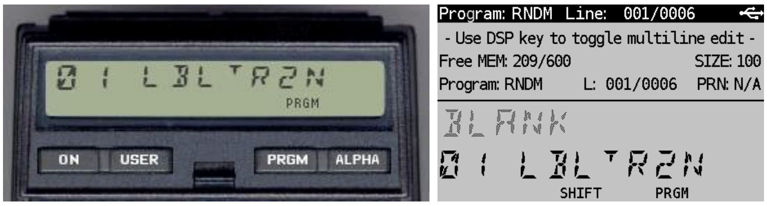
ROMAN

Display



```
Program: RNDM  Line:  001/0006
- Use DSP key to toggle multiline edit -
Free MEM: 209/600              SIZE: 100
Program: RNDM      L: 001/0006  PRN: N/A
BLANK
01 LBL ᵀR2N
          SHIFT       PRGM
```

(HP-41CX, Hewlett Packard 1983 and DM41X, SwissMicros 2020)

Overview[1]

Programs `R2N` (and `R3N`) and `N2R` convert decimal (Arabic) numbers to Roman notation and vice versa. The number conversion goes from 1-3999.

Many conversion programs have been created by several users aiming to build with the least number of bytes for the programs. The programs in this document require HP-41CX functions XTOA and ATOX. The main difficulty with the conversion to and from Roman numbers is the so-called subtractive rule. To convert numbers correctly the conversion rules are to be followed precisely. Here is a short summary. Roman numerals are written using seven different letters: I, V, X, L, C, D and M, they represent the numbers 1, 5, 10, 50, 100, 500 and 1,000. The use these seven letters makes up thousands of others. For example, the Roman numeral for 2 is written as 'II' which is just two 1's smushed together. The number 12 is XII which is just X (10) + II (2). Taking it a step further, the number 27 is written as XXVII, which when broken down looks like XX (20) + V (5) + II (2); all totaled up it equals to 27.

Roman numerals are usually written largest to smallest from left to right. However, this is not always true. The Romans didn't like writing four of the same numerals in a row, so they developed a system of subtraction.

The Roman numeral for 3 is written III, but 4 is not IIII. Instead, the subtractive principle is used. The number 4 is written as 'IV'. It shows the I (1) before V (5) and because the smaller number is before the larger number, it must be subtracted here – giving the value 4 for IV. The same principle applies to the number 9, which is written as IX.

There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

The number 994 is a great example of this rule – it's written CMXCIV. Broken down we have CM = 900, XC = 90 and IV = 4; adding all these up results in 994.

The solution approach to code the algorithm is from Sriharsha Sammeta as described in: https://www.geeksforgeeks.org/converting-decimal-number-lying-between-1-to-3999-to-roman-numerals/amp/

[1] *This program is copyright and is supplied without representation or warranty of any kind. The author assumes no responsibility and shall have no liability, consequential or otherwise, of any kind arising from the use of this program material or any part thereof*

Conversions

The solution approach follows the algorithm in which the normal and subtractive rules are implemented in a straightforward manner. In this approach the **MainSignificantDigit** in the number is considered first. For example, in 1234, the main significant digit is 1. Similarly in 345 it is 3.

To extract the main significant digit out, a divisor (div) like 1000 for 1234 (since 1234 / 1000 = 1) is required which is 100 in the case of 345 is 3 (345 / 100).

A lookup called **RomanNumeral** is defined as = {1 : 'I', 5: 'V', 10: 'X', 50: 'L', 100: 'C', 500: 'D', 1000: 'M'}

For each digit in the decimal number, the following logic applies (in which **div** refers to the Roman base):

```
if MainSignificantDigit <= 3
      RomanNumeral [div] * MainSignificantDigit

if MainSignificantDigit == 4
      RomanNumeral [div] + RomanNumeral [div*5]

if 5 <= MainSignificantDigit <= 8
      RomanNumeral [div*5] + (RomanNumeral [div] * (MainSignificantDigit-5))

if MainSignificantDigit == 9
      RomanNumeral [div] + RomanNumeral [div*10]
```

Here is an example: suppose the input number is 3984.

