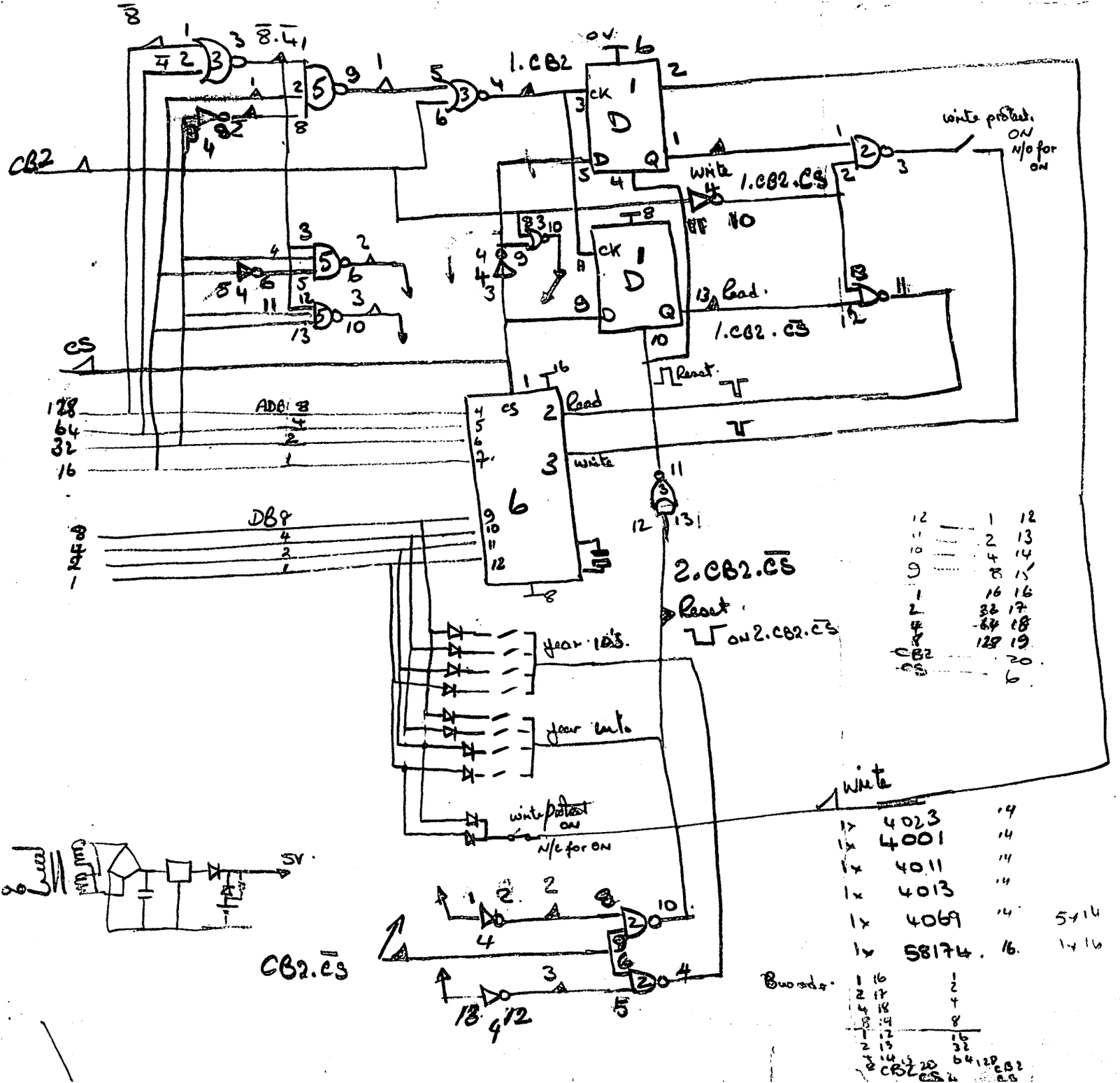


address	o/p Code
1.CS = Write	1 = 16 (to Select Read/Write
1.$\overline{CS}$ = Read.	
2.$\overline{CS}$ = Reset. Read Units	2 = 32 (20) Year
3.$\overline{CS}$ = Read 10's	3 = 48 (30)
2.CS = Read See Units	2 = 32 (20)
4.$\overline{CS}$ = Read/Write Tens Hours Units	3 = 32 (20) 4 = 64 (40)
5.CS = " Tens	5 = 80
6.CS = " Hrs Units	6 = 96
7.CS " Tens	7 = 112
8.CS. " Days Units	8 = 128
9.CS. " Tens	9 = 144
10.CS " Day of week	10 = 160
11.CS " Month Unit	11 = 176
12.CS " Tens	12 = 192
13.CS Write leap year	13 = 208
14.CS . Stop/start	14 = 224
15.CS write Interrupt	15 = 240

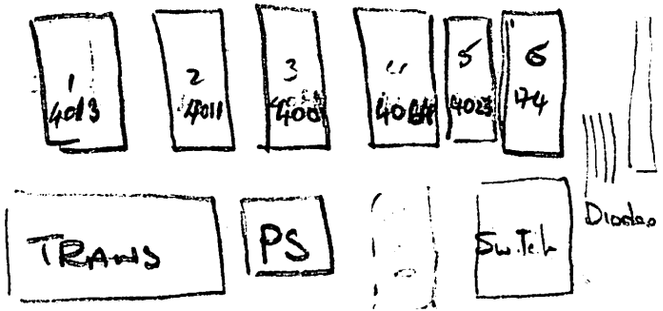
1 = leap year
2 = 11
4 = 13
8 = 15

1 = start
0 = Stop

1000 No INT

1x	4023	14
1x	4001	14
1x	4011	14
1x	4013	14
1x	4069	14
1x	58174	16

5 x 14
1 x 16



To enter a command type the single letter shown for the function you require. Some commands require one or more line numbers to be specified in addition. These are typed in immediately after the letter. The syntax is shown below. X represents the command letter, and (CR) represents carriage return.

X(CR)	used with A,K,R,W always
X(CR)	used with L,LN,P,PN to list all lines.
XAAA(CR)	used with I,S always
XAAA(CR)	used with C,D,L,LN,P,PN to operate on one line only
XAAA,BBB(CR)	used with C,D,L,LN,P,PN to operate on a block of lines.

Where AAA and BBB are decimal line numbers. The symbol $ may be used to reference the last line and may have a suffix of the form -AAA to represent a negative offset from this number.

e.g. L$-9,$ will list the last ten lines.

The search command requires a little more explanation to use it fully. After entering S followed by a line number and then carriage return the computer will wait for you to type another character. It will then search through the line and print out each character until it finds a match between the character you entered and the one it printed. At this point it will stop. You can now insert characters by typing them in or delete characters with delete or control H. Since a new line is being built up in the input buffer you can delete everything displayed in the line so far with control U, where upon the cursor will drop down to the next line. After deleting or inserting characters, or as an alternative, there are four ways of proceeding. Firstly you can terminate the line at the point shown by simply typing (CR). To append the rest of the line type control R,(CR). Both of these commands will cause the new line to be entered into the text area in place of the original line and the prompt will be re-displayed. The other two commands allow you to continue searching along the line. To search for another occurance of the same letter type control T,(CR). This will cause more of the line to be displayed upto the search character. Finally to search for another character type control N,(CR) and then the new letter. You can then continue by using any of the search commands described above.

At anytime when entering a command or text control H or delete can be used to delete the previously entered character. Control U will delete everything entered in the current line so that you can start a line again if it becomes a mess. To abort from text entry mode to command mode type control D. This should be done from an empty line as the line is not entered into the buffer first. To abort from a long unwanted listing, type control D. This will cause the prompt to be displayed. The editor provides a tab function so that neat listings can be generated. Tab is entered as control I. After each tab is entered the cursor steps along to the next tab position. These occur every eight characters.

We will now use the editor to enter a short program which will demonstrate the assembler later on.

