Charles D. Mire
artmaker43@gmail.com
Original creation on March 18, 2007

ASYMMETRIC ENCRYPTION

USING "SPREAD SPECTRUMS" OF BYTES AND RANDOM NUMBER BASES

GET: 1-, 2-, 3-, 4- or 5-Bytes converted into Random Number-Bases
PUT: 2-, 3-, 4-, 5- or 6-Bytes, respectively, as Characters in Number Base-256

ENCODING: Use Pseudo-Random number of "GET's" AND Pseudo-Random Number-Bases converted TO Base-10 and then ENCODED into Base-256 for "PUT's" into a new file.

DECODING: Use the SAME Pseudo-Random numbers (but REVERSE the "GET's" and the "PUT's") and the SAME Pseudo-Random Number-Bases; DECODED from Base-256 back TO Base-10 to be further DECODED INTO the same Pseudo-Random Number-Bases and the ORIGINAL Bytes.

Coding Conventions Used in Pseudo-Code

$	=	Tag for a string variable
#	=	Tag for a Double Precision Number (8-Bytes)
(No tag)	=	Short Integer (2-Bytes)
LOC(file)	=	Last Byte-Position that has been READ in a file
LOF(file)	=	Length of a file in Bytes
ASC	=	The ASCII Value of a Byte (0 to 255)
CHR$(Number)	=	A Character with the ASCII Value of (Number)
FIX(Number)	=	Whole Number Truncation of a Floating Point Number
/	=	Integer Division

**** NOTE: Pay CLOSE attention to the parenthesis. EXACT placement is important. ****

The number-base "ranges" have been chosen so that they will NEVER overflow into rounded-off "Scientific Notation" for any Base-10 Number conversion.

Also notice that when ENCoding, the DECoded ASCII Values of the Bytes are INcremented by +1 so that even a string of ASCII Zeros will be properly encoded. When DECoding, the ENCoded ASCII Values of the Bytes are DEcremented by -1.

In the "Pseudo-Code" illustrated on pages 4 and 5, I have only shown the implementation of pseudo-random "GETS" coupled with pseudo-random "Number-Bases". However, in my implementation of this code I have made modifications for four (4) distinct possibilities for encoding/decoding files:

1. Pseudo-Random "GETS" and Pseudo-Random "Number-Bases"
2. Pseudo-Random "GETS" and Fixed "Number-Bases"
3. Fixed "GETS" and Pseudo-Random "Number-Bases"
4. Fixed "GETS" and Fixed "Number-Bases"

I have also modified the code to allow for decoding directly to the screen (instead of to a decoded file) in each of the four possibilities. I have named my implementation of the code as: ASYMETRY.EXE

My other program, CONCEPT.EXE, is a *screen simulation program* that shows a step-by-step walk-through this concept. It **does not** read or write to any file, but it does clearly show how the math works. Screen shots are shown on pages 6 through 9 in this document. I only used a Number Base-512 so that the calculations would be easier to check. On the subject of "Dot.Number" Notation - This is nothing more than a positional notation system of representing numbers that are not Base-10 numbers. It is exactly like "Radix Notation" except that I use "Dots" instead of "Commas" as the numeral separators. For example on the page 8 screen-shot:

188.1.18.93 Base-512 = **188** * (512 ^ **3)** + **1** * (512 ^ **2)** + **18** * (512 ^ **1)** + **93** * (512 ^ **0)** = 25,233,204,417 Base-10
and
25,233,204,417 Base-10 = 5.224.4.36.193 Base-256 (In "Dot.Number" Notation) (Radix: 5,224,4,36,193)

Page 3 shows a simple tutorial on how to calculate from a Base-10 number into any other Number-Base and also shows the number 1,000 in Base-10 that has been converted into Base-30 using "Dot.Number" Notation.

