BASIC COMPILERS FOR THE COMPUCOLOR - By D.R. Peel

Wouldn't you know it. Just at the time when the Compucolor seems to be winding down, not one but two Basic Compilers appear on the market. First, we're informed by Mr. Keith Ochiltree of CUVIC, Victoria, Australia, that a Basic Compiler was about to realeased for the Compucolor over there. Apparently it will be selling for about $100.00 Australian and be on an "uncopiable" disk to protect against piracy. We don't have any further information on this product, so perhaps if you're interested, contact CUVIC at the address listed in the EDITOR'S CORNER for more data.

The second compiler we do have a lot of information on including the compiler itself. Not only is the product itself superb, but the pricing will blow your socks off. It is called FASBAS and we have included the full instruction manual in this issue of FORUM on the following pages. We did this rather than trying to go into long explanations in explaining how to use it.

The following letter was received from Mr. Peter Hiner, the author of FASBAS, and I am reprinting it here because I think Peter better explains how he wishes to handle the distribution than I could.

" Dear Mr. Peel,

Thank you for publishing in FORUM the letter from Dave Thomas, describing my Basic Compiler.

I am taking the liberty of sending you the program and documentation, in the hope that you will be pleased with the performance of the compiler and that you will review it in FORUM and distribute it within your user group.

I am asking a price of $25 (US) for the compiler and documentation ordered direct from me, but I would prefer to reach agreement with user groups, for them to make and distribute their own copies. In this case I would ask user groups to send me a fee of $15 (US) per copy made.

I am well aware of the impossibility of enforcing payment of fees. This is why I am offering the compiler at such a low price ($15 compared with a typical market price of $150-$300). I hope that people will be fair to me in return.

I should like to comment on the wide readership of FORUM. I have received two enquiries from the USA and two from Australia, following the publication of Dave Thomas's letter, and that did not give information on price etc. Clearly FORUM moves far and fast.

I look forward to hearing your comments on the compiler and on my offer.

Yours Sincerely,
Peter Hiner "

Well, I can only say that I'm extremely impressed with what Peter has been able to accomplish. I wouldn't have believed that a Basic Compiler could be written for the Compucolor that would run in 32K let alone on a 16K machine.

The price is incredible value for what the program is capable of doing. I would encourage any user who orders it directly from Peter to request NO documentation, since the entire manual is published here in FORUM. The reason for this is that it cost Peter over $5.00 in postage to mail the compiler to me plus the disk. Give him a break and leave him something extra for this fantastic programming effort.

The following pages are the actual documentation for FASBAS. They will give you all the information that you could possibly require about the program. FASBAS needs the following reqirements to run and may be ordered directly from Peter at the address listed below.

FASBAS

MINIMUM REQUIRMENTS: 16K MEMORY
SINGLE DISK DRIVE
V6.78 or V8.79 BASIC

MR. PETER HINER
11, PENNY CROFT
HARPENDEN
HERTS, AL5 2PD
ENGLAND

TELEPHONE 05827 64872

FASBAS

1. INTRODUCTION

FASBAS (Fast Basic) is a program which compiles Basic programs to make them run faster. It produces output to disk in a modified form of Assembly Language, for subsequent assembly into machine code using the FBASM assembler program.

After compilation and assembly, Basic programs will run up to 5 times faster, though the increase in speed will vary considerably, depending on the contents of the original Basic program. FASBAS interprets the Basic program and generates addresses for fast access to variables, subroutines, etc., but it uses the same ROM subroutines as Basic for trigonometric and other functions. So a minimum speed increase of 50% is almost certain, but further speed increases will depend on the proportion of time used by the Basic program for the fundamental jobs of interpretation and searching. Many articles have been written about how to speed up Basic by putting common subroutines at the beginning, by declaring common variables early in the program, and by using variables to replace constants. These particular techniques will speed up a Basic program but will not affect the speed of a compiled program. On the other hand, normal good practices (such as minimising activities within a FOR...NEXT loop) will speed up both.

