TANSOFT 2 PASS ASSEMBLER.

CONTENTS.

PART 1 - INSTALLATION.

1. The 2732 EPROM should be inserted in socket J2 of TANEX. Remember to orientate it correctly.

2. If the Microtanic EPROM extension board is being used then insert it in the socket for location $C000. This is the top left hand socket with the board orientated with the preset switches at the top right.Pin 1 should be facing left.Remember to set the preset switches correctly, either for 1 2732 and 2 2716s or, for 2 2732s.

PART 2 - WHAT IS A 2 PASS ASSEMBLER ?

1. This part is for those of you who have perhaps dabbled with the XBUG line assembler and are not sure how a 2 pass assembler works or what its advantages are. Some knowledge of assembler code formats as used in XBUG is assumed.This part won't tell you how to write assembly code programs,you'll need a book for this. There are plenty around, any on the 6502 should do although there are sometimes slight differences in the formats of the assembly code.

2. So,first for a couple of those 'Jargon' words which seem to riddle the computer industry - source code and object code. Source code is what you actually type in, e.g. the characters LDA #$0. Object code is the machine code produced from this - A900 (hex) - easy isn't it! In the case of the XBUG line assembler, what you type in (the source code), is lost after it disappears over the top of your TV, only the object (machine) code

is placed in store - i.e. A900 (hex) in the example above. It is of course the object code that you run or execute.

3. The first major difference of a 2 pass assembler is that the source code is kept. Any decent assembler (and of course this is!) will provide facilities to edit the source, dump it to tape, read it back and generally muck about with it. In this respect it is similar to maintaining and editing BASIC code. The object code which the assembler produces is placed in store (as XBUG does), so the normal XBUG cassette dump, examine and fetch commands are used to handle this.

4. So you pump source into the assembler and get runnable object out the other end, let's now see what goes on in between. Consider the following simple source code subroutine to clear the screen:-

```
CLEAR  LDX #$0          Set reg X to 0.
       LDA #$20         A to space
LOOP   STA $200,X       Store space in top half of screen
       STA $300,X       and in bottom half.
       INX              Next location.
       BNE LOOP         Finished? if not do next.
       RTS              Exit from SR.
```

You'll notice the use of the word LOOP by the branch instr. This is referring to the STA $200,X instruction. To call this subroutine you would code JSR CLEAR. LOOP and CLEAR are what are known as labels or symbols (take your pick). They are similar in some respects to BASIC line numbers but beware drawing too many conclusions from this analagy - labels can be moved around at will to reference any instruction (or location in store).

5. Notice that there is nothing in the above code which tells you where the resultant object code is going to be in store. You tell the assembler where to put it (exuse me) at assembly time (when you pump the source in). Relocatable code! Why 2 passes? On the first pass the assembler notes where you have told it to put the object code and then proceeds to calculate the addresses of all the labels appearing on the left hand side. It needs to make one complete pass of the source code to do this. Then its back to the beginning for a second pass when it will assemble the instructions (actually generate the object code) replacing all labels used in instructions with the correct address. So suppose in our CLEAR subroutine above we decide to assemble it at $400. After the first pass the assembler will have worked out that CLEAR is at $400 and LOOP at $404. So on the second pass it can calculate the offset for the branch instruction and if we wish to call the subroutine it will know that CLEAR is at $400.

6. This means that you can now insert those odd instructions that always seem to get overlooked directly into the source and simply re-assemble. The assembler will take care of all the address recalculations for you. No more of that 'shall I rewrite or put in another messy patch?'. If you've written any machine code, you'll know the feeling!

7. There are 5 instructions which you will not have come accross in XBUG. These are BYT, WOR, EQU, EPZ and ORG. BYT and WOR are just ways of defining data, e.g.

```
DATA     BYT $A,43,'C
```

will generate hex 0A, decimal 43 (= hex 2B) and character C (= hex 43) in locations DATA, DATA+1 and DATA+2 (wherever that might be!). WOR does the same kind of thing for 2 byte constants. It generates the LOW byte first and the HIGH byte second, so that:-

```
ADDR     WOR $1234
```

gives hex 34 in ADDR and hex 12 in ADDR+1.

