# BASIC OUTLINE

I. PRIMARY CODING GOALS

- A. Speeding Up BASIC
- B. Saving Memory
- C. BASIC Bugs

II. HOW ATARI BASIC WORKS

- A. The Token File Structure
  1. Token Output Buffer
  2. Variable Name Table
  3. Variable Value Table
  4. Tokenized Program
  5. Immediate Mode Line
  6. String Array Area
  7. Run Time Stack
- B. LOAD and SAVE BASIC Files
  1. Format Of SAVEd Disk File
  2. How The Pointers Are Used

III. ADVANCED PROGRAMMING TECHNIQUES

- A. Special Features Of BASIC And The OS
- B. Example Programs
  1. Initialize A String
  2. Delete Lines Of Code
  3. Modify String/Array Pointers
  4. Save And Retrieve BCD Numbers On Disk
  5. Name Table Modifier

# Improving Program Performance

Program performance can be improved in two ways. First the execution time can be decreased (it will run faster) and second, the amount of space required can be decreased, allowing it to use less RAM. To attain these two goals, the following lists can be used as guidelines. The methods of improvement in each list are primarily arranged in an order of decreasing effectiveness. Therefore the method at the top of a list will have more impact than one on the bottom .

Speeding Up A BASIC Program:

1. Recode - since BASIC is not a structured language, the code written in it tends to be a bit inefficient. After a lot of modification it becomes even worse. Thus, spending the time to "restructure" the code is worthwhile.

2. Put frequently called subroutines and FOR/NEXT loops at the start of the program - BASIC starts at the beginning of a program to look for a line number, so any line references near the end will take longer to reach.

3. For frequently called operations within a loop use in-line code rather than subroutines - the program speed can be improved here since BASIC spends time adding and removing entries from the run time stack.

4. Make the most frequently changing loop of a nested set the deepest - in this way the run time stack will be altered the fewest number of times.

5. Simplify floating point calculations within the loop - if a result is obtained by multiplying a constant by a counter, time could be saved by changing the operation to an add of a constant.

6. Try and set up loops as multiple statements on one line - in this way the BASIC interpreter will not have to get the next line to continue the loop.

7. Disable the screen display - if visual information is not important for a period of time, up to a 30% time savings can be made with a POKE 559,0.

8. Use assembly code - time savings can be made by encoding loops in assembler and using the USR function.

Saving Space In A BASIC Program:

1) Recode - as mentioned previously, restructuring the program will make it more efficient. It will also save space.

2) Remove remarks - remarks are stored as ATASCII data and merely take up space in the running program.

3) Replace a constant used more than twice with a variable - BASIC allocates seven bytes for a constant but only one for a variable reference, so six bytes can be saved each time a constant is replaced with a variable assigned to that constants value.

4) Initialize this variable with a read statement - a data statement is stored in ATASCII code, one byte per character, whereas an assignment statement requires seven bytes for one constant.

5) Try to convert numbers used once and twice to operations of predefined variables - an example is to define Z1 to equal 1, Z2 to equal 2, and if the number 3 is required, replace it with the expression Z1 + Z2.

6) Set frequently used line numbers (in GOSUB and GOTO) to predefined variables - if the line 100 is referenced 50 times, approximately 300 bytes can be saved by equating Z100 to 100 and referencing Z100.

7) Keep the number of variables to a minimum - each new variable entry requires 8 more bytes in the variable value table plus a few bytes for its name.

8) Clean up the value and name tables - variable entries are not deleted from the value and name tables even though all references to them are removed from the program. To delete the entries LIST the program to disk or cassette, type NEW, the ENTER the program.

9) Keep variable names as short as possible - each variable name is stored in the name table as ATASCII information. The shorter the names, the shorter the table.

10) Replace text used repeatedly with strings - on screens with a lot of text, space can be saved by assigning a string to a commonly used set of characters.

11) Initialize strings with assignment statements - an assignment of a string with data in quotes requires less space than a READ statement and a CHR$ function.