Sample Number Base Conversions

Convert 100 (Base 10) to Base 2

2(100
2(50 Remainder 0 (Least Significant Remainder)
2(25 Remainder 0
2(12 Remainder 1
2(6 Remainder 0
2(3 Remainder 0
2(1 Remainder 1
2(0 Remainder 1 (Most Significant Remainder)

Therefore 100 (Base 10) = 1100100 (Base 2) = **1.1.0.0.1.0.0** (Base 2) (**"Dot.Number" Notation**)

Convert 100 (Base 10) to Base 6

6(100
6(16 Remainder 4 (Least Significant Remainder)
6(2 Remainder 4
6(0 Remainder 2 (Most Significant Remainder)

Therefore 100 (Base 10) = 244 (Base 6) = **2.4.4** (Base 6) (**"Dot.Number" Notation**)

Convert 100 (Base 10) to Base 8

8(100
8(12 Remainder 4 (Lease Significant Remainder)
8(1 Remainder 4
8(0 Remainder 1 (Most Significant Remainder)

Therefore 100 (Base 10) = 144 (Base 8) = **1.4.4** (Base 8) (**"Dot.Number" Notation**)

Convert 1000 (Base 10) to Base 30

30(1000
30(33 Remainder 10 (Least Significant Remainder)
30(1 Remainder 3
30(0 Remainder 1 (Most Significant Remainder)

Therefore 1000 (Base 10) = **1.3.10** (Base 30) (**"Dot.Number" Notation**)

ENCODE

OPEN EXISTING ReadFile in BINARY Mode
OPEN NEW WriteFile in BINARY Mode

ENCODE:

```
IF LOC(ReadFile) = LOF(ReadFile) THEN
  CLOSE
  QUIT
ENDIF
```

Choose a NEW pseudo-random amount of GetBytes# from EXISTING file: 1, 2, 3, 4, or 5 Bytes

```
IF LOF(ReadFile) - LOC(ReadFile) < GetBytes# THEN
  GetBytes# = LOF(file) - LOC(file)
ENDIF

PutBytes# = GetBytes# + 1

IF GetBytes# = 1 THEN NumberBase# "range" from 257 thru 257     (Note: ONLY used for a "leftover-byte")
IF GetBytes# = 2 THEN NumberBase# "range" from 257 thru 65534
IF GetBytes# = 3 THEN NumberBase# "range" from 257 thru 4095    The NumberBase# "ranges" have been
IF GetBytes# = 4 THEN NumberBase# "range" from 257 thru 1625    calculated so that they will NOT overflow
IF GetBytes# = 5 THEN NumberBase# "range" from 257 THRU 1023    into rounded-off "Scientific Notation".
```

Choose a NEW pseudo-random NumberBase# within the appropriate "range"