9. EQU,EPZ and ORG are rather different in that they do not cause any object code to be generated. They are strictly directives to the assembler. Let's recode our CLEAR subroutine using them:-

```
SCRTOP EQU $200
SCRBOT EQU $300
CLEAR  LDX #$0
       LDA #'
LOOP   STA SCRTOP,X
       STA SCRBOT,X
       INX
       BNE LOOP
       RTS
       ORG *+256
       JSR CLEAR
```

EQU stands for equate and tells the assembler to use address $200 whenever it encounters the label SCRTOP (and $300 for SCRBOT). EPZ (Equate Page Zero) does the same thing for Page 0 addresses. If you use EQU and EPZ, it is a very good idea to put them at the front of the program to aid legibility (try finding them in a 40 page listing!). EPZ must be declared before its label is used anyway (there are no other restrictions on the order of things). We could have referred to SCRBOT as SCRTOP+256 or SCRTOP+$100 (but it's a bit silly).

10. The * in the ORG directive acts like a label and means 'this address'. 'This address' is 14 bytes (decimal) on from the beginning of the proram in our example. What the ORG tells the assembler is to now start assembling at 'this address' plus 256 (decimal) bytes on, i.e. location 270. ORG can also be used with a fixed address, e.g. ORG $500 but then you will not be able to relocate your program without changing the ORG address.Note that * can be used with any instruction so that BNE *+4 means branch on equal to this address plus 4 bytes (i.e. 4 bytes on).

11. Before you plunge off into all that technical stuff in Part 3, a few words on how to organize your source and object in store.I'll assume you only have Microtan and Tanex and therefore 7K bytes available. There are three portions of store that you need to keep your eye on, these are:-
a) that holding the source code,
b) that which is going to hold the object code,
c) that holding the assembler's symbol table - you never touch
this.
The placement of all three is under your control .For smallish programs the default values should be fine,these being:-

start	end	
$400	?	source
$1400	?	object
?	$1FFF	symbol table

Note that the symbol table is built backwards through store by the assembler,so you have to watch that it does'nt bump into the object code growing forwards. Now, whenever you edit or fetch source code, the assembler will tell you how much space it's occupying (the source end address). Whenever you do a pass 1 assembly, it will tell you the start address of the symbol table it's just built. Now if you do a pass 2 assembly with the object code generation switched off (option N - see that technical stuff in Part 3),you can get a full vet of your source code and see how big the object is going to be (the object end).At this point it is a good idea to dump your program to tape,you don't want your new bug ridden program to stomp all over that carefully typed source without keeping a copy. If there is enough room, all well and good, do a pass 2 again and you're away. If not then you can assemble direct from the source that was dumped to tape. In this way you can assemble programs to fill the whole store less that required to hold the symbol table. Remember to do a pass 1 again if you alter the object start location, otherwise all those label addresses will be wrong.

12. If you want to play it safe (at the expense of flexibility), you can shuffle the whole lot around:-

?	$5FF	labels
$600	?	source
?	?	object

In this way at least the labels won't stomp over your source but labels and source are in fixed sections. It's up to you.

13. If you have Tanram then put the source there and protect it using the paging system (lovely!).

14. Well good luck with Part 3, I sincerely hope that the above has given you sufficient background to forge (well crawl) ahead. The assembler provides a lot of user control. Whilst this is great for us buffs, it can cause confusion to a newcomer. Good luck - and enjoy yourself!

PART 2 - ALL THE TECHNICAL STUFF.

General.

1. This is a 2 pass assembler with full facilities for source code editing and maintenance of source code cassette files. Assembly may be performed from source held in store or on tape. It includes options to supress output of object, to list and/or print the assembly listing or labels, to dump or assemble parts of a program and to assemble code for EPROM location.

Entering the assembler.

2. Entry to the assembler is at $C000. On entry, the message START=C? is displayed. Key C CR for a cold start or Just CR for warm start. For a cold start, the message PRINTER? is displayed. If you have a printer connected via the new TANBUG monitor, then initialise it using Control P or V and key CR. The printer will remain inactive until the P (printer) option is exercised (see para 312. If, however, you wish to use your own routine, enter its address in hex followed by CR. It will be called when the print option is set with each character to be printed in the Accumalator. End of line is indicated by CR ($0D). The next message displayed is BLOCK GAP? This enables the inter block gap used when dumping source code to be altered. This is further explained in para 21. It will usually only be necessary to key CR to set the default value of $F.