12) Concatenate lines into multiple statements - three bytes can be saved each time two lines are converted into two statements on one line.

13) Replace once used subroutines with in-line code - the GOSUB and RETURN statements waste bytes if used only once.

14) Replace numeric arrays with strings if the data values do not exceed 255 - numeric array entries require six bytes each, whereas string elements only need one.

15) Replace set color statements with POKE commands - this will save 8 bytes.

16) Use cursor controls rather than POSITION statements - the POSITION statement requires 15 bytes for the X,Y parameters whereas the cursor editing characters are one byte each.

17) Delete lines of code via program control - see the advanced programming techniques section.

18) Modify the string/array pointer to load predefined data - see the advanced programming techniques section.

19) Small assembly routines can be stored in remark statements - remarks are stored as unchanged ATASCII data.

Lane Winner
11/25/80

# BASIC BUG LIST

1) An input statement with no variable is not flagged as an error.

2) LPRINT loops cannot be stopped by hitting BREAK.

3) PRINT A=NOT B locks up the keyboard.

4) DIM L (10) generates DIM L10).

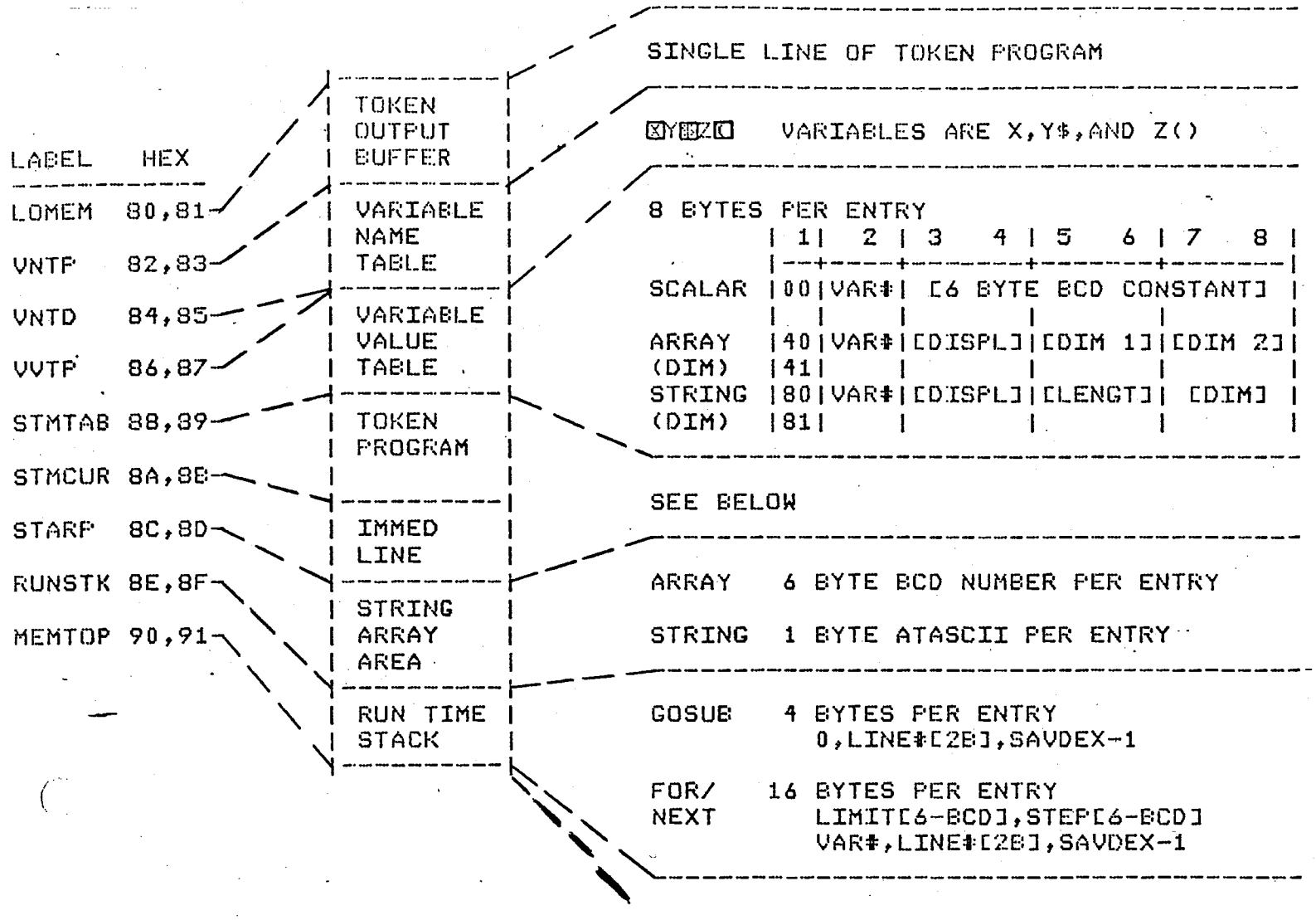
5) The following functions have wrong values:
LOG(0), CLOG(0), LOG(1), CLOG(1), most exponents.