On the subject of speed, it is interesting to consider a real example. In the September, 1981 issue of Data Chip, Ken Jenkins published a Basic program for comparing three routines for sorting variables, using the internal clock to measure time taken. I compiled this program and the table shows time taken before and after compilation. In every case the number of items sorted was 100.

Type of sort routine	Random data		Already in order		Reverse order	
	Basic	Compiled	Basic	Compiled	Basic	Compiled
Bubblesort	253	70	2	1	229	83
Shellsort	58	14	11	3	28	6
Quicksort	21	5	12	3	14	3

Times are in seconds, and exlude fractions of a second, so comparisons for the shorter times are not accurate. This program is quite a favourable test for FASBAS, as it makes heavy use of single dimension numerical arrays, which are given special treatment by FASBAS. A program using two-dimensional arrays would not have shown such good results.

A demonstration program SNAKE is included on the disk for practice in the FASBAS operating procedures. In Basic the SNAKE is so slow that you should beat it every time, but after compilation the increase in speed makes it a much more challenging game.

The price that has to be paid for speed is program size. The final compiled version will be about twice the size of the original Basic program, with a minimum size of about 750 bytes (containing a run time library of routines). This is not likely to be a practical problem, because there is a more severe limitation on the maximum size of Basic program that can be compiled. FASBAS generates a tokenised Assembly language file (SRC file) as an intermediate stage, and for a large program the SRC file will be 4 or 5 times the size of the Basic program. Therefore, due to the limited storage capacity of a disk, the maximum size of Basic program (excluding REM statements) that can be compiled is effectively about 8 Kbytes with one disk drive (or 12 K with two drives).

This problem can sometimes be overcome by splitting out part of the Basic program (e.g. instructions for a game) and then chaining the two parts of the program.

A lot of care has been taken to ensure that FASBAS is capable of compiling every possible combination of instructions that programmers might use in writing Basic programs. There are a few limitations imposed (which are detailed later), but within these limits there should be no problems. However, it would be a brave man who declared his program to be free of bugs, particularly a program of this complexity. So please let me know if you hit problems.

2. DEMONSTRATION PROGRAM

The FASBAS disk contains a Basic game SNAKE.

I recommend that you try compiling this first, to see what FASBAS can do to speed things up and to gain confidence in how compilation ought to work before you try something more complicated.

1. Put the FASBAS disk in the default drive (say drive 0)
2. Enter Escape D (for FCS mode) and RUN FASBAS
3. In response to 'Select Version', just hit RETURN
4. Leave the FASBAS disk in the drive
5. In response to FILESPEC >, enter SNAKE
6. When compilation is complete, control returns to FCS mode
7. Enter RUN FBASM
8. In response to the prompt, enter ASM SNAKE to 0:
9. The last address indicated by the assembler should be 98CB
10. Enter LOAD SNAKE
11. Enter SAVE SNAKE. PRG 82A0 - 98CB

Now you can test the difference between the Basic and compiled versions.

I suggest you try the Basic version first (you will need to enter Escape W to initialise Basic).

When you have beaten the snake, enter Escape D RUN SNAKE, and see if you can still win.

3. INSTRUCTIONS FOR USE OF FASBAS

3.1 Debugging Basic

First of all, debug your Basic program thoroughly, so that it runs consistently, without any error messages. Compiled programs save time by omitting many of the error checks provided by Basic, so programs with errors in them may crash in a cloud of dots and lines, produce a meaningless error message or simply disappear into thin air.

3.2 Check for unacceptable Basic statements

FASBAS will accept nearly all the Basic instruction set, but there are a few exceptions, which are detailed later. You may prefer to let FASBAS flag up unacceptable statements, rather than scan through the Basic program yourself. However, I do recommend that you check for premature exit from FOR...NEXT loops, as this will not cause FASBAS to indicate an error but may cause unexpected results when you run the compiled program. This is the most common problem encountered (see section 5.1 for more details).