3. The major user parameters should now be displayed. These remain on the screen during all assembler operations. Their general meanings are given below (detailed explantions are given in the appropiate section dealing with each command).

Action - 2 character command (e.g. ES-edit source)
Name - Used to name source files, up to 6 chars.
#Lin st - Line start number .All source lines are numbered. This gives the start line no. for assembly or dumping
#Lin end - Gives the line no. following the end of the current source line.Updated by the editor and fetch file routines.
*Sce st - Gives the start address in store (hex) for the storage of source code.
Sce end - Gives the end address +1 of the current source code. This parameter is updated by the editor and fetch fil routines.
*Obj st - Gives the start address (hex) for the storage of the object (assembled) code.
Obj end - Gives the end address +1 of the object code. Updated by the assembler.
Sym st - Gives the address of the start of the symbol table.It is updated by the assembler.
*Sym end - Gives the end address for storage of the assembler's symbol table. User specified because the symbol table is built backwards through store.

Those parameters marked with * are user supplied. Those marked with # assembler or user supplied.
Keying tab will cause the curser to move to each parameter position relavent to the command. Keying CR will action the command. The bottom 3 lines of the screen are used for editing (the very bottom), option indicators (the next up) and message codes.

4. A summary of commands is given below for reference:-

ES - Edit source,see paras 5 - 12.
A1 - Assemble pass 1 from store ,see paras 25 - 26.
A2 - Assemble pass 2 from store,paras 25 - 26.
F1 - Assemble pass 1 from file,paras 25 - 26.
F2 - Assemble pass 2 from file,paras 25- 26
LL - List labels,para 27.
FF - Fetch file,para 24.
EF - Examine file,para 23.
DF - Dump file,para 22.
OL - List option,para 30.
OM - Multi part source option,para 33.
ON - No object option,para 29
OP - Print option,para 31.
OR - ROM option for assembly to an EPROM location,para 34.
OS - Slow (CUTS) recording option,para 32.
EX - Exit to TANBUG

Entering a new program.

5. To enter new source code,key ES as the action (Edit Source), tab to the source start parameter if you wish to alter the default source start address and CR. A flashing curser should appear on the bottom line.This is where all editing action takes place. Key N (for New) CR. N0001 should appear where 0001 gives the start line no. and will be automatically incremented as each line is entered.

6. The format of editing lines is as follows:-

```
Col=   1 |2  3  4  5 |6 |7  8  9 10 11 12|13|14 15 16|17........
 Ed comm |Line no.   |SP|Optional label   |SP|OP code |OPerand
```

The curser will normally be aligned on column 7 ready for source code entry. Tab will rotationally move the curser round columns 7,14 and 17. BS (Back space) will clear the bottom line and set the curser on col 1 ready for a new edit command. Other editing commands are covered in the next section below. Control R will move the curser right and Control L left.

7. After keying the source code line,key CR to enter it. Source code is vetted prior to entry (as far as is possible). If all is OK the source line will scroll to the top half of the screen and the line no. incremented ready for the next line. If an error is detected the line will not be accepted and an error code is displayed on screen line 14. To exit from the editor key LF.

8. To display lines of code already entered, use the editor's S command (Show). Key BS (to clear the bottom line ready for a new editor command),S1 (Show from line 1) then CR. The first 8 lines of code will be displayed on the top half of the screen. The 'S' line no. will be incremented automatically so that subsequent lines may be displayed by simply keying CR.

Altering an existing program.

9. N line no. CR sets the editor to accept new code at any valid line no. If this equals the end line no, then new code is added to the end,if less than the end line no then new code will overwrite any existing code. Otherwise the action is as for entering new programs as described above.

10. Other editor commands are:-
A - Amend. The specified line no is replaced.
I - Insert. The line is inserted before the specified line no.
D - Delete. The specified line is deleted.
Whenever an insertion or deletion is made, the lines are effectively renumbered. So if you are working from a printed listing amend lines at the end of the program first so that earlier line numbers are not disturbed.