The first command to use is kill. This effectively clears the text buffer ready for entering new text. In response to the prompt, type K(CR). The editor will respond with another prompt and is now ready to accept text. Since the text buffer is empty we will use the Append command. Enter the following listing. Use tabs to format it if you wish.

```
A
; A PROGRAM TO DEMONSTRATE TRAP
DW ENDPRG
START: LXI    D,STRING      ; PTR TO FIRST STRING
       LXI    H,SCREEN      ; PTR TO TOP OF SCREEN
       RST 1                ; CLEAR SCREEN
       CALL MSTRNG          ; MEMORY MAP MESSAGE
       MVI H,128            ; PTR TO VDU ROW 3
       CALL MSTRNG          ; OUTPUT STRING
       JMP MONTR            ; JUMP TO TRAP MONITOR
EOT    EQU Ø4H              ; DEFINE SYMBOLS
MONTR  EQU ØD5BØH
SCREEN EQU 11ØØH
STRING:DB 'A TEST FOR TRAP', EOT
STR:   DB 65,2ØH,123Q,84D,'R','I','N','G',EOT
ENDPRG:DS Ø                 ; EXAMPLE OF 'DS'
      END
  'Control D'
```

We now have a source file in memory. However a few errors have been included so that we can use some of the other editor commands. Firstly list the file using LN to obtain line numbers as well. The first change is in line 7. At present this is

```
                MVl H,128  ; .....
```

However the register we set up should be L. To change it use the Change command and retype the line:-

```
C7
                MVl L,128  ; PTR TO VDU ROW 3
'Control D'
```

The next error is line 12 where the screen is declared to start at location 11ØØH. In fact it should start at 1ØØØH . Use the search command to modify this:-

```
S12(CR)
```

now type 1. The editor prints out

```
SCREEN EQU 1
```

We need to change the second 1, so type 'Control T',(CR) and then the display will be

```
SCREEN EQU 11
```

By typing 'delete' and then Ø, the change is accomplished. Type 'Control R',(CR) to append the rest of the line.

In the original text the subroutine MSTRNG was ommitted. A suitable position for this is prior to line 10. It can be entered using insert as follows:-

```
I 10
; SUBROUTINE TO MAP VDU
MSTRNG:  LDAX D              ; GET CHARACTER
         MOV M,A             ; MEMORY MAP IT
         INX H               ; BUMP POINTERS
         INX D
         CPI Ø4H             ; WAS CHARACTER EOT?
         RZ                  ; YES, DONE
         JMP MSTRNG          ; NO, MAP REST
'Control D'
```

The program is now complete and it can be listed to verify that it is correct, using the L command. Line 10 is superfluous and has been included to demonstrate the final editing command. It can be deleted by typing D10(CR). If there are any errors you should now be able to change them. Before proceeding save the program on tape using the Write command. To invoke this, type W(CR). The editor will respond with TAPE HEADER=. Now type in the name you wish to give the file followed by carriage return. The tape recorder will be turned on and the file written onto tape. When it has finished the tape recorder is turned off and the prompt is displayed. To read a file back into memory type R(CR) and in reply to the TAPE HEADER= prompt give the file name followed by carriage return. The editor will respond with FILES FOUND: It will then print each header on the tape on a new line until it finds the required name. If a file is being loaded the cursor stays on the same line to indicate this. If an error is detected during the loading procedure, a question mark is printed on the next line. Note that only one question mark will be printed however many errors occur.

Now type 'escape' to return to the executive. After the menu has been printed type 2 to select the assembler.

Assembler

The assembler will respond with the prompt:

PASS?

There is a choice of six options. These are 0-4 and S. The assembler is a two pass assembler. This means that it scans through the source code twice before it completely generates the object code which the 8080 can execute. During the first pass values are assigned to all of the labels and other operands used by the program. The names and values are stored in a symbol table. During the second pass the object code is generated using the symbol table to give values to mnemonic operands. This assembler has two types of first pass. The one usually used is number 1. This clears out all of the

the program being assembled. The other first pass is Ø This is used when you assemble a program in parts, as you might if you do not have enough memory to hold all of the source code at once. It does not clear the symbol table, instead it adds any new symbols onto the end of those from the previous assembly. This saves putting a long list of equates at the beginning of a program to define the symbols used in the previous sections. There is a choice of three second passes. The first, number 2, will dump the object codes onto tape. This will usually be used after a program has been debugged to save it for future use. TRAP has a facility for loading these tapes which we will use later. Secondly there is number 3. This will generate an assembly listing of the program like the one on page 16. The listing can be generated on a printer by typing P3 as opposed to 3 in reply to the prompt. Before the listing starts a form feed code is issued which clears the VDU or sets the paper to the top of a form on the printer. At the end of a listing the symbol table is printed. The last type of second pass is number 4. This assembles the program directly into memory so that it can be executed directly. The final option to the pass message is to print just the symbol table. To do this type S or PS to obtain a copy on the printer.

The mnemonics used by this assembler are the industry standard ones recommended by Intel. These are listed in a number of books and the newcomer to assembly language programming is referred to 8080A/808S Assembly Language Programming published by Osbourne and Associates. This gives full details of each instruction and continues as a tutorial in assembly language programming. In addition to the normal instructions the assembler supports a number of pseudo-ops. These are directives to the assembler to perform certain operations. The pseudo-ops supported are ORG, END, DB, DW, DS and EQU. The ORG is used to originate the program or part of a program at a certain address. It sets the address at which assembly starts and takes the form:

```
ORG  2ØØØH
```

The numeric part can be any valid number or symbol.

The ORG should be used at the beginning of a program to define where it should start. If one is not used the start is set to zero by default.

END is the operator which stops the assembler. It must be the last line of a program as anything after it is ignored by the assembler.

The DB directive is used to define the value of one or more bytes in memory. This is useful for setting up tables and ASCII strings. It can take the following forms and can be preceeded by an optional label.

```
DB 23H          ; STORES ONE BYTE
DB 23H,24H      ; STORES TWO BYTES
DB SYMBOL       ; STORES ONE BYTE
DB 'R'          ; STORES ASCII CODE FOR R
DB 'A STRING'   ; STORES A STRING OF BYTES
DB 'MORE',Ø4H
```

Once again numbers can be in any valid base or a symbol. The result of DB SYMBOL is that the least significant byte of the value associated with SYMBOL is stored.

DW is similar in effect to DB except that it assigns a value to two bytes. However it can only be used to assign a value to a single word. The operand part can be a number, a symbol or a single character. The word generated will be in the usual 8080 form with the least significant byte stored first and then the most significant byte.

DS is used to reserve a block of memory for purposes such as a stack or buffer etc. The operand can be any valid number or symbol. The memory block reserved is not initialised in any way.

EQU is used to assign a value to a symbol. This is used to make it easier to change the program later (only one line need be changed for any number of references to a value) and to make the program easier to understand later. The form of an equate is:

```
SYMBOL EQU VALUE
```

VALUE can be another symbol, the value of which is defined or a numeric or ASCII constant. Once a symbol has been set using an equate, its value cannot be changed.

The source code is normally entered using the editor. The format of a line is:

```
label:  Mnemonic  Operands ; comment.
```

There are certain restrictions which apply to each of these four fields of a line and these are defined below. Each field is delimited by one or more spaces or tabs or a colon, comma or semicolon in the case of labels, register operands and comments.

A label consists of upto six alphanumeric characters, of which the first must be alphabetic. If a label consists of more than six characters the last ones are ignored. A label must be immediately followed by a colon. All labels used in a program must be unique.

The mnemonic field is one of the standard Intel instruction mnemonics or one of the pseudo-ops described above. It can be preceeded and followed by any number of space or tab separators.

The operand field consists of one or two operands. If there are two operands the first will be a letter or group of letters to specify a register. This must be followed by a comma to separate it from the second operand. Any other operands may either specify a register or memory. Registers are specified by A, B, C, D, E, H, L, M, SP or PSW. Memory may be referenced in several ways. The first is to use a mnemonic name. This must be defined elsewhere either by using it as a label or by using an equate. The alternative is to express it as a constant. Constants come in several forms. Firstly they may be a single ASCII character enclosed in single quotes, such as 'A'. The characters used as terminators must not be used in this way. They are space, colon, semi-colon, comma, tab and carriage return. Secondly constants may be expressed as numbers in octal, decimal or hexadecimal. A number may have upto five digits. It is followed by a Q, D or H to specify the radix as 8, 10 or 16. If no terminator is provided the number is assumed to be decimal. Numbers are taken as modulo 2^8 or 2^{16} according to whether they represent 8 or 16 bit values.

