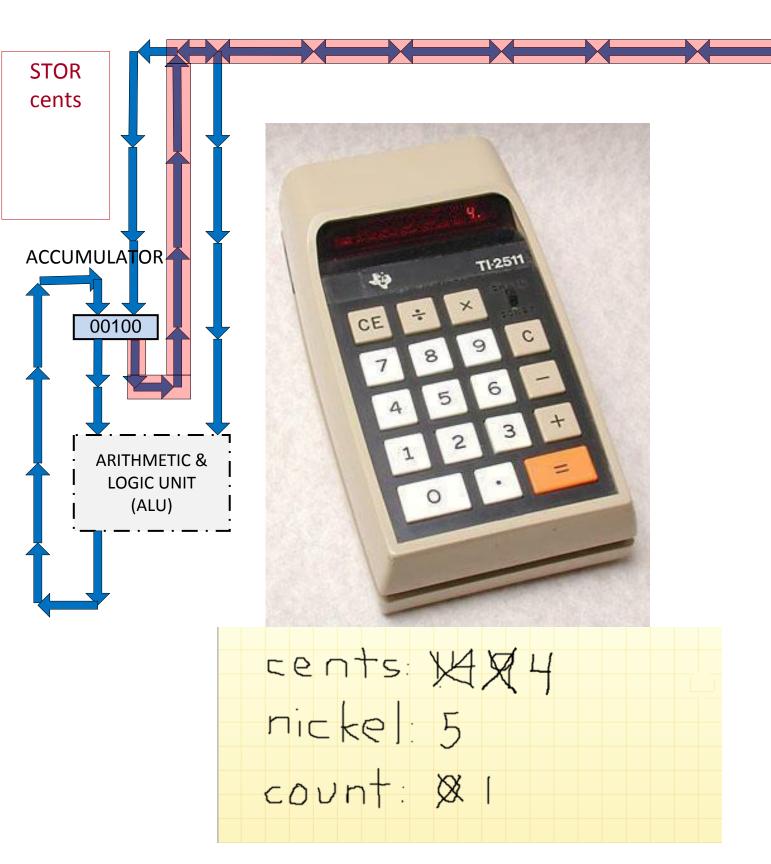


Addresses: Program: LOAD Again: cents nickel LDone: _ SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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:'