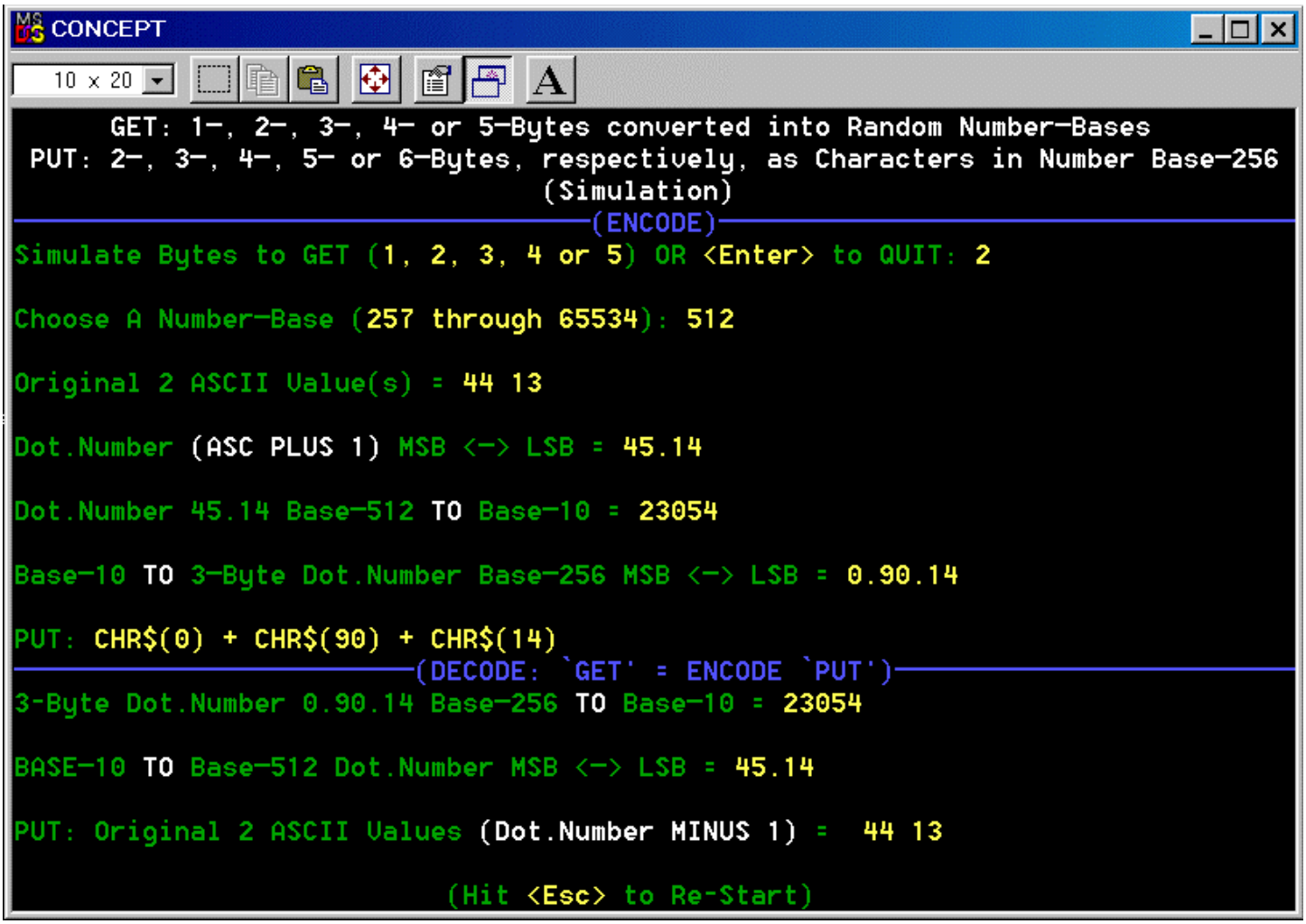
```
ReadBytes$ = STRING$(GetBytes#, 32)
GET ReadBytes$ (from EXISTING ReadFile)

FOR X# = 1 to GetBytes#
  Number# = Number# + (ASC(MID$(ReadBytes$, X#, 1)) + 1) * NumberBase# ^ (GetBytes# - X#)
NEXT X#
(Notice:  ASCII Value of ReadBytes$ are Incremented by +1 before converting to Base-10)
********** (Number# is in Base-10) **********

FOR X# = 1 TO PutBytes#
  Base256Bytes$ = CHR$(Number# - (256 * FIX(Number# / 256))) + Base256Bytes$
  Number# = FIX(Number# / 256)
NEXT X#

********** (The ASCII values of Base256Byte$ are in Base-256) **********

PUT Base256Bytes$ (into NEW Encoded WriteFile)
Base256Bytes$ = ""
Number# = 0
LOOP BACK TO ENCODE  UNTIL DONE
```