The comment field is used to indicate the purpose of an instruction for future reference when you need to know what a program is doing. Comments should be included in all your programs for this reason. A comment must be preceeded by a semi-colon.

All of the fields are optional, but are needed in some cases (e.g. PUSH must be followed by an operand). Blank lines are not permitted.

Having described the format of an assembly language program we will now demonstrate the use of the assembler.

The assembler generates eight error messages. They signify errors which are fatal to the assembler'soperation. The error messages take the form of a plain language statement followed by a line number and on passes 1, 2 and 4 the line containing the error is printed. In the case of the MISSING TERM error, if the missing term is the last term on the line the following line will be printed owing to the way that the assembler works, however the line number will indicate the line with the error. The error messages are self explanatory and in most cases after correcting the source program, the program can be re-assembled. The only exception is after a SYMBOL TABLE FULL or ASSEMBLY INTO SOURCE REGION error. To overcome the first error, it is necessary to assemble the program in two parts or to add more memory to allow a larger symbol table. In the second case, the program must either be relocated to a different part of memory or written to tape and then loaded later. It should be noted that no check is made to ensure that assembly does not occur into the symbol table and if this occurs some strange things will happen.

So far we have loaded a source file into memory and entered the assembler. In response to the prompt, type 1. This will assemble the program and create a symbol table. After a moment the prompt will be issued again. Now type 3. The screen will clear and the program will be listed on the screen. Notice that the program has been origined at zero by default. After the program, the symbol table is printed and then the prompt issued. Type 4 in response to this to assemble the program into memory. An error message will be printed

source text. Type 'escape' to leave the assembler and then in response to the executive's menu select the editor. We will now insert an extra line to define the program origin. Type the following:

I2

```
    ORG   18ØØH
```

'control D'

Now escape from the editor and select the assembler. Since we have changed the origin we must create a new symbol table, before assembly into memory. Use pass 1 and 4. Type S to list the symbol table and note the value of the symbol ENDPRG. It should be 183AHex. If not check that the program has been entered correctly. Now escape back to the executive and type 3 to enter the dis-assembler.

Dis-assembler

The dis-assembler will announce itself with the prompt:

START:

The purpose of the dis-assembler is to produce formated listings of machine code programs in memory. The listings are formated in a similar way to the listings produced by the assembler. The only difference is that there is no comment field. It is possible to obtain labels and mnemonic operands. This feature will be dealt with next. If mnemonics do not exist for operands their values will be printed as two or four digit hex constants as appropriate. Dis-assembled listings can be produced on either a printer of the vdu. Care must be taken when dis-assembling programs because data will be dis-assembled as well as instructions. Hence it is only useful to dis-assemble the part of a program which contains the op-codes. Obviously the start address for dis-assembly must be at the beginning of an instruction.

In response to the START: prompt type 18Ø2(CR). If you make a mistake, type any non-hex character and a question mark will be printed. You must then re-enter the whole number. After typing the carriage return the prompt END: will be displayed. This is asking for the address at which to stop dis-assembling. Type 182Ø(CR). Now the prompt PRINT? is displayed. If you type Y the listing is directed to the printer, otherwise it is displayed on the VDU. The listing generated is of the program typed in and assembled earlier. It contains labels because the symbol table has not been cleared since the assembler was used and generated a symbol table. To abort a long dis-assembly, type control D and the START: prompt will be re-displayed. At the end of the dis-assembly, this prompt will also be displayed. In response to the prompt type escape to enter the executive. Now type 4 to create a symbol table.

Symbol Table Creator

This program allows you to generate a symbol table which is used by the dis-assembler to put labels and mnemonic operands into the listings that it generates. The symbol table generated is identical to the one used by the assembler in its format. The labels can consist of up to six alpha-numeric characters, but the first character must be alphabetic.

Symbols can either be appended onto the end of a current symbol table or a new table can be generated. On entry into the program the prompt

CLEAR SYMBOL TABLE?

is printed. If you type Y the table is cleared otherwise symbols are appended onto the end of the current table. Type Y and then dis-assemble the part of TRAP from CØØØ to CØ2Ø by typing escape in response to the next prompt. This will show an example of a dis-assembly with no labels. After this re-enter the symbol table creator and clear the symbol table. The next question asked is:

set up table end address?

This address is the last address which is available to the symbol table. The table starts at the top of RAM (as set up by the monitor's initialisation) and symbols are inserted down from here. Hence the address defines the low limit of the table. This address is set up by the editor as the top of the text buffer. However if the editor has not been used or you have a program residing above the text which you do not want to destroy, this address must be changed. Assuming that you have at least 3k of RAM in your TRITON you need not set this address for this example so type any response other than Y. The next prompt asks for the SYMBOL VALUE: This is specified as a hex number in the same way as the addresses entered earlier. The last four digits of a number are used throughout TRAP when a sixteen bit value is required or the last two for an eight bit value. Hence 16Ø3 and 2516Ø3 are equivalent. In addition leading zeros can be omitted and ØØØ3 can be entered as 3. Enter CØØØ as the value and then the prompt

SYMBOL NAME:

is displayed. This is asking for the label. If an invalid character or more than six characters are entered then a question mark is issued and you must retype the whole name. Type EXECTV(CR) in reply. In response to further prompts enter the following values and names.

1470	STACK	CØ32	RCV
CØ55	MENU	CØ2D	CHECK
ØØ23	PDATA	CØ24	INVAL
CØ43	OPTAB	CØØA	GOPT

Finally type escape to return to the executive. If you now perform a dis-assembly from CØØØ to CØ2Ø all of the labels will be included. If an attempt is made to assign two values to a label message will say SYMBOL REDEFINED. The original value will be retained and the new entry ignored. If an attempt is made to enter a label when there is not room the message TABLE FULL will be printed and then after a few seconds the menu is displayed. Escape to the executive and type 5.

Single Step

Single step responds with the prompt START: It is asking for the address from which you wish to single step. The single step routine is one of the powerful debugging facilities

available within TRAP. It allows you to execute a single instruction in a program and then examine and change the machine registers and memory locations before executing the next instruction. When you first enter the single step function the stock pointer is set to the top of memory. This is because the debug routines use the stack at 147ØH which is the normal default stack. If you wish to use a different stack, you should set it in the first instruction. Programs which are being single steped should not use memory locations between the limits 14Ø4 - 141Ø, 1468 - 147Ø and 14EØ 14E5H inclusive. To start single stepping type the address' 18Ø2 in reply to the prompt. After typing the return key the display will show a dis-assembly of the program at that location, in this case LXI D,1821. No mnemonics are printed because the symbol table was redefined earlier. Underneath the dis-assembled line, a register dump is displayed. The D register will be equal to 1821, the stock pointer, S, will be equal to the top of memory, and the program counter, P, will be pointing to the next instruction at 18Ø5. The values of the other registers are not defined. The format of the register display will be described shortly. After executing the step the monitor section of TRAP is entered. It is possible to step through RAM or ROM.

Monitor

The TRAP monitor contains many of the functions of the HUMBUG monitor. The functions available are B, T, G, O, C, V, R, P, H and S. The monitor is provided as an aid to debuging rather than for program entry and this is reflected in the functions provided. To abort from functions and listings use control D. This will get you back to the monitor in most cases. We will now look at each function in detail. Errors are flagged by a question mark.

R - Register display and modify.