11. The required command is keyed followed by the line no. If CR is keyed at this point (i.e. with the curser at col 6), the line being amended/deleted/inserted before is displayed as a checking aid. To enter the amentment line,key over the displayed line and key CR. It will be entered (if valid) as long as the curser is past col 6. Alternatively, key the amendment line, CR immediately after the editor command for direct entry. In the case of delete simply move the curser past col 6.

Summary of editor commands.

12. Below is a brief summary of commands for reference:-

N CR	- New program from line 1.
N n CR	- New code from line n.
A n CR	- Amend line n.
I n CR	- Insert before line n.
D n CR	- Delete line n.
S n CR	- Show from line n.
LF	- Exit from editor.
BS	- Clear bottom screen line,set curser at col 1.

Tab   - Tab curser to cols 7,14 and 17.
Cntrl L- Curser left.
Cntrl R- Curser right.

Source code formats.

13. All source code takes the format:-
Label/space/OP code/space/OPerand
where Label is an optional name of up to 6 characters, the first character alphabetic the remainder alphabetic or numeric,
OP code is any of the standard 6502 op code mnemonics plus EPZ,EQU,ORG,BYT and WOR.
For standard op codes the operand may take one of the following formats:-

Address   - page 0,absolute or relative.
Address,X   -   "   "   "   "   .
Address,Y   -   "   "   "   "   .
(Address),Y- indirect Y
(Address,X)- indirect X.
#Constant   - immediate.
@   - accumalator (replaces A as used in XBUG).
Null   - for other single byte ops.
(Address)   - for JMP indirect

Address can consist of:-
a) A label or * (meaning 'this address') with (optionally) +/- constant,
b) A constant (giving a fixed address).

Constant can consist of:-
a) $hhhh - where hhhh specifies up to 4 hex characters.
b) 'c   - where c is a single ASCII character.
c) nnnnn - where nnnnn specifies up to 5 decimal digits.

14. Examples of valid instructions are:-

```
LABEL  LDA TABLE
       STA TABLE,X
       STA TABLE+4
       LDA $40
       LDA 'A     (=LDA $41)
       LDA 32     (=LDA $20)
       LDA #23
       LDA #$FF
       LDA 'b
       LDA (ABC-6),Y
       ASL @
       BEQ LOOP
       BNE *+4    (4 bytes on from the BNE instr address)
       JSR SUB2
       JMP (ADDR)
```

16. The format and effect of other ops is as follows:-

ORG Constant or * +/- Constant

This will cause the assembler to start placing the object code at the address given by the constant (first istance) or +/- constant on or back from the current address (second instance). This op overides the object start address specified via the assembler parameter. If you wish to keep your programs relocatable, avoid the first instance. Note that this op is necessary when using option R (see para 34).

Label EPZ/EQU Constant.

These equate an address (given by the constant) with a label.The label field is mandatory. EPZ is used for page 0 addresses and EQU for other addresses. Note especially that EPZ declarations must be made before the label is referenced. Failure to do this will result in a (detected and reported) assembler failure. EQU may be placed anywhere. It is generrally good practice to make all label declarations at the beginning of the program.

Label BYT/WOR Const,const....

These are used to place constants in store. BYT will place a single byte constant whereas WOR will place a two byte constant with the low byte first and the high byte second.
e.g. AD   WOR $1234 gives $34 in AD and $12 in AD+1.Label is optional.
Examples of valid statements are:-

```
ABC     BYT $0,'W,23
        WOR $321,15342
```

Label BYT/WOR Address,address....

BYT and WOR may also have labels as operands. The effect of this is to store the address of the label as the 'constant'. This is useful for loading addresses for indirect addressing whilst still retaining relocatability.

e.g. Suppose TAB is the label against a table:

```
TAB    BYT $1,$2,$3,$4
       BYT $5,..........
       etc.
```

By declaring:-

```
ATAB   WOR TAB
```

it is now possible to pick up the address of TAB for,say,storing in page 0 in order to indirect address:-

```
ITAB    EPZ $40
        LDA ATAB
        STA ITAB
        LDA ATAB+1
        STA ITAB+1
        LDA (ITAB),Y
```

17. Comments may be included in source code starting at cols 7 or 17 (if no operand). They should be preceeded by ; (semi-colon).