DECODE

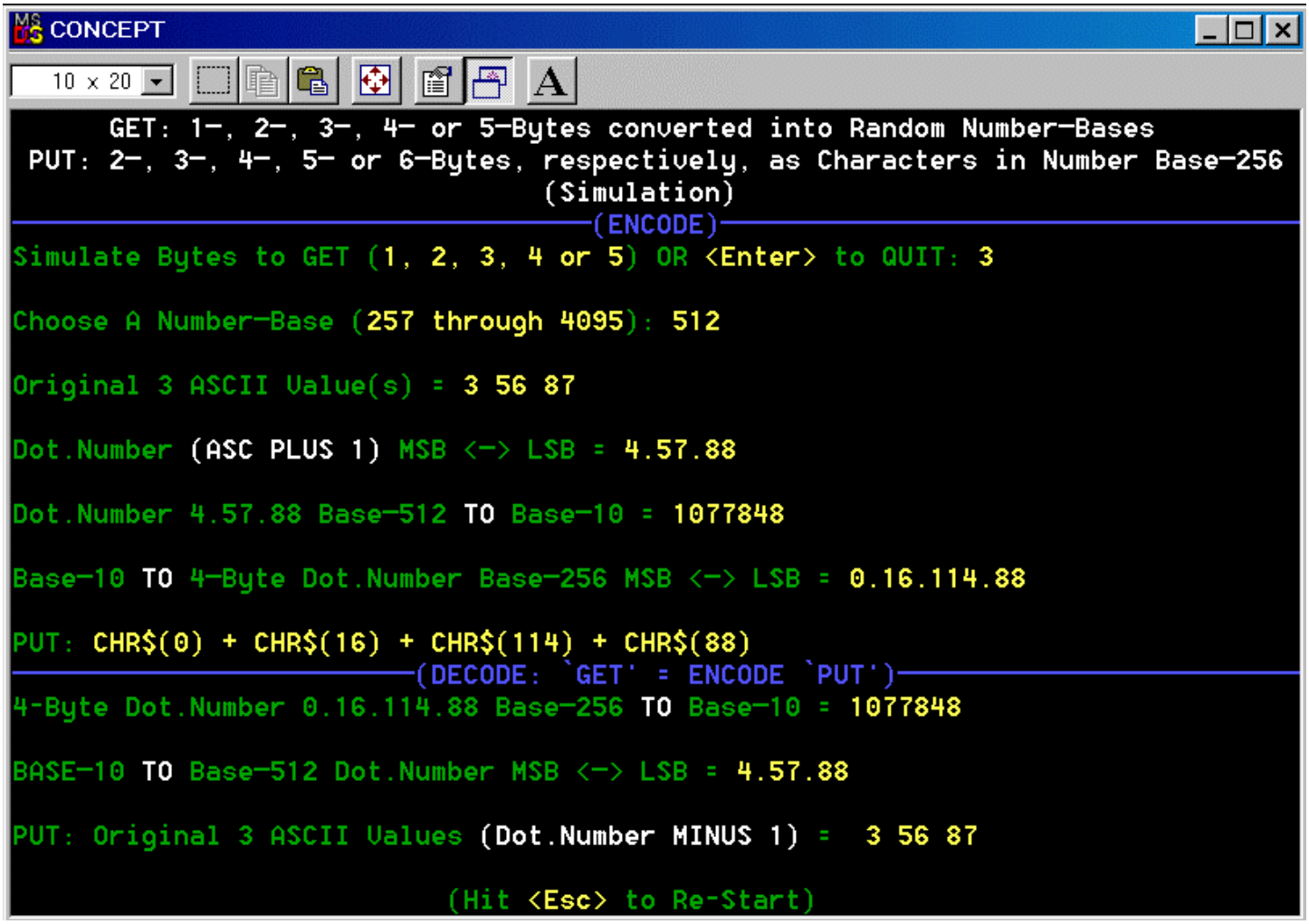
OPEN EXISTING ReadFile in BINARY Mode
OPEN NEW WriteFile in BINARY Mode

DECODE:

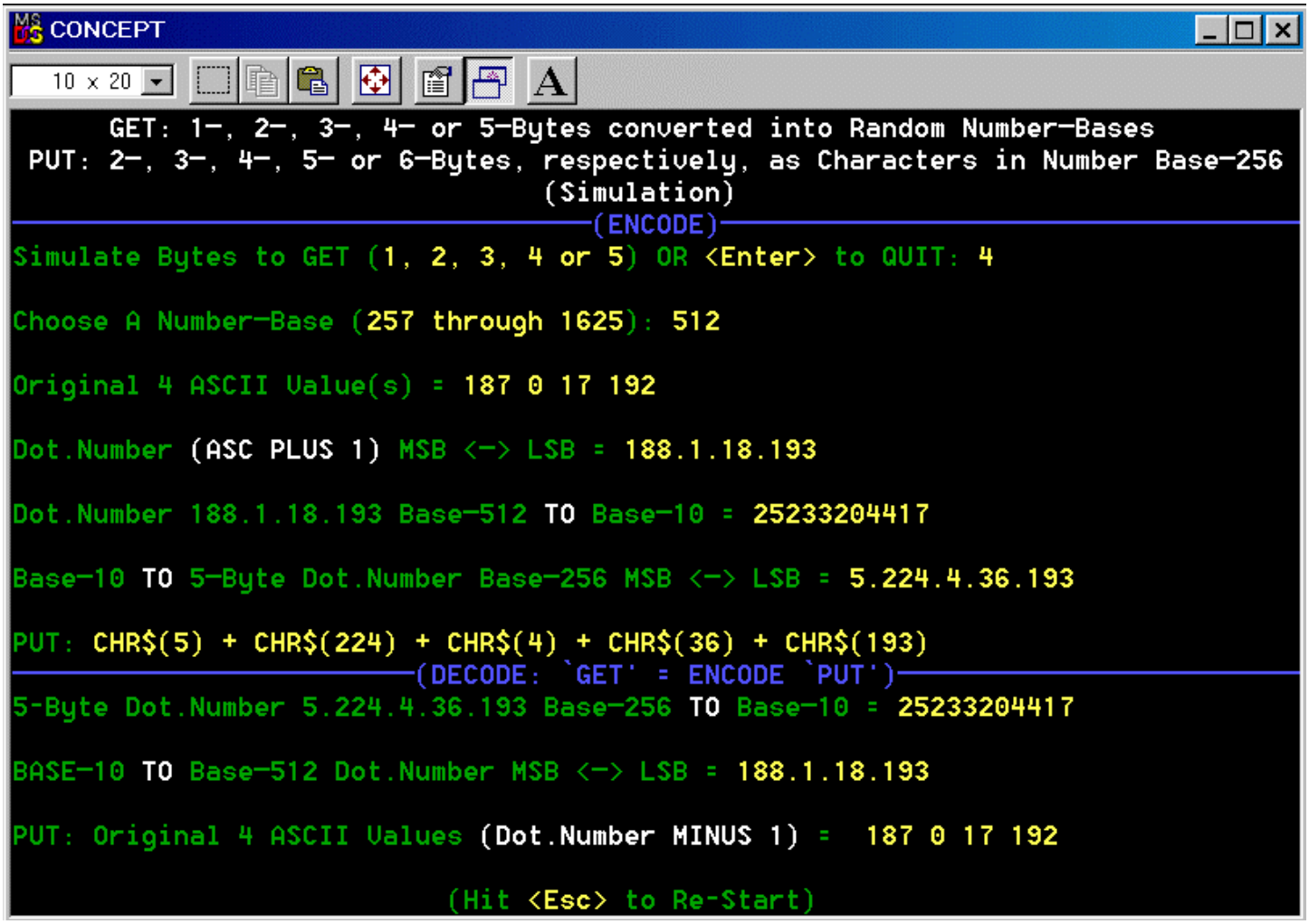
```
IF LOC(ReadFile) = LOF(ReadFile) THEN
  CLOSE
  QUIT
ENDIF

Choose SAME pseudo-random amount of PutBytes# from EXISTING file: 1, 2, 3, 4, OR 5 Bytes

IF LOF(ReadFile) - LOC(ReadFile) < GetBytes# THEN
  GetBytes# = LOF(ReadFile) - LOC(ReadFile)
ENDIF

PutBytes# = GetBytes# - 1

IF GetBytes# = 2 THEN NumberBase# "range" from 257 thru 257     (Note: ONLY used for a "leftover-byte")
IF GetBytes# = 3 THEN NumberBase# "range" from 257 thru 65534
IF GetBytes# = 4 THEN NumberBase# "range" from 257 thru 4095
IF GetBytes# = 5 THEN NumberBase# "range" from 257 thru 1625
IF GetBytes# = 6 THEN NumberBase# "range" from 257 thru 1023

Choose THE SAME NEW pseudo-random NumberBase# within the appropriate "range"

ReadBytes$ = STRING$(GetBytes#, 32)
GET ReadBytes$ (from EXISTING ReadFile)
FOR X# = GetBytes# TO 1 STEP -1
  Number# = Number# + ASC(MID$(ReadBytes$, X#, 1)) * (256 ^ (GetBytes# - X#))
NEXT X#

********** (Number# is in Base-10) **********

FOR X# = 1 TO PutBytes#
  OriginalBytes$ = CHR$(Number# - (NumberBase# * FIX(Number# / NumberBase#)) - 1) + OriginalBytes$
  Number# = FIX(Number# / NumberBase#)
NEXT X#
(Notice:  ASCII Value Decremented by -1 before converting to OriginalBytes$)
********** (OriginalBytes$ has the Original 1 to 2, 3, 4 or 5 Bytes) **********

PUT OriginalBytes$ (into a NEW Decoded WriteFile)
OriginalBytes$ = ""
Number# = 0
LOOP BACK TO DECODE  UNTIL DONE
```