After typing R the monitor will wait for another character. Type space. This will give a dump of the pseudo-registers. When a breakpoint is encountered or a single step is executed the 8080 registers are dumped into memory so that the program's environment is saved ready to be reloaded when execution is resumed. Using this function it is possible to modify any of the registers first. The first group of characters displayed, indicates the state of the flags. Each flag bit is represented by its initial letter and is followed by a one or zero to indicate whether the flag is set or reset. In the order displayed, the flags are carry, parity, auxillary carry, zero and sign. The next field indicates the value of the accumulator. The following three fields represent the BC, DE and HL register pairs. In each case the first two digits represent the most significate register half (i.e. B, D and H). The final two fields give the sixteen bit values of the stack and program pointers. To modify a register type the letter shown in the display after the R. Note that C, E and L are not accepted and F is used to modify the flags. To change the B register type RB, enter the two digits required and then re-enter the contents of C as they were (i.e. last two digits displayed). After modifying a register a dump of them all is given. To modify the flags, type RF. The old value of the carry will be displayed and you must enter a 1 or Ø to indicate whether you want it set or reset. After the 1 or Ø enter carriage return. If an

invalid combination of keys is pressed the cursor will drop onto the next line and you must re-enter the status of that flag bit. The parity flag is then displayed and so on until the sign flag is changed.

P - Program

This function allows you to inspect and change memory locations in the same way as in the TRITON monitors. A start address is requested and then the address is displayed followed by the data in that address. The data can be optionally modified by entering a hex number. The last two digits are used to step on to the next location type carriage return. To step back to the previous location type up-draw (shift~). When you have finished modifying memory exit with control d.

H - Hex dump

This function produces a formated hex dump between two specified addresses. In response to the start prompt enter CØØØ. An end prompt will now be displayed. Enter CØ4Ø After typing return, the first address will be displayed followed by sixteen data bytes. Further lines are displayed in this format until the last address is reached.

V - Vdu switch

The HUMBUG and V6.1 monitors provide a printer interface and all of the monitor output can be sent to the printer by typing V. With the L6.1 monitor the output will be echoed on the VDU as well. If a further V command is issued, output reverts as normal to the VDU.

G - Go

The go function is used to start a program running. It asks for a start address and then responds with

SET BREAKPOINTS?

If the answer is Y, the breakpoints set up in the breakpoint table will be set into the program. The use of breakpoints will be described later. If anything else is entered, the program is run normally. The top address of memory is loaded as a defulat stack pointer to prevent the user stack being corrupted by the monitor during debug operations.

C - Continue

The continue function is used to continue running a program after a breakpoint or after single-stepping part of it. The contents of the pseudo registers are loaded back into the 8080 machine registers. Since the program counter is one of these registers, the computer jumps to the address held in the pseudo program counter.

O - Output to tape

This function allows areas of memory to be output to tape using the format described earlier. Typing O will cause the machine to ask for a start address after this and end address is requested. These addresses specify the inclusive limits

of the area of RAM to be dumped. Finally a tape header is requested. This is the name recorded onto the tape for the file and can use any characters. No load facility is included in the monitor since this is achieved by the Load Program facility.

S - Step

The S function is used to continue single stepping a program after the first step or after a breakpoint. The address from which the step is performed is assumed to be in the pseudo-program counter. The pseudo-registers are reloaded before the instruction is executed to maintain user program integrety. After the instruction is executed the registers are dumped back and the monitor is re-entered.

T - Trace

The T function is used to trace through the execution of a program. It performs a series of single steps repeatedly until the number of requested steps has been executed. Used from the monitor a start address is not requested. The address in the pseudo program counter is used. After typing T the number of steps you wish to trace is requested. This is entered as a decimal number. After this you are asked whether the output should be printed. After this question has been answered the Trace is executed with each line being dis-assembled before it is executed. The register dump indicates the effect that the instruction has had on the processor. The trace can be aborted by typing control D. This results in a return to the monitor.

B - Breakpoint set

The B function simply provides an easy way of entering the breakpoint utility without going via the menu.

When a breakpoint is encountered, TRAP saves the 8Ø8Ø registers in the pseudo registers and then resets all of the breakpoints so that the code is in its original form without the breakpoint instructions. These consist of RST 7's and so if the program starts to execute non-existant memory a breakpoint will be generated. If an RST7 is found and there is no breakpoint in the table at that address a message is printed to indicate that no breakpoint has been found. When the registers are saved the program counter is decremented to point to the location of the next instruction to be executed, unlike HUMBUG. When the program is continued, it is single stepped on this instruction so that an immediate breakpoint is not generated, then the breakpoints are loaded into the program and it is allowed to run.

Using the G function run the program that you typed in earlier. Do not set the breakpoints. The start address is 1802. It should clear the screen and memory map two messages onto it. Owing to a bug in the program the EOT character will also be mapped onto the screen. Finally it re-enters the monitor. Now save the program on tape using the O function. Use 18ØØ for the start address and 183A for the end address. After recording the program escape to the menu, then type 8 to load a program.

Load Programe

This utility is used to load tapes which have been generated either by the assembler Pass 2 or by the monitor. We will use it to reload the tape just recorded. In response to the TAPE HEADER = prompt type in the name used during recording, followed by carriage return. The computer responds with FILES FOUND:

It will now list each file found on the tape. If the file is being loaded the cursor will remain on the same line as the file name, otherwise it drops down to the next line. If a checksum error occurs during loading a question mark is printed and the rest of the file is loaded. If the error was in a block length byte, the end may not be recognised and interupt 2 will need to be used to abort. If a file is loaded succesfully a jump occurs to the executive. Type 7 to enter the trace utility.

Trace

As explained earlier trace is used to execute a series of single-steps and display the register contents between each step. Enter 1802 as the start address and then enter 10 as the number of steps. The first ten steps of the program will be executed. At the third step the instruction executed is a RST 1. After a few steps the accumulator is loaded with a Form-feed code and the monitor character output routine is called. It is not advisable to trace through this as there are upto about 1000 instructions executed before the routine returns! If a program enters a monitor routine it is advisable to set a breakpoint at the return address and then continue to this. When the trace has finished it re-enters the monitor. Now type escape and 6 to enter the breakpoint routine.

Breakpoint Set

The breakpoint facilities provided by TRAP are fairly extensive. Eight breakpoints may be set within a program and each breakpoint point can be passed up to 255 times before a breakpoint is executed. On entering the breakpoint routine you are asked whether all of the break-points should be cleared. When initially entering the routine, this should be done. The routine then asks if you wish to insert, delete or list breakpoints. Each function is initiated by typing its initial letter. Type I to insert a breakpoint into the table. The number of this breakpoint is then requested and this is a number between 1 and 8. Enter 1. The address is now requested. Enter 1804 for this. Finally the number of loops is requested. Use 1 in this case. We now have a breakpoint immediately after the clear screen instruction. We will also put a breakpoint in the subroutine to map characters onto the screen. Insert this at address 1818. Set the loop count as 16. This allows the instruction at 1818 to be executed 15 times before a breakpoint occurs on the sixteenth encounter of address 1818. Make sure you use a different breakpoint number or you will overwrite the previous one. If you now list the breakpoints, the two which are set will be displayed. The format is:-

number: address number of loops.

A PROGRAM TO DEMONSTRATE TRAP

```
0000                        ORG     1800H
1800  3A 18                 DW      ENDPRG
1802  11 21 18    START:     LXI     D,STRING;PTR TO FIRST STRING
1805  21 00 10               LXI     H,SCREEN;PTR TO TOP OF SCREEN
1808  CF                    RST     1       ;CLEAR SCREEN
1809  CD 17 18              CALL    MSTRNG  ;MEMORY MAP MESSAGE
180C  11 31 18              LXI     D,STR   ;PTR TO 2ND STRING
180F  2E 80                 MVI     L,128   ;PTR TO VDU ROW 3
1811  CD 17 18              CALL    MSTRNG  ;OUTPUT STRING
1814  C3 B0 D5              JMP     MONTR   ;JUMP TO TRAP MONITOR
1817  1A           MSTRNG:  LDAX    D       ;GET CHARACTER
1818  Fe 04                 CPI     04H     ;IS IT EOT?
181A  C8                    RZ              ;YES,DONE
181B  77                    MOV     M,A     ;ELSE MEMORY MAP IT
181C  23                    INX     H       ;BUMP POINTERS
181D  13                    INX     D
181E  C3 17 18              JMP     MSTRNG  ;MAP REST
1821                 EOT    SET     04H
1821                 MONTR  EQU     0D5B0H
1821                 SCREEN EQU     1000H
1821  41 20 54      STRING:  DB      'A TEST FOR TRAP',EOT
1824  45 53 54
1827  20 46 4F
182A  52 20 54
182D  52 41 50
1830  04
1831  41 20 53      STR:     DB      65,20H,123Q,84D,'R','I','N','G',EOT
1834  54 52 49
1837  4E 47 04
183A                 ENDPRG:  DS      0       ;EXAMPLE OF A DS
183A                 END
```

```
ENDPRG 183A  START 1802  STRING 1821   SCREEN 1000
MSTRNG 1817  STR   1831  MONTR D5B0    EOT    0004
```