3.3 Running FASBAS

Load disk containing FASBAS into the default drive, enter FCS mode (Escape D) and key in RUN FASBAS. (Note that FASBAS will automatically configure itself to run on V6.78 or V8.79 machines.)

FASBAS loads to address 82A0 and you can not re-run it using Escape T etc. FASBAS uses overlay techniques to reduce memory requirements, and you must always start from scratch.

3.4 Select target machine

It is not feasible to generate compiled programs that will automatically run on either V6.78 or V8.79 machines, as that would increase the size of the final program considerably. However, FASBAS allows you to select the version on which the compiled program is to run.

To compile for the same version of machine as you are using, enter 0 (or Return).

To compile for the other version of machine, enter 1.

FASBAS will now load the appropriate table of addresses and routines.

3.5 Load disk containing Basic program

With a single disk drive, there are no options, and FASBAS will output the Assembly language file to the disk containing the Basic program (so make sure there is plenty of space).

With more than one disk drive you can select the drives for Basic input and Assembly language output (see next section).

3.6 Response to FILE SPEC >

The minimum response required is the name of the Basic program to be compiled (e.g. TEST). This will cause the Basic program to be read from the disk in the default drive and the compiled version to be written onto the same disk (as an SRC file with the same name as the Basic program).

The maximum response allows input and output drives, program names and versions to be specified. For example:

CD0 : TEST. BAS; 01 TO CD1 : DOODLE. SRC; 02

Any legitimate variation or subset of the above can be entered (but if you insist on making the output file type anything other than SRC, remember to specify file type during assembly).

3.7 Compilation

FASBAS makes two passes through the Basic program. The first pass is quite fast since FASBAS is only looking for DATA and DIM statements. During the second pass the compiled program is generated and written to disk.

The Basic program line numbers are displayed, so that you can see progress. If FASBAS detects any Basic statements that it can not accept, it will display an error message against the offending line number, and will continue compiling at the next line.

At the end, FASBAS exits to FCS mode.

3.8 Correction of errors

Inspect the Basic program line number(s) indicated as containing an error, and check for functions not accepted by FASBAS. As well as obvious items such as DEF FNA(X), FASBAS will detect undimensioned numerical arrays with more than one dimension, and unsupported FILE statements (e.g. FILE "T").

Some of the more subtle points (such as RESTORE to a line number not containing data) will not cause FASBAS to generate an error message. In the above example of RESTORE, an error will be indicated during Assembly, since there will be a pointer to a non-existent data reference (e.g. (Shift)XHD200 means LXI H,D200 and refers to data supposedly stored in line 200). Some errors may not cause any indication at all until the compiled program is run. For example, jumping out of FOR...NEXT loops prematurely may even work satisfactorily when the compiled program is run (but it will probably cause the program to give peculiar results or to crash).

A particular point to watch for is the use of POKE to load machine code programs in what you believe to be spare RAM (for subsequent use by the CALL function). Since the compiled version of a program will be much larger than the original Basic program, the addresses being POKED may no longer be spare. The compiled program uses memory space in a very similar way to Basic, that is to say:-

82A0 Start of compiled program (which includes space for storing variables and single dimension numerical arrays).

VVVV { End of compiled program
Start of storage space for string pointers, multi-dimension numerical arrays and string array pointers

WWWW { End of storage space
Start of spare space

XXXX { End of spare space
Start of stack space

YYYY { End of stack space
Start of string manipulation space (50 bytes unless altered by a CLEAR statement).

ZZZZ { End of string manipulation space
End of available memory (unless memory space reserved during Basic initialisation)

As you can see, the start and end of spare space are not very well defined. The start depends on size of program and storage space for strings and arrays, and the end depends on both the amount of space allocated for string manipulation and how big the stack grows during program run. Note that the stack grows from end towards start, and although it should decrease as often as it grows, so that it finishes back at the end point, it will reach peak size during deeply nested subroutines.