6) Line editing problem (usually deleting lines) sometimes locks up keyboard.

7) ATN function is wrong in second BASIC cartridge.

8) Trig functions cannot evaluate N*360 in DEG or N* 3.14159 in RAD where n>=2.5E7 in second BASIC cartridge.

9) A PRINTed CTL R or CTL U character is treated as a semicolon.




EXAMPLE PROGRAM

```
10 REM TOKENS
20 FOR X=PEEK(130)+PEEK(131)*256 TO PEEK(140)+PEEK(141)*256-1
30 PRINT PEEK(X);" ";:NEXT X
```

EXAMPLE OUTPUT (PARTIALLY FORMATTED FOR READING)

```
RUN

(VNT) 216 ([X])
(VND)   0 (DUMMY)
(VVT)   0   0  65 118  51   0   0   0
(STM)  10   0  12  12   0  84  79  75  69  78  83 155 (ATASCII)
       20   0  75  75   8 128  45  70  58  14  65   1  48   0   0   0  44  37
              70  58  14  65   1  49   0   0   0  44  36  14  65   2  86   0
               0   0  25  70  58  14  65   1  64   0   0   0  44  37  70  58
              14  65   1  65   0   0   0  44  36  14  65   2  86   0   0   0
              38  14  64   1   0   0   0   0  22
       30   0  19  15  32  70  58 128  44  21  15   1  32  21  20  19   9 128
              22
(IMM)   0 128   6   6  37  22
```