Now escape to the executive and enter the monitor. Use the G. function to start the program at address 1802. Set the breakpoints this time. The screen should clear and then a breakpoint is reached causing the registors to be dumped and the monitor to be entered. The program counter will indicate the first breakpoint address of 1809. The HL and DE registers are loaded with 1000 and 1821 Hex. So far, so good, so continue by typing C. The first 15 letters of STRING are memory mapped onto the screen at the top left and then a breakpoint is issued at address 1818 as explained. The accumulator now contains 04 the EOT character. Step the processor once using S. The instruction MOVM,A is encountered and the accumulator is mapped onto the screen. Clearly it is necessary to examine the accumulator before outputing a character to see if it is EOT. Using the P function, patch this modification in as follows:-

1818 - FE, 1819 - 04, 181A - C8, 181B - 77, 181C - 23
1810 - 13

Now re-run the program with breakpoints set. Continue from the first breakpoint so that the message appears. Now single step and the CPI04 is encountered. This sets the zero flag. The next step causes a return on zero. We have achieved the aim of not displaying the EOT. Now type continue to carry on. The program will run to completion and return to the monitor. Since this return is not via a breakpoint, the breakpoints are not reset and this must be done manually at 1809 and 1818 using P to insert CD and FE. Now run the program without breakpoints to confirm that it works completely. This concludes the description of the facilities of TRAP.

Implementation of TRAP

TRAP is contained in a set of eight 2708 type EPROMS and uses the memory block from c000 - DFFF. The EPROMS should be fitted to an EPROM Card with position 7 selected for the address decoding. The EPROMS should be fitted as follows (with reference to the EPROM card circuit).

TRAP A	1C7	TRAP E	1C11
TRAP B	1C8	TRAP F	1C12
TRAP C	1C9	TRAP G	1C13
TRAP D	1C10	TRAP H	1C14

Transam
Microsystems Limited
59/61 Theobald's Road
London WC1X 8SF.
Tel: 01-404 4554
Telex: 24224 (Ref. 1422)

Mr J E Young
P O Box 5939
Manama
Bahrain

9 April 1985

Dear Mr Young

Thank you for your letter regarding the Triton system, you would be surprised how many people are still getting a lot of mileage out of these machines.

Please find enclosed the TRAP documentation, I trust it is what you need. Regarding graphics boards, I am afraid we do not have any add ons for the Triton any longer, the person to contact would be:

Mr John Owen
4 Guffitts Rake
Meols
Wirrel
L47 7AD

He runs the User Group and may be able to help.

Yours sincerely

Dale Hyde
Sales

Enc

TRITON SOFTWARE

TRAP V2.1 - TRITON RESIDENT ASSEMBLY LANGUAGE PACKAGE

The various monitor programs available for TRITON provide the machine code programmer with a limited tool for entering, de-bugging and running programs. TRAP has been designed to make the writing and testing of machine code programs as easy as possible. It provides an editor to allow you to enter programs into the machine in standard mnemonic form. The editor can of course be used for other purposes such as composing documents for printing later on. The file created with the editor can be saved on tape for later use or changes. Once the source listing of the program has been entered the assembler can be used to convert it into instruction codes that the processor can execute. This object code can be put directly into TRITON's memory, onto tape or printed out as an assembly listing on the VDU or a printer. One of the other options in TRAP allows you to load programs assembled onto tape. The rest of TRAP is devoted to a powerful set of debugging aids. Firstly there is TRAP's own monitor. This has been included so that TRAP is more convenient to use. The monitor is a subset of the HUMBUG monitor and has the functions P G O R C H V. The register dump and modify routine has been changed for easier use. In addition Step, Trace and Breakpoint set have been included, these are jumps into other parts of TRAP to save losing the information on the screen. Programs run under TRAP can have up to eight breakpoints inserted in them to stop the program at strategic positions. On reaching a breakpoint the monitor is entered. In some cases it is desirable to run a program one instruction at a time in order to locate a bug and TRAP provides a single step routine to do this. Successive instructions can be executed by using the monitor step command. It is also possible to do an automatic step using the Trace function. This allows you to specify how many steps you want to execute and displays the state of the processor after each step. The final part of TRAP allows you to dis-assemble programs that are in TRITON's memory to produce an assembler type listing. It is possible to build up a symbol table of labels and addresses before you start so that a fully labelled listing is produced.

TRAP is designed to be run on the TRITON Vdu and unless you direct output to the printer all output is on the Vdu. TRAP can be used with any of the existing TRITON monitors. The package resides in eight 2708 EPROM'S in address space CØØØH to DFFFH. The tape format used within TRAP is somewhat different to that used in the monitor. The synchronising bytes and the tape header are recorded in the same way but the main body of the file is split up into blocks. Each block can consist of up to 64 bytes of data. It is preceeded by the starting address of the block, and a count of the number of data bytes. A checksum is added at the end of each block for error detection. We will now look at each section of TRAP in detail and specify the commands which are available. To enter TRAP use the G Function and go to address CØØØH.

Executive

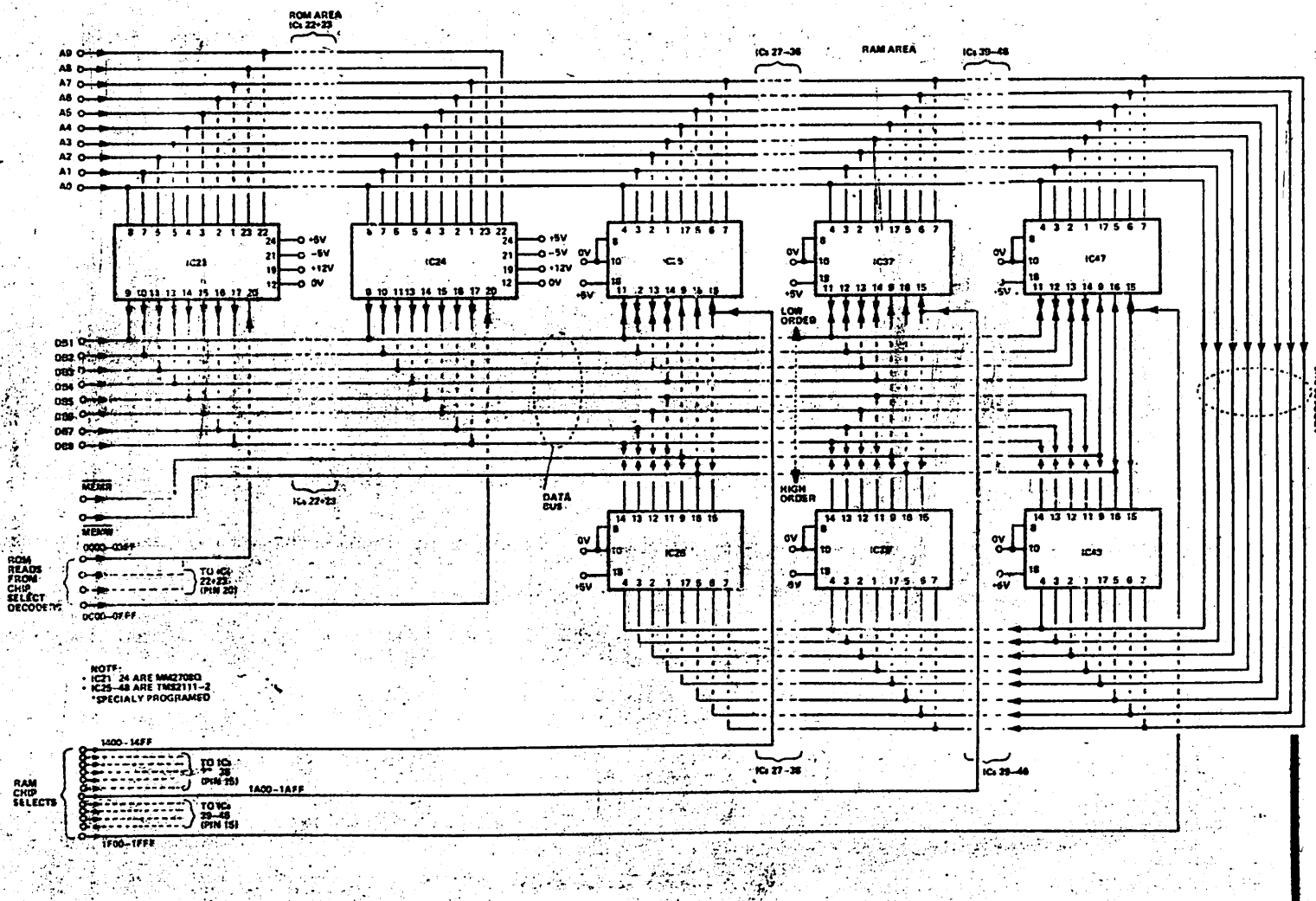
The executive is the name given to the program which joins all the functions of TRAP into a complete package and interfaces it to you, the user. When you enter TRAP, the executive takes control and prints out a menu. This lists all of the functions available and waits for you to select one. To select the function that you require, simply enter the number which is alongside the name, you do not need to press the return key. If you enter anything else you will be prompted again for a correct entry. The executive will then transfer control to the required function. At anytime when TRAP is expecting you to type anything, you can press 'ESCAPE' and return to the executive. If at any time you press 'CONTROL C' you will leave TRAP and enter the monitor on the main board. Type 1 and enter the editor.