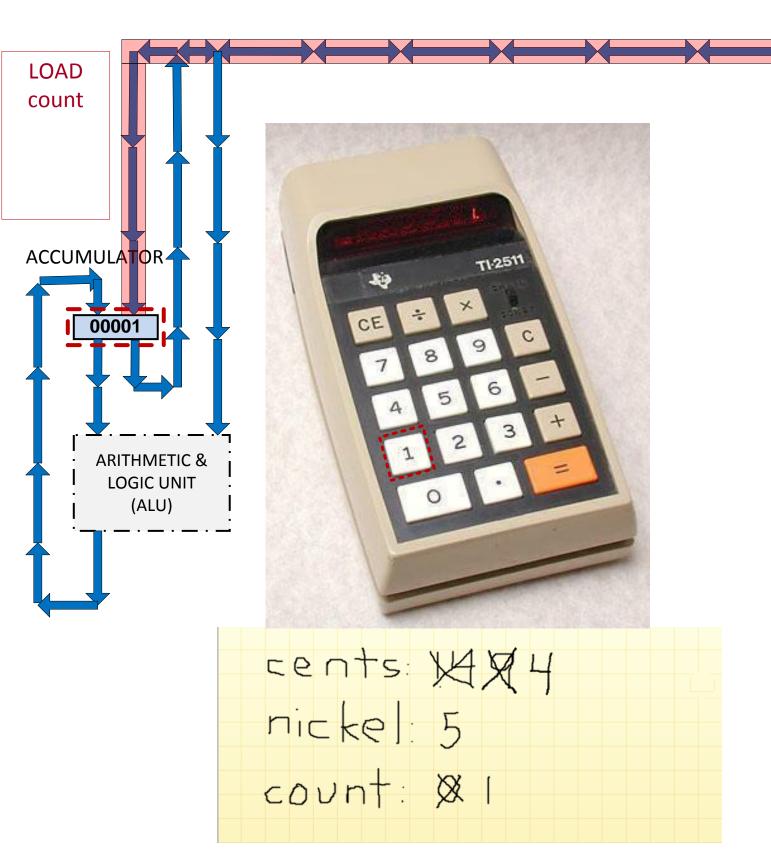
Hex



X			
00	Ц	11010	
01	ㅂ	10110	
02	Ц	10100	
03	Ц	10111	
04	ㅂ	10000	
05	Ц	01011	
06	Н	10111	
07	Н	11011	
80	H	10110	
09	H	11010	
0A	H	11000	
0B	H	00010	
0C	Н	11011	
0D	Н	11000	
0E	Н	10000	
0F	Н	00000	
10	Н	00000	
11	Н		
12	Н		
13	Н		
14	Н		
15	H		
16	H	00100	
17	H	00101	
18	\mathbb{H}	00001	
19	\mathbb{H}		
1A	\mathbb{H}		
1B	H		
1C	H		
1D	H		
1E	\mathbb{H}	10000	
1F	H	00000	

Assembly Addresses:	
Again:	LOAD
9	cents
	JLT
	nickel
	Done:
	SUB
	nickel
	STOR
	cents
•	LOAD
	count
	INC
	STOR
	count
	JMP
Donoi	Again: HALT
Done:	HALI
cents	4
nickel	5
count	1
	JMP
	Again:

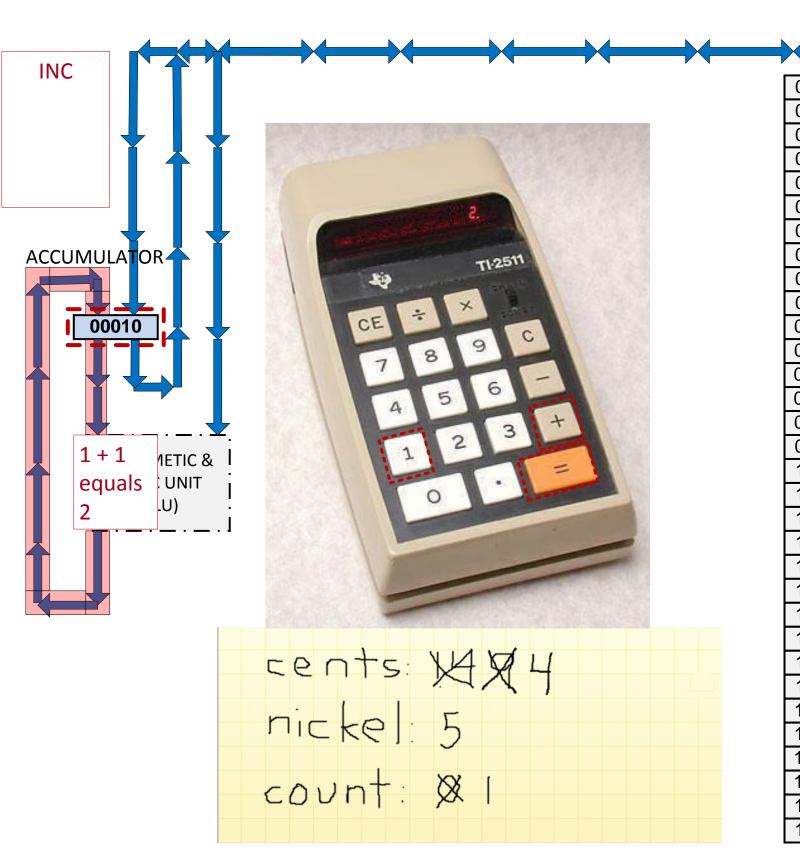
Explanation	He
# conv the value at 'conte' to the Acquirilator	
# copy the value at 'cents' to the Accumulator	00
# if the value at 'nickel' is less than the Accumulator then	02
# jump to the address 'Done:'	03
" jump to the dual coo Bone.	04
# subtract the value at 'nickel' from the Accumulator	0:
# and put the result back into the Accumulator	06
# copy the value in the Accumulator to 'cents'	0
	08
# copy the value at address 'count' to the Accumulator	09
	0/
# add 1 to the Accumulator	OE
# copy the value in the Accumulator to 'count'	00
	10
# unconditional jump to the address 'Again:'	OF
	01
# stop the processor – end of program	10
	1
	12
	1:
	14
	1:
# variable the name 'cents' is the address and 14 is the value	10
# variable the name 'cents' is the address and 5 is the value	1
# variable the name 'count' is the address and 0 is the value	18
	19
	1/
	16
	10
	1[
# unconditional jump to the address 'Again:'	1E