NOTE - BOXED CHARACTERS APPEAR AS INVERSE VIDEO ON THE SCREEN

## COMMANDS

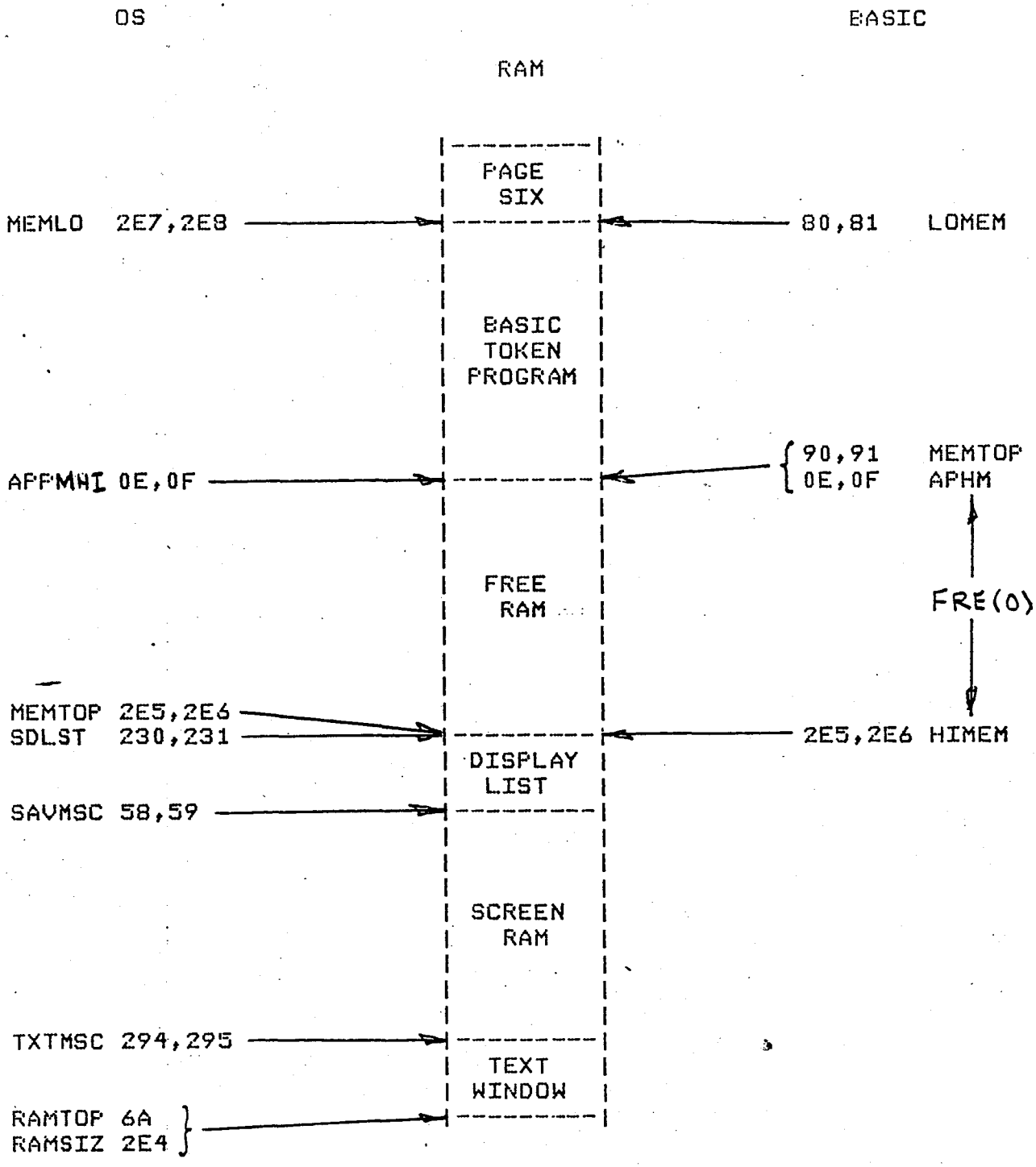
| HEX | DEC | |
|---|---|---|
| 00 | 0 | REM |
| 01 | 1 | DATA |
| 02 | 2 | INPUT |
| 03 | 3 | COLOR |
| 04 | 4 | LIST |
| 05 | 5 | ENTER |
| 06 | 6 | LET |
| 07 | 7 | IF |
| 08 | 8 | FOR |
| 09 | 9 | NEXT |
| 0A | 10 | GOTO |
| 0B | 11 | GO TO |
| 0C | 12 | GOSUB |
| 0D | 13 | TRAP |
| 0E | 14 | BYE |
| 0F | 15 | CONT |
| 10 | 16 | COM |
| 11 | 17 | CLOSE |
| 12 | 18 | CLR |
| 13 | 19 | DEG |
| 14 | 20 | DIM |
| 15 | 21 | END |
| 16 | 22 | NEW |
| 17 | 23 | OPEN |
| 18 | 24 | LOAD |
| 19 | 25 | SAVE |
| 1A | 26 | STATUS |
| 1B | 27 | NOTE |
| 1C | 28 | POINT |
| 1D | 29 | XIO |
| 1E | 30 | ON |
| 1F | 31 | POKE |
| 20 | 32 | PRINT |
| 21 | 33 | RAD |
| 22 | 34 | READ |
| 23 | 35 | RESTORE |
| 24 | 36 | RETURN |
| 25 | 37 | RUN |
| 26 | 38 | STOP |
| 27 | 39 | POP |
| 28 | 40 | ? |
| 29 | 41 | GET |
| 2A | 42 | PUT |
| 2B | 43 | GRAPHICS |
| 2C | 44 | PLOT |
| 2D | 45 | POSITION |
| 2E | 46 | DOS |
| 2F | 47 | DRAWTO |
| 30 | 48 | SETCOLOR |
| 31 | 49 | LOCATE |
| 32 | 50 | SOUND |
| 33 | 51 | LPRINT |
| 34 | 52 | CSAVE |
| 35 | 53 | CLOAD |
| 36 | 54 | [IMPLIED LET] |
| 37 | 55 | ERROR- [SYNTAX] |