Editor

The editor's prompt is >. The editor commands are summarised as follows:-

A - Append	Appends the lines of text entered to the end of the text currently in the text buffer. This means lines are added after the last line.
K - Kill	Resets the text buffer pointer and effectively clears the text buffer, ready for storing new text.
L - List	Displays the line or lines specified on the Vdu.
LN	As above, but adds decimal line numbers.
P - Print PN	As for list but output is on the printer via the serial output port.
D - Delete	Delete the line or lines specified from the text buffer.
I - Insert	Insert the lines of text entered immediately before the line specified.
C - Change	Delete the line or lines specified and insert the lines entered in their place. The number of new lines does not need to match the number of lines deleted.
R - Read	Load text from the tape recorder into the text buffer.
W - Write	Dump text from the text buffer onto the tape recorder.
S - Search	Searches the line specified for the next character entered. The contents of the line up to that point are printed and new characters may be inserted or original characters deleted the rest of the line can then be optional appended, or a search for another character made.

RAM & ROM



Circuit diagram of the ROM and RAM circuitry. Note that in the basic machine IC 24 is omitted as are ICs 33-48.

HOW IT WORKS

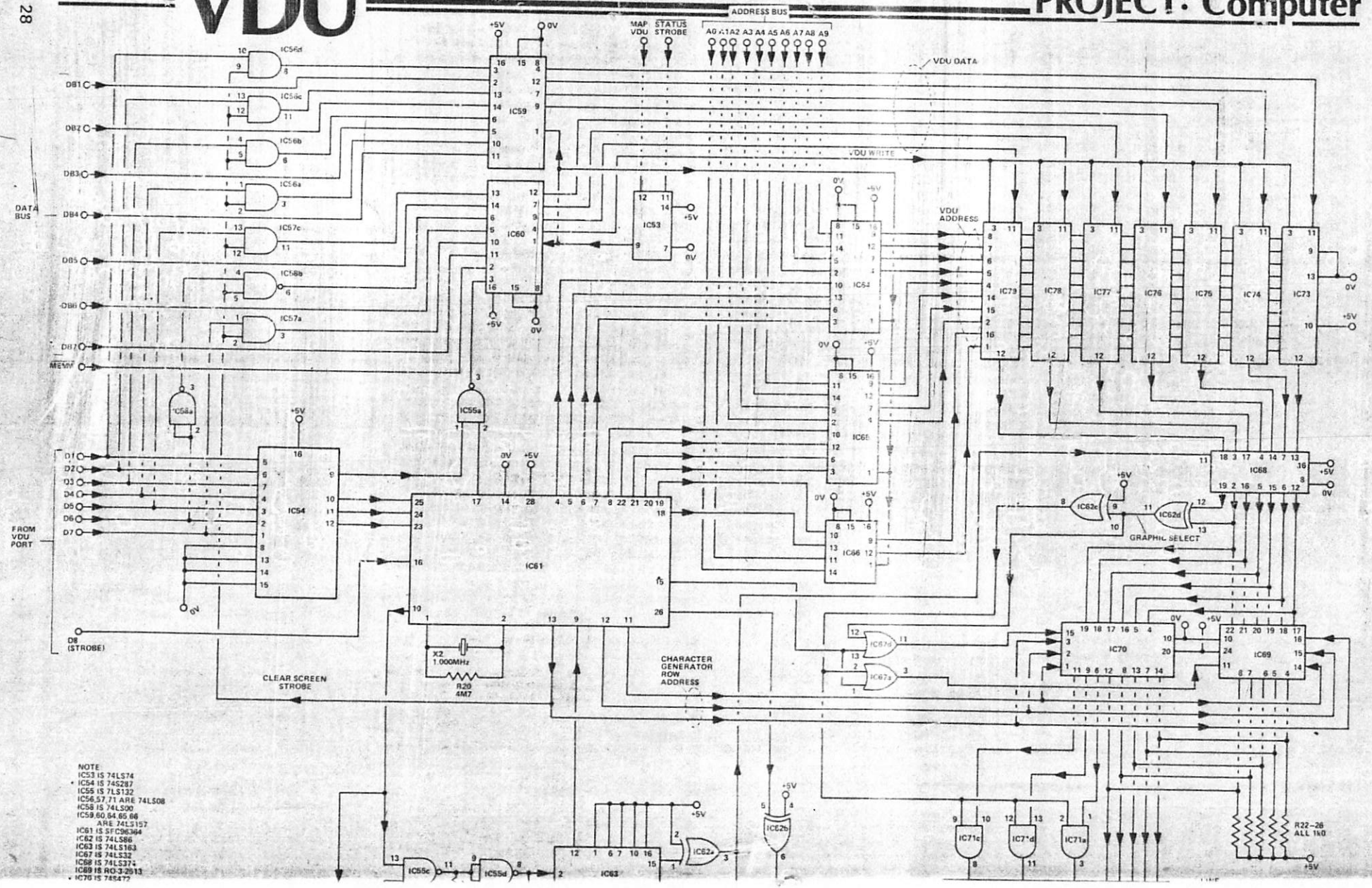
The circuit diagram of this section has been abbreviated as most of the memory circuitry is a repeat of the same theme. You can clearly see the difference between the ROMs and the Read/Write RAMS. There are four of the former – all 2708s but in the standard machine only three are used immediately. The 2708 is an ultra violet erasable ROM which contains 1,024 (decimal) bytes of memory each being 8 bits wide. To access a specific byte within it you need a 10 bit address and A0 through A9 are used for this purpose. The eight output pins are tri-state which are enabled by a "0" on pin 20 (the chip select input). The respective outputs from each of the ROMs can therefore be commoned together on the data bus. The "Programming Enable" pin (18) is only used when the devices are being programmed and therefore is left disconnected within the system. We use the block select signal gated with MEMR to provide the Chip Select strobe for the ROMs (this is described elsewhere).

The Monitor program is located within IC21 which starts at address location 0000H so that the computer will always go through a firmware initialisation routine when switched on. The Power On Reset ensures that the first instruction the CPU reads will be the one located at 000H. BASIC is located within ICs 22 and 23.