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

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

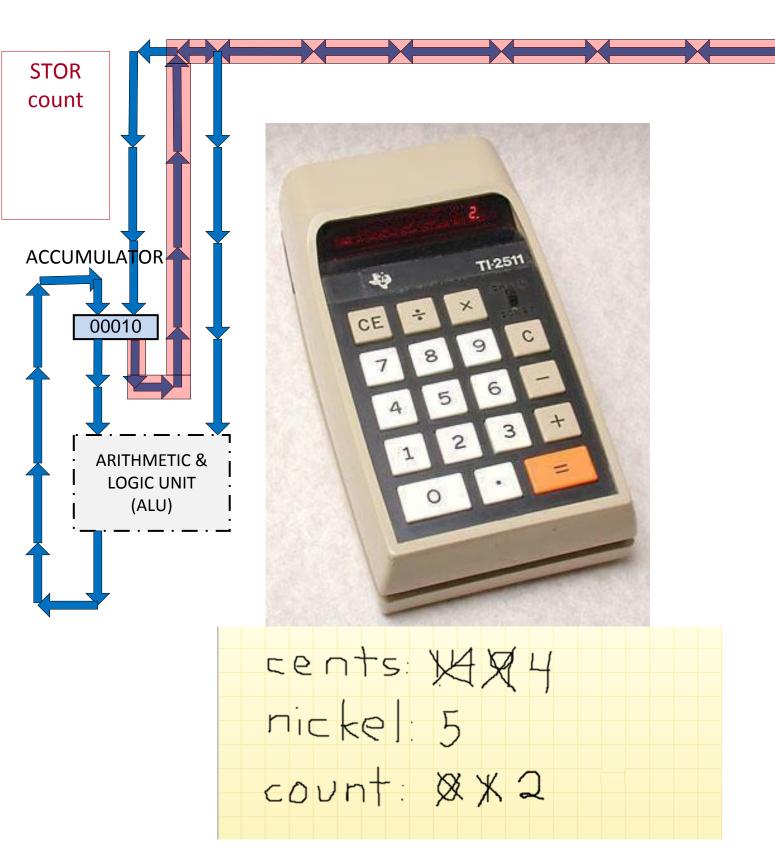
Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator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



			Assembly Language			
				Addresses	:	Program:
					1	
00	11010			Again:	ļ	LOAD
)1 –	10110				ļ	cents
2 –	10100					JLT
3 –	10111					nickel
)4 –	10000					Done:
5 –	01011					SUB
6 –	10111					nickel
7	11011					STOR
8	10110					cents
9 –	11010					LOAD
A	11000				_	count _
В	00010					INC
C -	11011				-	STOR
D	11000					count
E	10000					JMP
F	00000					Again:
0	00000			Done:		HALT
1						
2 -						
3 –						
4						
5	-					
6	00100			cents		4
7	00101			nickel		5
8	00001			count		1
9	-					
A						
B – C –						
C	-				1	
D -	_				1	
E –	10000				1	JMP
F -	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 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 15 16 18 19 1A 1B 1C 1D 1E



Addresses: Program: LOAD Again: cents nickel Done: SUB nickel STOR cents LOAD count INC_ STOR count JMP Again: HALT Done: cents nickel count JMP Again:

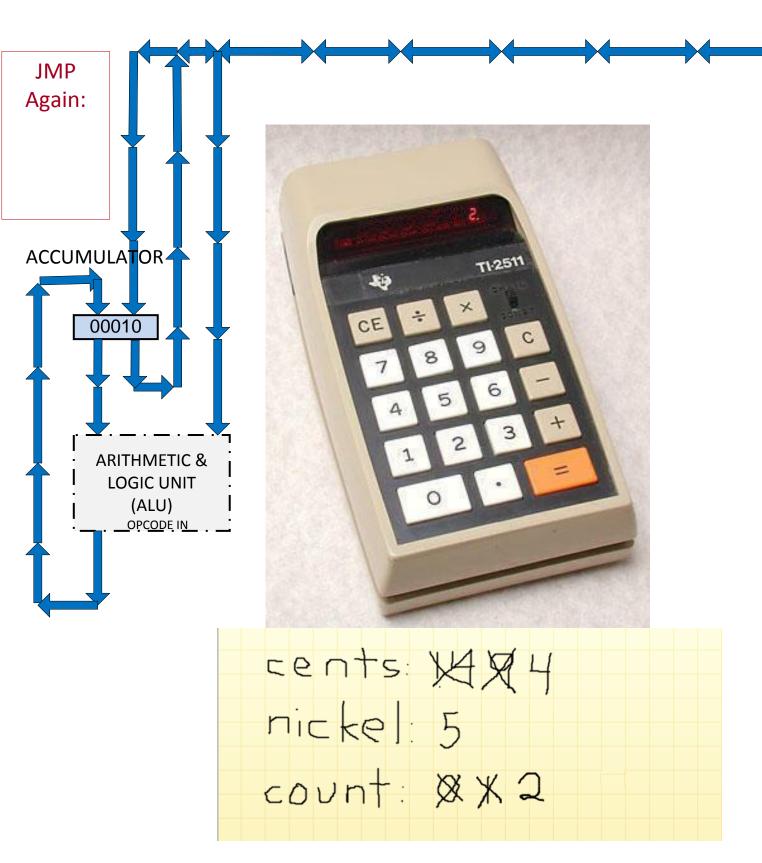
Assembly Language

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

1D

1E



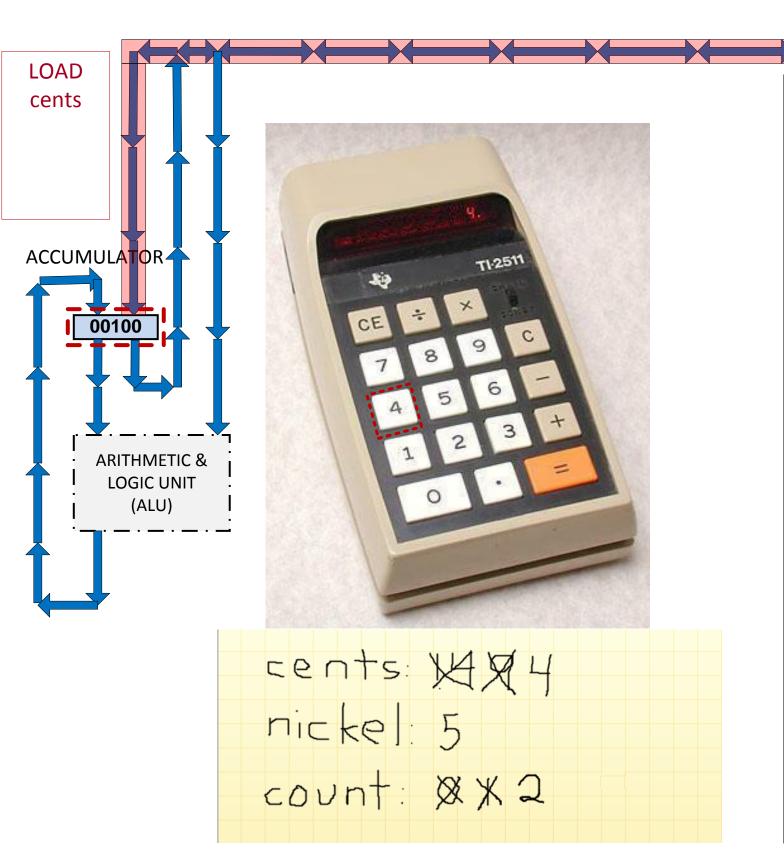
<u> </u>			Addresses:	Program:
00	\vdash	11010	Again:	LOAD
01		10110		cents
02	\vdash	10100		JLT
03	H	10111		nickel
04		10000		Done:
05		01011		SUB
06		10111		nickel
07	\vdash	11011		STOR
80	H	10110		cents
09	H	11010		LOAD
0A	H	11000		count
0B	H	00010		INC
0C	H	11011		STOR
0D	H	11000		count _
0E	\vdash	10000		JMP
0F	\vdash	00000		Again:
10	\vdash	00000	Done:	HALT
11	\vdash			
12	\vdash			
13	\vdash			
14	\vdash			
15	\vdash			
16		00100	cents	4
17		00101	nickel	5
18		00010	count	2
19				
1A				
1B	H			
1C	\vdash			
1D	\vdash			
1E	\vdash	10000		JMP
1F	\vdash	00000		Again:

Hex Explanation # copy the value at 'cents' to the Accumulator 00 01 02 # if the value at 'nickel' is less than the Accumulator then 03 jump to the address 'Done:' 04 05 # subtract the value at 'nickel' from the Accumulator 06 and put the result back into the Accumulator 07 # copy the value in the Accumulator to 'cents' 08 09 # copy the value at address 'count' to the Accumulator 0A 0B 0C 0D 0E # add 1 to the Accumulator # copy the value in the Accumulator to 'count' # unconditional jump to the address 'Again:' 0F # 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 1A 1B 1C 1D 1E # unconditional jump to the address 'Again:'

14 15 16

18

19



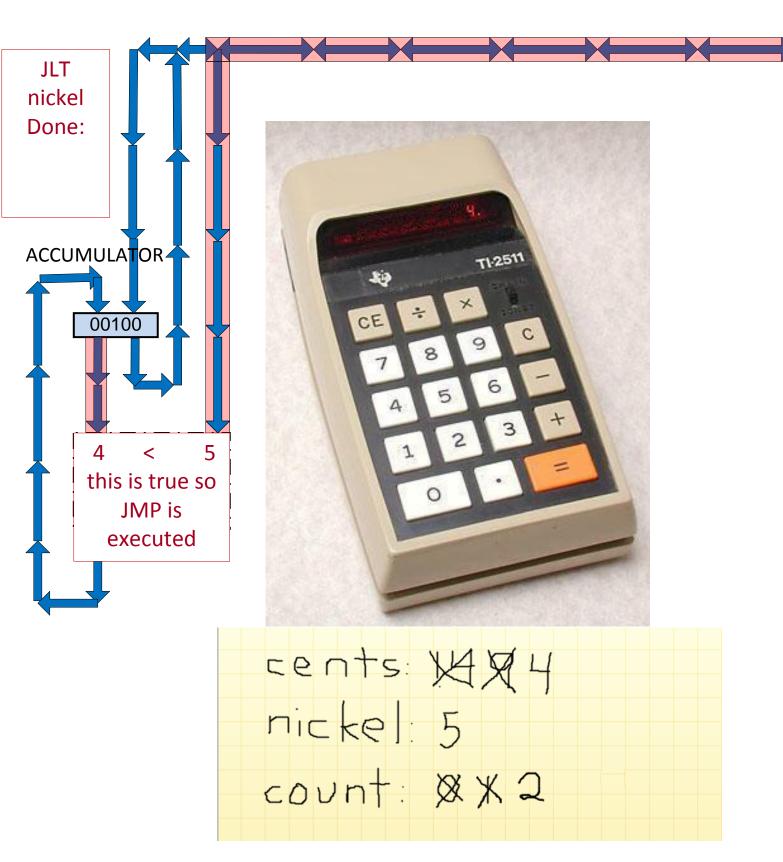
X			
00	Ш	11010	
01		10110	
02		10100	
03		10111	
04	Н	10000	
05	H	01011	
06	Н	10111	
07	Н	11011	
08	Н	10110	
09	H	11010	
0A	Н	11000	
0B	Н	00010	
0C		11011	
0D	Н	11000	
0E	H	10000	
0F		00000	
10	Н	00000	
11	Н		
12	H		
13	Н		
14	Н		
15	Н		
16	Н	00100	
17	Н	00101	
18	Н	00010	
19	Н		
1A	Н		
1B	Н		
1C	H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