## OPERATORS

| HEX | DEC | |
|---|---|---|
| 0E | 14 | [NUM CONST] |
| 0F | 15 | [STR CONST] |
| 10 | 16 | " |
| 11 | 17 | [NOT USED] |
| 12 | 18 | , |
| 13 | 19 | $ |
| 14 | 20 | : [STMT END] |
| 15 | 21 | ; |
| 16 | 22 | [LINE END] |
| 17 | 23 | GOTO |
| 18 | 24 | GOSUB |
| 19 | 25 | TO |
| 1A | 26 | STEP |
| 1B | 27 | THEN |
| 1C | 28 | # |
| 1D | 29 | <= [NUMERICS] |
| 1E | 30 | <> |
| 1F | 31 | >= |
| 20 | 32 | < |
| 21 | 33 | > |
| 22 | 34 | = |
| 23 | 35 | ^ |
| 24 | 36 | * |
| 25 | 37 | + |
| 26 | 38 | - |
| 27 | 39 | / |
| 28 | 40 | NOT |
| 29 | 41 | OR |
| 2A | 42 | AND |
| 2B | 43 | ( |
| 2C | 44 | ) |
| 2D | 45 | = [ARITHM ASSIGN] |
| 2E | 46 | = [STRING ASSIGN] |
| 2F | 47 | <= [STRINGS] |
| 30 | 48 | <> |
| 31 | 49 | >= |
| 32 | 50 | < |
| 33 | 51 | > |
| 34 | 52 | = |
| 35 | 53 | + [UNARY] |
| 36 | 54 | - |
| 37 | 55 | ( [STRING LEFT PAREN] |
| 38 | 56 | ( [ARRAY LEFT PAREN] |
| 39 | 57 | ( [DIM ARRAY LEFT PAREN] |
| 3A | 58 | ( [FUN LEFT PAREN] |
| 3B | 59 | ( [DIM STR LEFT PAREN] |
| 3C | 60 | , [ARRAY COMMA] |

## FUNCTIONS

| HEX | DEC | |
|---|---|---|
| 3D | 61 | STR$ |
| 3E | 62 | CHR$ |
| 3F | 63 | USR |
| 40 | 64 | ASC |
| 41 | 65 | VAL |
| 42 | 66 | LEN |
| 43 | 67 | ADR |
| 44 | 68 | ATN |
| 45 | 69 | COS |
| 46 | 70 | PEEK |
| 47 | 71 | SIN |
| 48 | 72 | RND |
| 49 | 73 | FRE |
| 4A | 74 | EXP |
| 4B | 75 | LOG |
| 4C | 76 | CLOG |
| 4D | 77 | SQR |
| 4E | 78 | SGN |
| 4F | 79 | ABS |
| 50 | 80 | INT |
| 51 | 81 | PADDLE |
| 52 | 82 | STICK |
| 53 | 83 | PTRIG |
| 54 | 84 | STRIG |




OS AND BASIC POINTERS (NO DOS PRESENT)