The RAM area of memory comprises TMS 2111-2 chips. These each contain 256 locations that are four bits wide. As we need to store eight bit bytes of data two chips are required for each 256 byte block of memory. The odd number designations IC25 to IC47 correspond to the low order nibble of the byte while the respective even numbers (IC26 to IC48) correspond to the high order. Only eight address lines (A0 through A7) are required to uniquely select a byte within this organisation of a chip pair but we need to specify which pair by means of the Chip Select lines (these have been decoded elsewhere in the system).

The 2111s have internal chip select and Read/Write gating so we are able to drive the MEMR and MEMW inputs direct from the

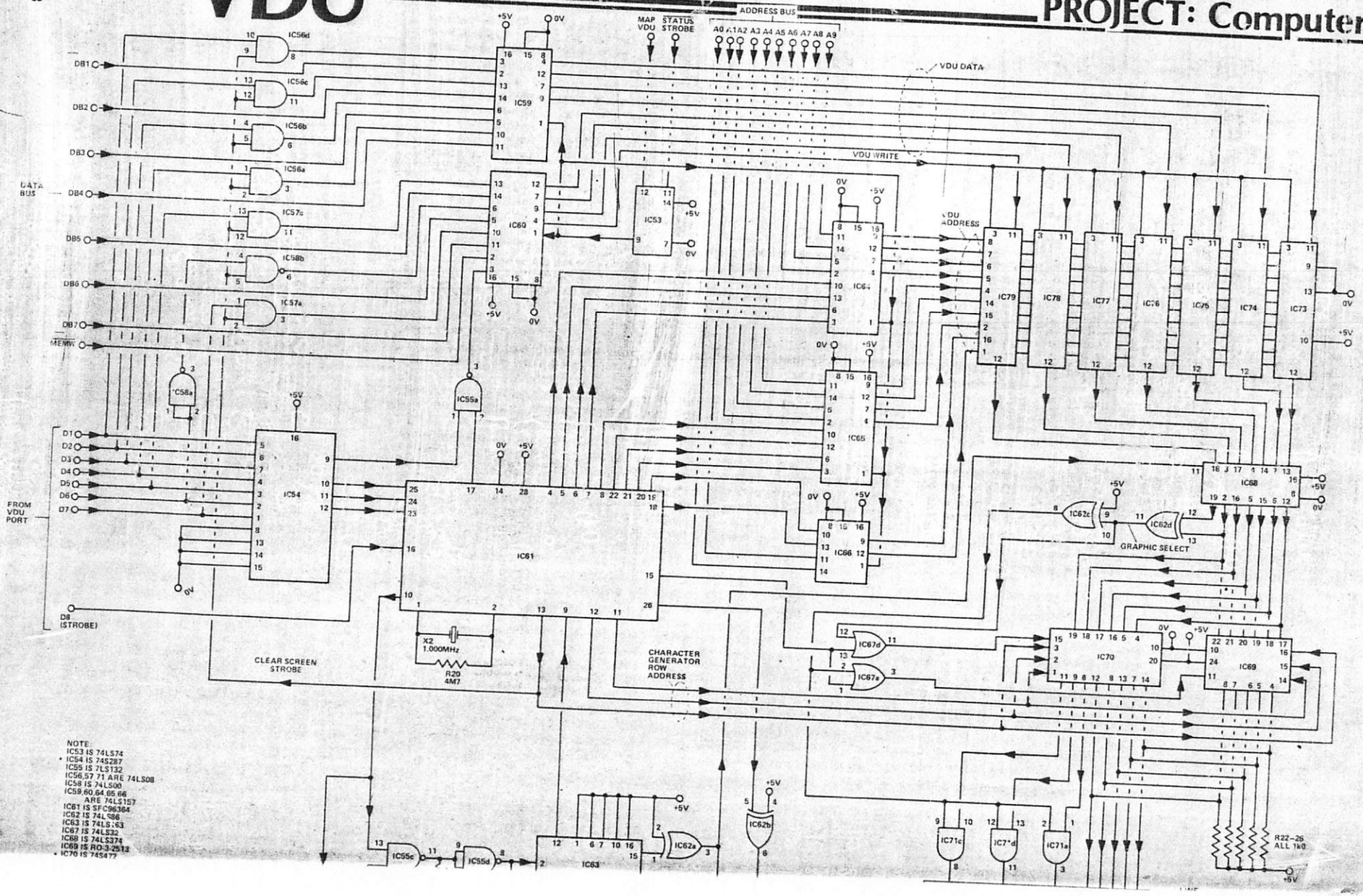
VDU



NOTE
IC53 IS 74LS74
IC54 IS 74S287
IC55 IS 7LS132
IC56,57,71 ARE 74LS08
IC58 IS 74LS00
IC59,60,64,65,66 ARE 74LS157
IC61 IS SFC96364
IC62 IS 74LS86
IC63 IS 74LS163
IC67 IS 74LS32
IC68 IS 74LS374
IC69 IS RO-3-2513
IC70 IS 74S472