The above remarks apply equally to Basic and compiled programs. The safe way to obtain spare space is to reserve it after the end of string manipulation space. This can be done by initialising Basic (Escape W) and, when asked 'Maximum RAM available?' entering a number which is n bytes less than the actual memory available. That is not practicable, since you would have to remember to do it every time you ran the program.

The same result can be obtained by POKE and CLEAR instructions at the start of the Basic program. Here is an example for 16K and 32K machines (values for (a) and (b) can be found from the table):

```
0   POKE 32940, (a) : POKE 32941, (b) : CLEAR 50 (or more)
```

Number of bytes to be reserved	16K Machine		32K Machine	
	(a)	(b)	(a)	(b)
128	127	191	127	255
256	255	190	255	254
512	255	189	255	253

This is too complicated for many people and so the usual practice is to use spare space after the Basic program, variables and arrays, allowing generous margins for errors in estimation, and then to run the program and see if it works. The same procedure can be applied to compiled programs, allowing twice as much space as for the original Basic program.

3.9 Editing

FASBAS produces a modified form of assembly language output (using single byte tokens to save space). The SRC file, which is generated by FASBAS, can be edited using the Com-tronics CTE program (version 2.23 on disk). For some unknown reason the ISC editor (EDT) loses the last block of the SRC file. No doubt this is caused by a deficiency in the way FASBAS writes the SRC file to disk, but I have not been able to discover the reason. The CTE editor increases the size of the SRC file by automatically inserting carriage return characters (which have been omitted by FASBAS to save space), so you may not be able to edit long programs.

Although there is no normal reason for editing the SRC file the identities of tokens are given in the following table. In each case the token is a letter together with shift key operated. Note also that FBASM assumes that all numbers are hexadecimal and the H postscript is always omitted, as are spaces and commas.

Shifted Letter	Hex Value	Significance
A	61	LDA
B	62	STA
C	63	CALL
D	64	DCX
E	65	DAD
F	66	CPI
G	67	XCHG
H	68	PUSH
I	69	INX
J	6A	JMP
K	6B	ANA
L	6C	LHLD
M	6D	MOV
N	6E	DB
O	6F	DS
P	70	POP
Q	71	XRA
R	72	RET
S	73	SHLD
T	74	XTHL
U	75	INR
V	76	MVI
W	77	DCR
X	78	LXI
Y	79	JNZ
Z	7A	JZ

3.10 Assembly

The FBASM assembler program is derived from the ISC Assembler and is used in the same way as the ISC Assembler is used for programs in normal assembly language. The size of assembled program will be between a third and a half of the size of the SRC file (in other words, about twice the size of the original Basic program). With a large program and a single disk drive you may need to delete the original Basic program from disk to leave enough space for assembly.

In response to the prompt (>), enter ASM.... TO 0: (or TO 1:). Entering ASM.... only will cause assembly without writing to disk.

If FBASM finds an error it will stop (hit return to continue assembly). You should assume in the first instance that errors result from some unacceptable practice in the original Basic program (e.g. RESTORE to a line number not containing data, or a redundant line containing instruction to GOTO or GOSUB a line number which does not exist). You can identify the line number of the Basic program from the reference labels in the SRC file (e.g. L999 : precedes the compiled contents of line 999. Ignore auxiliary references such as L999A: or L999AO:).

If you believe an error has actually been caused by a bug in FASBAS, then please let me know.

3.11 Running the compiled program

At last the fruits of your labour will be revealed. The program can be run as normal for LDA programs, or you can load it and then save it again as a PRG.

At the end of a program run, control will return to Basic, with a READY message. Programs can be interrupted only by using the RESET key. To run a program again, use ESC ^.

3.12 Changing an LDA to a PRG type program

The FBASM program produces an LDA file which can be run by entering (in FCS mode):-

```
RUN TEST. LDA
```

The only drawback is that LDA files are slow to load, and therefore it is preferable to convert them to PRG. This can not be done by simply renaming the file in the Directory, as LDA files contain extra information (load addresses for each block of data) which must be removed.