18. Note that it is not possible to assemble in page 0 since the assembler uses page 0 extensively.

Cassette file input/output.

19. File facilities provide for the dumping, examining and reading of source code. Object code should be handled using the normal XBUG cassette routines. Source code should always be dumped prior to running an assembled program in case the program runs amuck and corrupts the source.Alternatively place the source in TANRAM (if you have it!) and set $FFFF to protect it. Source code is compacted in store and on tape by the removal of redundent spaces. The default speed for recording is fast mode, should you wish slow (CUTS) mode then set option S (see para 11)

20. 'Files are output as a series of 256 (max) byte blocks.Each file is preceeded by a label block containing the name of the

file (as taken from the user name parameter) and ends with a trailer block containing a string of >>>>>>>>. In order to conserve user RAM space,the i/o routines use $200 (the top half of the screen) as a buffer. You may therefore observe i/o progress.

21. During assembly from tape,the processing actually takes place during the inter-block gap. This has been set long enough for most purposes (about 0.7 secs). Should programs with a very large number of labels be assembled from tape (necessatating long searches of the symbol table), it may be necessary to increase this gap. This may be done by responding to the prompt BLOCK GAP? during a cold start with a higher value. The default value is $0F.

DF - Dump file command.

22. The source code identified by the source start address parameter is dumped from line start no to line end no. This will normally be the complete program but you may alter the line start and/or the line end nos and only dump a portion of the code. There is nothing to stop you having two or more sets of source code at different RAM locations (if you're careful!).

EF - Examine File.

23. This command should be used *immediately* after dumping to check the recording. A tick will appear on the message screen line if the file is successfully found. Read or compare failures will cause message code R to be displayed,the file should then be re-examined or re-dumped. Escape from the read routines is possible by keying Control A but only if a block is currently being read (i.e. a file header block is not being searched for).

FF - Fetch File.

24. This reads the file identified by the name parameter into store at the address given by the source start parameter. The line start (=1) and line end parameters are updated. Escape is possible using Control A as above.If there are read errors R is displayed on the message line. If failure persists,then the file is partly recoverable because only the block(s) in error will be missing. To fully recover it will be necessary to identfy the missing lines of source code and re-key them.

Assembly.

25. Assembly may be performed from source code held in store or on tape (dumped via the DF command).To assemble the following commands are used:-

A1 - Assemble pass 1 from store.
A2 - Assemble pass 2 " " .
F1 - Assemble pass 1 from tape.
F2 - Assemble pass 2 " " .

Before assembling from tape,it is wise to first assemble from store as a means of fully vetting the code. Although as much vetting as possible is done by the editor during code entry, certain errors cannot be detected at this stage (e.g. labels not declared). If there is insufficient space in RAM to hold the object while vetting, then output of object can be supressed using the N option (see para 11). Errors during assembly will cause assembly to terminate with the offending line displayed on the bottom line and an error code on the message line. The error should be corrected and the program re-assembled.A list of error codes is given in the appendix.

26. Before assembly the following parameters must be correctly set:-

a) The source start,line start and line end parameters although these are automatically set by the editor and Fetch File command.

b) The object start address. Assembled code will be placed here unless overidden by an ORG instruction.

c) The symbol table END address. This specifies where the assembler is to build its symbol table containing the labels and thier associated addresses. This table is built BACKWARDS through store.

After assembly (including supressed object code runs), the end address +1 of the object code and the START address -1 of the symbol table will be updated.You may therefore see if sufficient space is available to hold the object. The assembler makes no other demands on RAM space above $400.

Listing the labels.

27. To list the labels,use the LL (List Label) command. This will display the labels and thier addresses.Options L (List) and P (Print) are also available - see below.

Options.

28. Various options are available during assembly. Each is switched on and off by the successive use of the appropriate command. Acive options are displayed on the screen below the message line. Note that options L (List) and P (Print) are not available when assembling from tape because of the time delays imposed by keying and printing respectively.

ON - No object.

29. This causes supression of the storage of object code. It has no other effect.

OL - List.