Iteration 1: Initial number = 3984
MainSignificantDigit is 3; div = 1000. RomanNumeral [1000] * 3 gives: MMM

Iteration 2: Updated number = 984
MainSignificantDigit is 9; div = 100.
RomanNumeral [100] + RomanNumeral [100*10] gives: CM

Iteration 3: Updated number = 84
MainSignificantDigit is 8; div = 10.
RomanNumeral [10*5] + RomanNumeral [10]*(8-5) gives: LXXX

Iteration 4: Updated number = 4
MainSignificantDigit is 4; div = 1.
RomanNumeral [1] + RomanNumeral [10*5] gives: IV

The result by clubbing all the above gives MMMCMLXXXIV for the number 3984.

Example (1): N2R

KEYSTROKES	DISPLAY	COMMENTS
[XEQ][ALPHA]N2R[ALPHA]	N=?	Start program to convert to Roman
3984[R/S]	MMMCMLXXXIV	Get the Roman notation for 3984
[R/S]	N=?	Run it again
858[R/S]	DCCCLVIII	The Roman notation of 858

Example (2): R2N

KEYSTROKES	DISPLAY	COMMENTS
[XEQ][ALPHA]R2N[ALPHA]	R=?	Start program to convert to decimals
XCVIII[R/S]	N=98	Get the number of XCVIII
[R/S]	R=?	Run it again
MDCCCLXVIII[R/S]	N= 1868	Get the number of MDCCCLXVIII
[R/S]	R=?	Try another one

Program Design

One of the challenges in the HP-41CX programming language is to create lookup tables. For the conversion from decimal to Roman numbers, the Roman (alphanumerical) numbers have been stored in R01-R07 with the ASTO instruction.

For the conversion from Roman to decimal numbers, a lookup table is created which stores the decimal values of the Roman equivalents as integer and the character code of the Roman notation as fraction. For example: the character "M" is stored in R07 as 1000,77. The second challenge is the non-linear mixed radix base for Roman numbers. The first radix needs to be multiplied by 5 to get to 5 but from the second to the third a multiplication by 2 must be applied. During initialization in `N2R` this alternation was done via a MOD instruction. The linear reference to the registers was coincidentally found to be done via a LOG function, supplemented with 1 and multiplied by 1,75. The lookup table then shows as follows:

BASE	ROMAN	CHAR	1,75*(BASE+1)	REGISTER	VALUE
1	I	73	**1**,7500	R01	1,73
5	V	86	**2**,9732	R02	5,86
10	**X**	88	**3**,5000	R03	10,88
50	L	76	**4**,7232	R04	50,76
100	**C**	67	**5**,2500	R05	100,67
500	D	68	**6**,4732	R06	500,68
1000	**M**	77	**7**,0000	R07	1000,77

Above initialisation was done via a loop with LBL 00. Instead of the LBL 00 routine in `N2R` the numerical values from above table could also be written as hard coded listing like in `R2N` to initialise with alphanumerical values, see `R3N` in which also LBL 05 and GTO 05 has been taken out because there is no need to bypass the LBL 00 initialisation loop anymore, making `R3N` one byte shorter than `R2N`.

Program Listing

The listing of programs N2R (Numerical to Roman) is given below:

```
01▪LBL "N2R"
02 "I"
03 ASTO 01
04 "V"
05 ASTO 02
06 "X"
07 ASTO 03
08 "L"
09 ASTO 04
10 "C"
11 ASTO 05
12 "D"
13 ASTO 06
14 "M"
15 ASTO 07
16 "N=?"
17 PROMPT
18 CLA
19 STO 00
20 1 E4
21 STO 08
22▪LBL 10
23 10
24 ST/ 08
25 RCL 00
26 X=0?
27 GTO 11
28 RCL 08
29 /
30 INT
31 X=0?
32 GTO 10
33 STO 10
34 XEQ IND X
35 RCL 10
36 RCL 08
37 *
38 ST- 00
39 GTO 10
40▪LBL 01
41▪LBL 02
42▪LBL 03
43 1
44 XEQ 12
45 DSE Y
46 XEQ 13
47 RTN
48▪LBL 04
49 1
50 XEQ 12
51 5
52 XEQ 12
53 RTN
54▪LBL 05
55▪LBL 06
56▪LBL 07
57▪LBL 08
58 5
59 XEQ 12
60 RCL Y
61 5
62 -
63 X=0?
64 RTN
65 1
66 XEQ 12
67 DSE Y
68 XEQ 13
69 RTN
70▪LBL 09
71 1
72 XEQ 12
73 10
74 XEQ 12
75 RTN
76▪LBL 12
77 RCL 08
78 *
79 LOG
80 1.75
81 *
82 LASTX
83 +
84 ARCL IND X
85 RTN
86▪LBL 13
87 ARCL IND X
88 DSE Y
89 GTO 13
90 RTN
91▪LBL 11
92 AVIEW
93 END
```