The simplest way requires that you note the last memory address displayed during the FBASM run. If this last address were for example 9872, then the method would be:-

```
FCS > LOAD TEST

FCS > SAVE TEST. PRG 82AO - 9872
```

(Note that the start address for compiled programs is always 82AO)

If you forgot to note the last address during the FBASM run, you can carry out assembly again (omitting the instruction 'TO O:' or 'TO 1:', as the LDA file has already been written to disk).

4. BASIC STATEMENTS ACCEPTED FREELY BY BASIC

The following categories of Basic statement may be used without any abnormal limitation:-

Multiple statements per line

Equivalence statements (e.g. A = 1, B = C)

Mathematics (e.g. B = A + (2/3^ 2) - 32E2)

Logical connectives (e.g. C = X OR 3)

Trigonometric and other functions (e.g. COS(X),ABS(X)etc)

PEEK and POKE (e.g. PEEK (A + B), POKE X,Y)

INP and OUT (e.g. INP(X), OUT X,Y)

PRINT (e.g. PRINT TAB(X); A$; "OK"; A+B,Y)

PLOT (e.g. PLOT 12,3,X,Y)

IF...THEN... and IF... GOTO ... (e.g. IF X THEN 200)

GOTO and GOSUB

REM

END (Restart program by using Escape^)

ON... GOTO... and ON... GOSUB

FRE(X) and FRE(X$)

WAIT X,Y and WAIT X,Y,Z

Strings and functions (e.g. A$, CHR$(X),MID$(A$,B,C))

String arrays (any number of dimensions)

CLEAR (e.g. CLEAR, CLEAR 500)

CALL (e.g. Y = CALL(X))

INPUT (variables, arrays, strings, single or multiple)

This list contains practically everything and it is much easier to look at the lists of statements accepted with limitations or not accepted at all.

5. BASIC STATEMENTS ACCEPTED WITH LIMITATIONS

The following categories of statement have some limitation imposed on their use:-

FOR... NEXT loops

DATA, READ, RESTORE

NUMERICAL ARRAYS

DIM statements

LOAD, RUN

FILE, GET, PUT

5.1 FOR....NEXT loops

All normal FOR....NEXT loop statements are permitted, except for premature exit from loops. This is not good practice but is tolerated by Basic, although in some cases it could cause the stack to grow so large that it overwrites part of the program. Type b) below will cause compiled programs to loop back to the wrong place.

Premature exit can occur in three forms:

a) Straightforward jump out of a loop, e.g.:

```
10 FOR X = 1 TO 10 : READ A(X)
20 IF A(X) = 0 THEN 200
30 NEXT
```

This should be rewritten as follows:-

```
10 FOR X = 1 TO 10 : READ A(X)
20 IF A(X) = 0 THEN X = 10 : NEXT : GOTO 200
30 NEXT
```

b) Skipping part of the inner of two nested loops, by using a conditional NEXT to go back to the outer loop e.g. :

```
110 FOR A = 1 TO 3 : FOR B = 1 TO 10
120 READ X(A,B) : IF X (A,B) = 0 THEN NEXT A
130 PRINT X(A,B) : NEXT B,A
140 ......
```

This form of premature exit will certainly not be compiled correctly and it must be rewritten to close the loop, e.g. :

```
110 FOR A = 1 TO 3 : FOR B = 1 TO 10
120 READ X(A,B) : IF X(A,B) = 0 THEN B = 10: NEXT B,A: GOTO 140
130 PRINT X(A,B) : NEXT B,A
140 .......
```

c) Sometimes the value of the variable controlling a FOR ....NEXT loop must be preserved at exit as it is used afterwards, and in this case setting the variable to its final value will give the wrong results. So a new variable must be used to store the exit value temporarily while the loop is closed, e.g. :-

```
50 FOR X = 1 TO 10
60 IF A(X) = 0 THEN 80
70 NEXT
80 PRINT X
```