30. During pass 2 assembly,source and object code is generated on the bottom line of the screen and scrolled to the top half of the screen. The list option causes this process to halt every 8 lines pending the keying of LF so that code can be examined. CR will switch off the option and allow the assembly to complete uninterrupted. Note that because of the limited screen width the source appears on one line and the object on the next.The curser character is used to generate CRs for printing where source and object appear on the same line. This option has the same effect when listing labels.

OP - Print.

31. Causes the source and object to be printed using the print routine set up during cold start. This will either be the new TANBUG routine or one supplied by you. It is called with each character to be printed stored in the accumalator. CR ($0D) indicates a new line. It is necessary for user routines to save both X and Y regs. It also causes the label list to be printed for the LL (List Label) command.

OS - Slow.

32. Sets slow (CUTS) mode for cassette recording.

OM - Multi-Part.

This allows the consequtive assembly of several sections of source code.Its effect is to inhibit the resetting of the object assembly address on assembly. It also inhibits the clearing of the symbol table prior to Pass 1. Suppose there are 3 files of source code on tape which in fact constitute one program. To assemble these as one object program,perform the following:-

a) F1 on the first file as normal.
b) Set option M and F1 on the second and third files.
c) Clear option M,rewind the tape and F2 on the first file.
d) Set option M and F2 on the second and third files.
Assembly is now complete.

OR - ROM assembly.

34. This allows programs to be assembled for subequent EPROM or PROM programming. In this mode ORG will determine the object address for assembly purposes whilst the object start parameter determines the object's storage address.At the end of the assembly the object start parameter will be updated to show the END address of the stored object whilst the object end parameter will show the assembled end address.

Warm starting.

35. A warm start allows entry to the assembler with user parameters as previously set. However, this depends on certain locations not being corrupted in the meantime. These locations are:-
$40 to $43
$62 to $73
$FF
In addition, if the printer has been 'de-initialised', it will be necessary to key Control P twice to re-initialise (or do a cold start).

Additional notes.

36. Interrupts are disabled when in the assembler.

37. Certain extra op codes (MDF,MEN and MAC) are accepted by the assembler. This is to facilitate the eventual inclusion of macro facilities. They will be treated as WOR. Labels including the characters [\ ] are also allowed but should not be used as as they will have special significance to the macro assembler.

APPENDIX.

Error Codes

General.

Tick File found (not an error condition)

R Read or compare failure when reading cassette files.

C Invalid assembler commamd.

N Line number does'nt exist.

G Start line no > end line no.

Editor.

E Edit command invalid.

N Line no does'nt exist.

A Arguement (line no) to command required.

Assembler.

S Single byte operand wanted.
Offset too large for branch instruction.

D Double byte operand wanted.

I Invalid operand.

L Label wanted in label field.

A Addressing error (e.g. label not declared)

C Constant is invalid.

O Op code is invalid.

X Duplicate labels.

K Label not allowed in label field.

R Reconciliation failure probably caused by failure to pre-declare a page 0 label or a symbol table corruption.

AMENDMENTS TO ASSEMBLER

VERSION V1.2

The following changes have been made to the first version of the assembler.

a) The facility to escape from the read or examine routines by keying Control A is removed. This is because it causes problems when using non-latched type keyboards which continuously present data to Microtan's keyboard port. This causes sporadic exiting from the read or examine routines.

b) EPZ and EQU may now contain a label as operand. The label must have previously been defined.

```
e.g. A       EPZ $40
     B       EPZ A+2
```

Due to lack of space in the 4K, the error codes suffer slightly. Error code S is displayed for EPZ or EQU under the following circumstances:-
i) a page 0 address calculated (<$100) for EQU,
ii) a two byte address calculated (>$FF) for EPZ,
iii) RHS label not previously declared.

c) Instructions of the form LDA A,Y where A is a page zero address now generate the absolute (3 byte) form of the instruction rather than reject it as an invalid page zero format. i.e. LDA $40,Y generates machine code (hex) B9 40 00.

d) Instructions of the form LDA #label and LDA #label> are now included to aid in the manipulation of indirect addresses. #label gives as the immediate operand the low (or only in the case of page zero addresses) byte of the label address. #label> gives the high byte.

e.g.

```
A      EPZ $23
B      EQU $4567
       LDA #A      gives A923
       LDA #B      gives A967
       LDA #B>     gives A945
```

e) The header message on entry is amended to show version V1.2.