SPECIALIZED PROGRAM EXAMPLES

```
10 REM STRING INITIALIZATION
20 DIM A$(1000)
30 A$(1)="A":A$(1000)="A"
40 A$(2)=A$
```

```
10 REM DELETE LINE EXAMPLE
20 GRAPHICS 0:POSITION 2,4
30 ? 70:? 80:? 90:? "CONT"
40 POSITION 2,0
50 POKE 842,13:STOP
60 POKE 842,12
70 REM THESE LINES
80 REM WILL BE
90 REM DELETED
```

—

```
GOTO 100
10 REM STRING/ARRAY SAVE
15 REM GOTO 10 FOR FIRST RUN
20 DIM A$(10):A$="WWWWWWWWWW"
30 STARP=PEEK(140)+PEEK(141)*256
40 STARP=STARP+10
50 HI=INT(STARP/256):LO=STARP-HI*256
60 POKE 140,LO:POKE 141,HI
70 SAVE "D:STRING":STOP
100 STARP=PEEK(140)+PEEK(141)*256
110 STARP=STARP-10
120 HI=INT(STARP/256):LO=STARP-HI*256
130 POKE 140,LO:POKE 142,LO:POKE 144,LO
140 POKE 141,HI:POKE 143,HI:POKE 145,HI
150 DIM A$(10)
160 A$(10,10)="W"
170 STOP
```

```
5 REM SAVE AND RETRIEVE BCD NUMBERS ON DISK
10 DIM A(0),B$(6)
20 B$(6,6)=CHR$(32)
30 VTAB=PEEK(134)+PEEK(135)*256
40 POKE VTAB+10,0
50 OPEN #1,8,0,"D:TEST"
60 FOR C=1 TO 15:A(0)=C:? #1;B$:NEXT C
70 CLOSE #1
80 OPEN #1,4,0,"D:TEST"
90 FOR C=1 TO 15:INPUT #1,B$:? A(0):NEXT C
100 CLOSE #1:END
```

## Advanced Programming Techniques

When the fundamentals of Atari BASIC are understood some interesting applications can be written. These can be strictly BASIC operations, or they can also involve features of the operating system.

Example 1 - String Initialization - This program will set all the bytes of a string of any length to the same value. BASIC copies the first byte of the source string into the first byte of the destination string, then the second, third, and so on. By making the destination string the second byte of the source, the same character can be stored into the entire string.

Example 2 - Delete Lines Of Code - By using a feature of the operating system, a program can delete or modify lines of code within itself. The screen editor can be set to accept data from the screen without user input. Thus by first setting up the screen, positioning the cursor to the top, and then stopping the program, BASIC will be getting commands that have already been entered.

Example 3 - Saving The String/Array Area - If an array or string is always initialized to the same size and data, then an appreciable amount of program space can be saved by storing the information during the SAVE and then deleting the initialization code for the next run.

Example 4 - Save BCD Numbers To Disk - Whenever numeric data is written to a device it is sent as ATASCII information. This means the number 10 is written as an ATASCII 1 followed by a 0. This makes a mess out of fixed length records. One way to correct this is to store the six byte BCD number to disk directly by equating it to a string, and then writing that string. It can be retrieved in the same way.