Addresses: Program: LOAD Again: cents nickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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



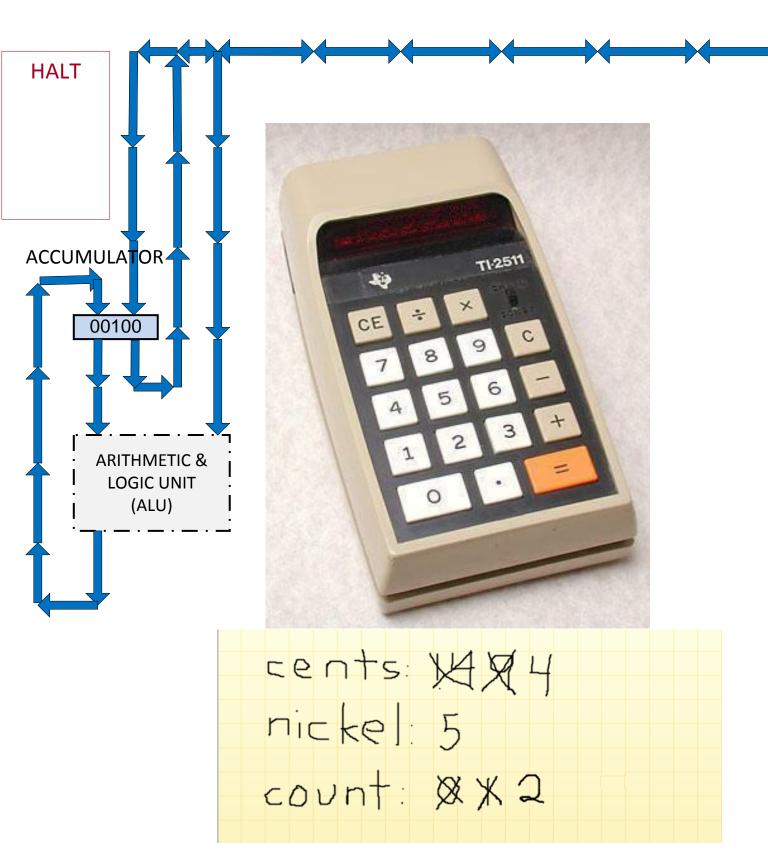
X		
00	11010	
01	10110	
02	10100	
03	10111	
04	10000	
05	01011	
06	10111	
07	11011	
08	10110	
09	11010	
0A	11000	
0B	00010	
0C	11011	
0D	11000	
0E	10000	
0F	00000	
10	00000	
11		
12		
13		
14		
15 -		
16	00100	
17	00101	
18	00010	
19		
1A		
1B		
1C		
1D		
1E	10000	
1F	00000	

Addresses: Program: LOAD Again: cents Inickel Done: SUB nickel STOR cents LOAD count INC STOR count JMP Again: HALT Done: cents nickel count JMP Again:

Assembly Language

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

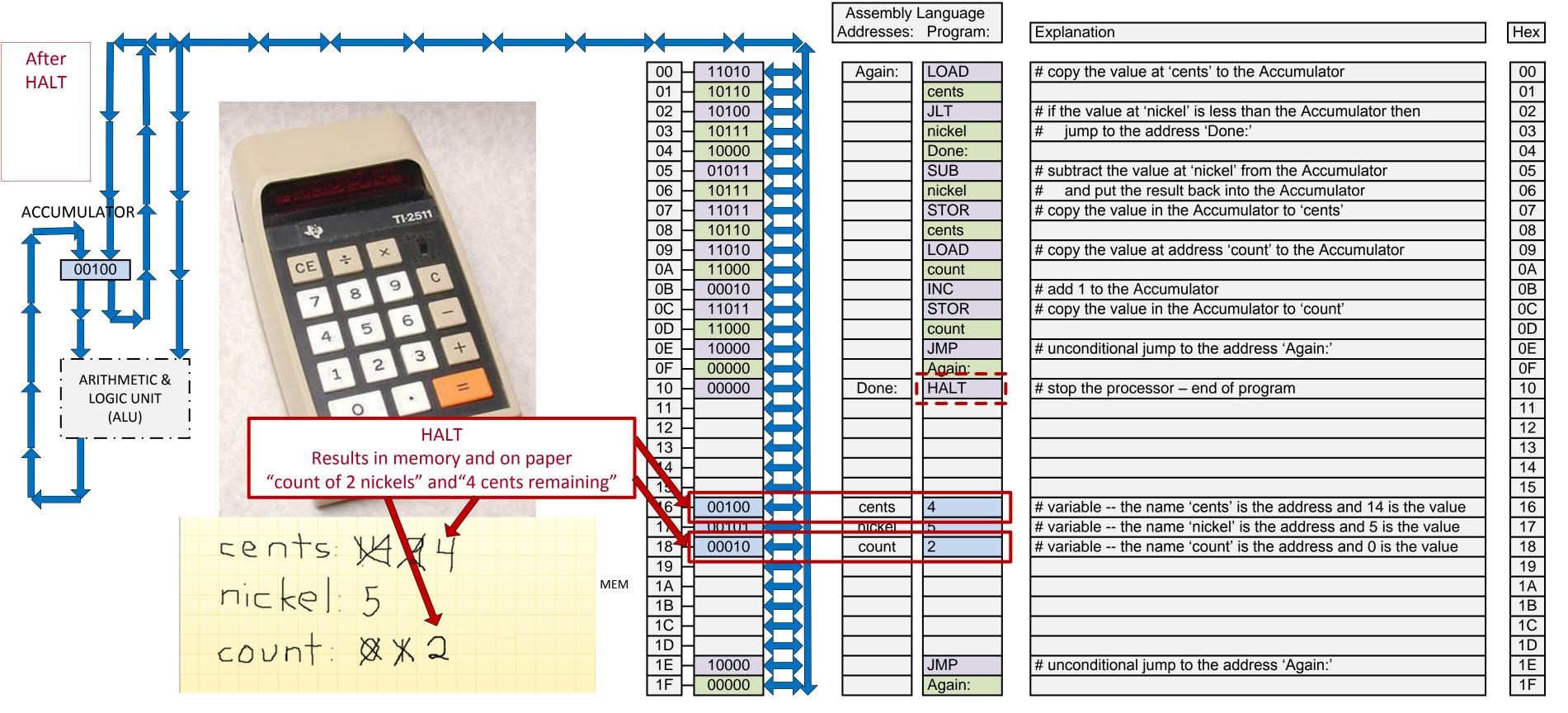
Hex



1				1	Addresses	:	Program:
V				_			
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
AC		11000					count
)B		00010					INC
C		11011					STOR
)D		11000					count
ЭE		10000					JMP
0F		00000					Again:
10		00000			Done:		HALT
11							
12							
13	_						
14	_						
15	-						
16	-	00100			cents		4
17		00101			nickel		5
18	H	00010			count		2
19	H						
1A							
1B	-						
1C	H						
1D	H						
1E	H	10000					JMP
1F	-	00000	\longleftarrow	-			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 15 16 18 19 1A 1B 1C 1D 1E



Next Presentation: Memory, ALU, and Control Circuitry

End of Presentation