This could be rewritten as:

```
50 FOR X = 1 TO 10
60 IFA(X) = 0 THEN Y =X : X =10 :NEXT:X = Y:GOTO 80
70 NEXT
80 PRINT X
```

Unfortunately some Basic programs are so badly structured, with jumps out of a loop and back in again, that it is difficult to tell whether there has been a premature exit or not. A useful method of finding instances of type b) above is to edit the Basic program, changing suspected statements like NEXT X to simply NEXT, and then running the Basic program to see if it still works. It is best not to change too many at once, or you will not be able to work out where the problems are. This technique will not find the cases where extra bytes are left on stack, but these may not be critical (and Basic can not stop the stack growing in such cases either).

5.2 DATA, READ, RESTORE

These statements can be used freely except that if the statement RESTORE to a line number (e.g. RESTORE 200) is used, then the referenced line must contain some DATA. (Basic allows RESTORE 200 to cause reading of data from the next DATA statement at or after line 200.)

This error will not be found by FASBAS itself, but will cause an error indication during assembly, due to reference to non-existent label D.....

5.3 Numerical Arrays

Numerical arrays with two or more dimensions (e.g.A(1,2), B(1,2,3) etc) are handled exactly the same as in Basic, using most of the same access routines, and as a result they do not give much increase in speed. Single dimension numerical arrays such as X(3) are treated as a special case to reduce access time, but this does cause slight complications.

Essentially all numerical arrays can be used freely, except that multi-dimension arrays must be declared using a DIM statement (for single dimension arrays the DIM statement is optional).

Failure to include a DIM statement for a multi-dimension numerical array will cause FASBAS to generate an error message against every line containing the array.

(Note that no such limitations apply to string arrays, which are all handled as in Basic.)

A second peculiarity of numerical arrays is that single dimension arrays can not be loaded using the LOAD"....ARY" statement. This can be fiddled by turning them into two dimensional arrays with one dimension set to zero (e.g. DIM A(20,0)), with consequent loss of speed.

5.4 DIM Statements

Dimensioning of multi-dimension numerical arrays is described in section 5.3.
Dynamic dimensioning, by using variables instead of constants (e.g. DIM A$(X,Y)), is not allowed, and will cause an error indication from FASBAS. This is because when running the compiled program, variable X and Y would be found to have value zero, and therefore a BS ERROR would result if this were not corrected.

5.5 LOAD and RUN

The LOAD statement can be used freely, with the sole exception that LOAD....ARY can be used only for multi-dimension arrays (see section 5.3). This is because compiled programs use a different method for accessing single dimension arrays.

The RUN statement can be used by itself or following a LOAD statement in the same line. Used by itself, the RUN statement is taken as an instruction to run the compiled program from the beginning, and any attempt to specify a line number will be ignored (e.g. IF A$ = "Y" THEN RUN 20 will cause the program to run from the beginning rather than from line 20).

When following a LOAD statement, RUN is assumed to be an instruction to run a Basic program that has been loaded, and in this case a line number can be specified. The following example will work the same as in Basic:

```
1500  IF A$ = "N" THEN LOAD "MENU": RUN 100
```

Note that LOAD "MENU": GOTO 100 would probably not give the results expected, since this technique is normally used in Basic when it is required to carry forward variable values from one program to another, and a compiled program does not store variables in a suitable location or format for subsequent use by Basic.

Compiled programs can also be chained, but not using the LOAD statement. For these you must use PLOT 27,4 : PRINT "RUN TEST. LDA" (or "RUN TEST" for PRG type programs).

5.6 FILE, GET, PUT

FASBAS does not accept FILE "A", "T" or "E" statements, and will give an error indication. FILE "N", "R", "C", and "D" statements, GET and PUT can be used freely without any limitations. However, with a large program you might possibly find that, because the compiled program is larger than the original, the memory space available for file buffers is not sufficient.

6. STATEMENTS NOT ACCEPTED BY FASBAS

The following statements are not accepted by FASBAS and will cause an ERROR indication.

DEF
FN
LIST
SAVE
CONT