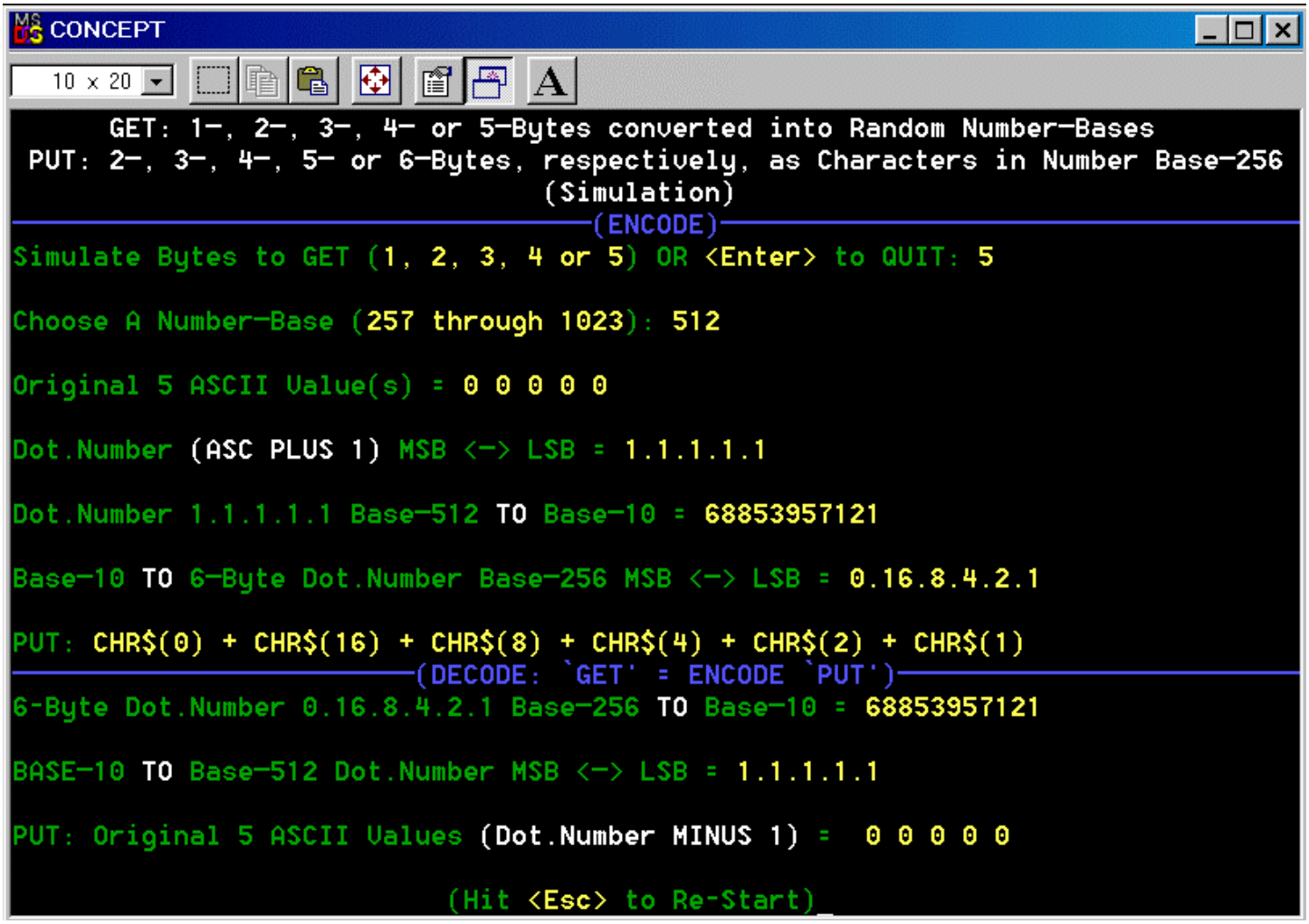
Concept Simulation of Getting 2-Bytes and Putting 3-Bytes



Concept Simulation of Getting 3-Bytes and Putting 4-Bytes



Concept Simulation of Getting 4-Bytes and Putting 5-Bytes



Concept Simulation of Getting 5-Bytes and Putting 6-Bytes