(156 bytes)

and for R2N (Roman to Numerical) shown here:

```
01▪LBL "R2N"
02 7
03 "MDCLXVI"
04 1000
05▪LBL 00
06 STO IND Y
07 ATOX
08 100
09 /
10 ST+ IND Z
11 RDN
12 5
13 /
14 RCL Y
15 2
16 MOD
17 1.5
18 *
19 1
20 +
21 *
22 DSE Y
23 GTO 00
24▪LBL 05
25 .
26 STO 00
27 STO 08
28 "R=?"
29 AON
30 PROMPT
31 AOFF
32▪LBL 01
33 7
34 STO 09
35 ATOX
36 X=0?
37 GTO 02
38▪LBL 03
39 RCL IND 09
40 FRC
41 100
42 *
43 X=Y?
44 GTO 04
45 RDN
46 DSE 09
47 GTO 03
48▪LBL 04
49 RCL IND 09
50 INT
51 RCL 08
52 X<Y?
53 ST- 00
54 X<Y?
55 ST- 00
56 RDN
57 ST+ 00
58 STO 08
59 GTO 01
60▪LBL 02
61 "N="
62 FIX 00
63 CF 29
64 ARCL 00
65 FIX 05
66 SF 29
67 PROMPT
68 GTO 05
69 END
```

(121 bytes)

The alternative of R2N is listed as R3N (with hard coded initialisation):

```
01▪LBL "R3N"
02 1,73
03 STO 01
04 5,86
05 STO 02
06 10,88
07 STO 03
08 50,76
09 STO 04
10 100,67
11 STO 05
12 500,68
13 STO 06
14 1000,77
15 STO 07
16 ,
17 STO 00
18 STO 08
19 "R=?"
20 AON
21 PROMPT
22 AOFF
23 LBL 01
24 7
25 STO 09
26 ATOX
27 X=0?
28 GTO 02
29 LBL 03
30 RCL IND 09
31 FRC
32 100
33 *
34 X=Y?
35 GTO 04
36 RDN
37 DSE 09
38 GTO 03
39 LBL 04
40 RCL IND 09
41 INT
42 RCL 08
43 X<Y?
44 ST- 00
45 X<Y?
46 ST- 00
47 RDN
48 ST+ 00
49 STO 08
50 GTO 01
51 LBL 02
52 "N="
53 FIX 0
54 CF 29
55 ARCL 00
56 FIX 5
57 SF 29
58 AVIEW
59 END

(120 bytes)
```

Registers, Labels and Flags

REGISTERS	COMMENTS
R00	Decimal value
R01	Value for 1 or “I”
R02	Value for 5 or “V”
R03	Value for 10 or “X”
R04	Value for 50 or “D”
R05	Value for 100 or “C”
R06	Value for 500 or “L”
R07	Value for 1000 or “M”
R08	div
R09	Previous div or counter
R10	Temporary numeral value

LABELS N2R	COMMENTS
LBL00	Loop to initialize
LBL01-09	Lookup N2R; loops R2N
LBL10	Looping through div values
LBL11	Display Roman values
LBL12	Get Roman value from reg.
LBL13	Repeat Roman value
LABELS R2N	
LBL00	Initialise registers
LBL01	Loop each Roman character
LBL02	Display decimal value
LBL03	Check match for each value
LBL04	Handle match of Roman char.
LBL05	Restart point

FLAGS	COMMENTS
-	Flags not used

Downloads

The RAW/TXT format of the program is available via the website: ROMAN (in zip file).