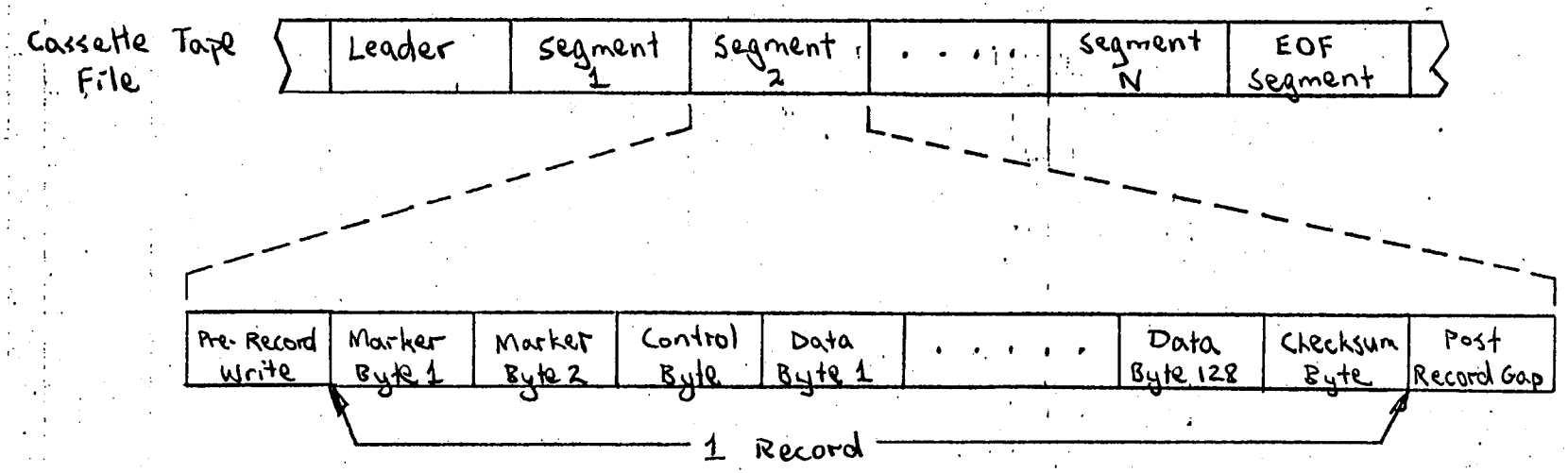
Example 5 - Player/Missile Graphics With Strings - A fast way to move player/missile graphics data is shown in this example. A dimensioned string has its string array area offset value changed to point to the P/M graphics area. Writing to this string with an assignment statement will now write data into the P/M area at assembly language rates.

```
100 REM PLAYER/MISSILE EXAMPLE
110 DIM A$(512),B$(20)
120 X=X+1:READ A:IF A<>-1 THEN B$(X,X)=CHR$(A):GOTO 120
  0 DATA 0,255,129,129,129,129,129,129,129,129,255,0,-1
2000 POKE 559,62:POKE 704,88
2020 I=PEEK(106)-16:POKE 54279,I
2030 POKE 53277,3:POKE 710,224
2040 VTAB=PEEK(134)+PEEK(135)*256
2050 ATAB=PEEK(140)+PEEK(141)*256
2060 OFFS=I*256+1024-ATAB
2070 HI=INT(OFFS/256):LO=OFFS-HI*256
2090 POKE VTAB+2,LO:POKE VTAB+3,HI
3000 Y=60:Z=100:V=1:H=1
4000 A$(Y,Y+11)=B$:POKE 53248,Z
4010 Y=Y+V:Z=Z+H
4020 IF Y>213 OR Y<33 THEN V=-V
4030 IF Z>206 OR Z<49 THEN H=-H
4420 GOTO 4000
```

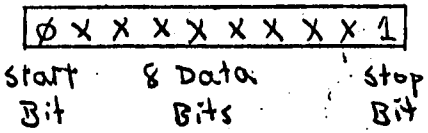
EXAMPLE 5

Tapes run at 1200 baud

# CASSETTE TAPE INFORMATION




## Byte Format




Mark Tone (1) = 5327 Hz
Space Tone (Ø) = 3995 Hz

Leader : 20 Sec Mark Tone

Pre-Record Write (PWRT):
Mark Tone

Post Record Gap (PWG):
Short: Mark, Normal: Unknown

## Inter-Record Gap Times

| | PWRT | PRG |
|---|---|---|
| Short | 0.25 Sec | 0-N Sec |
| Normal | 3 Sec | ≤1 Sec |

Marker Bytes = $55
ØIØIØIØI

## Control Byte Options

$FC Full data record (128 bytes)

$FA Partially full - count is in byte prior to checksum (data byte 128)

$FE End Of File record with 128 zero bytes

## Checksum Algorithm

Partial Sum
+ Current Byte
+ Carry
Result