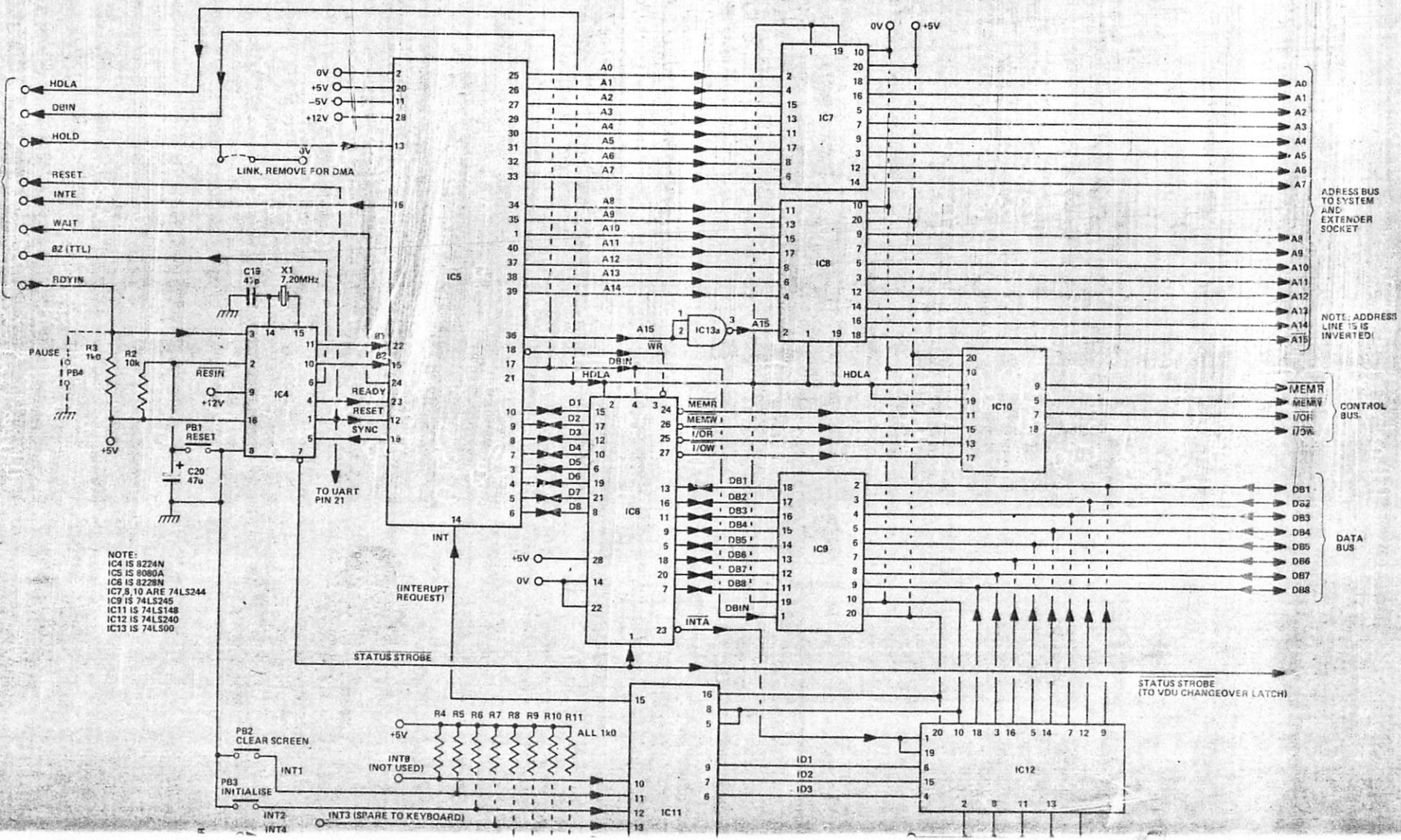
VDU

PROJECT: Computer



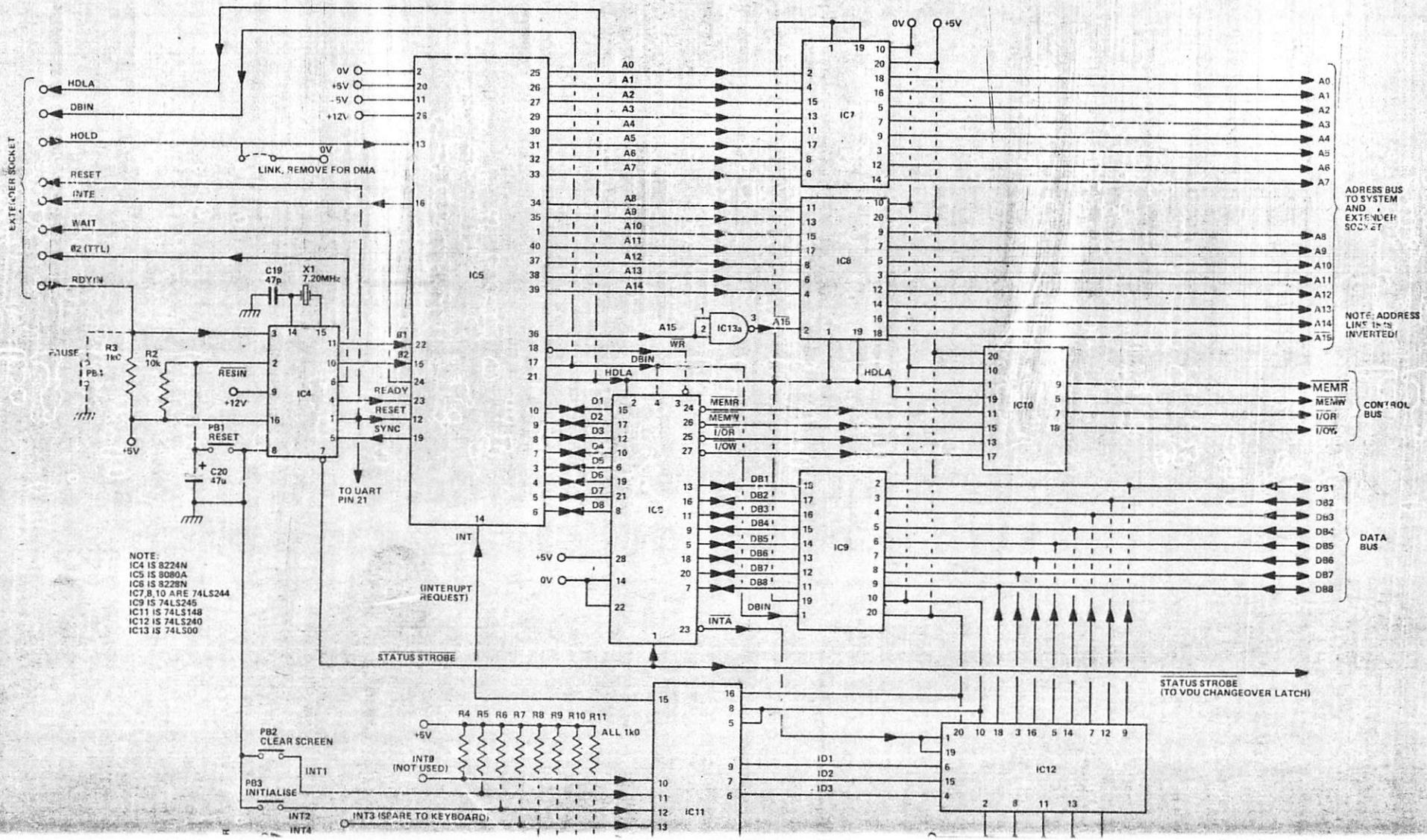
NOTE
IC53 IS 74LS74
IC54 IS 74S287
IC55 IS 74LS132
IC56,57,71 ARE 74LS08
IC58 IS 74LS00
IC59,60,64,65,66 ARE 74LS157
IC61 IS SFC96364
IC62 IS 74LS86
IC63 IS 74LS163
IC67 IS 74LS32
IC68 IS 74LS374
IC69 IS RO-3-2513
IC70 IS 74S472

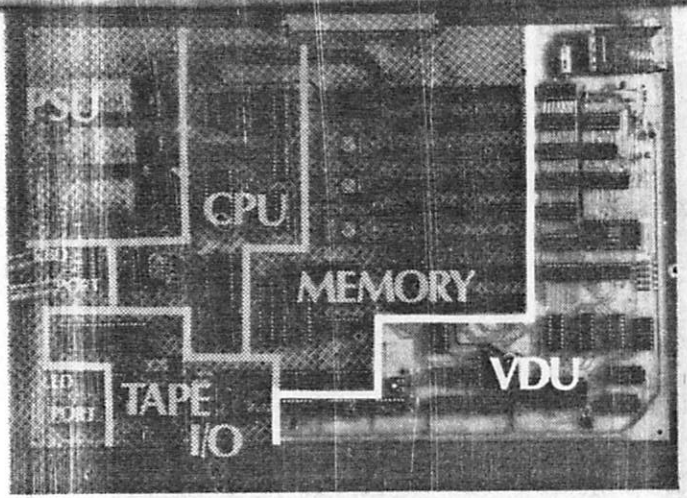
CPU

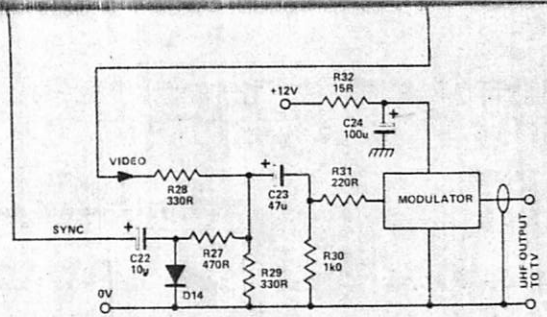

EXTENDER SOCKET
HDLA
DBIN
HOLD
RESET
INTE
WAIT
82 (TTL)
RDYIN
0V
+5V
-5V
+12V
LINK, REMOVE FOR DMA
C19 47p
X1 7.20MHz
IC5
IC4
PAUSE
R3 1k0
R2 10k
PB4
RESIN
+12V
PB1 RESET
C20 47u
READY
RESET
SYNC
TO UART PIN 21
INT
(INTERUPT REQUEST)
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
A15
IC7
IC8
IC13a
WR
HDLA
DBIN
IC6
D1
D2
D3
D4
D5
D6
D7
D8
MEMR
MEMW
I/OR
I/OW
IC10
IC9
DB1
DB2
DB3
DB4
DB5
DB6
DB7
DB8
INTA
STATUS STROBE
ADRESS BUS TO SYSTEM AND EXTENDER SOCKET
NOTE: ADDRESS LINE 15 IS INVERTED!
CONTROL BUS
DATA BUS
STATUS STROBE (TO VDU CHANGEOVER LATCH)
NOTE:
IC4 IS 8224N
IC5 IS 8080A
IC6 IS 8228N
IC7,8,10 ARE 74LS244
IC9 IS 74LS245
IC11 IS 74LS148
IC12 IS 74LS240
IC13 IS 74LS00
R4 R5 R6 R7 R8 R9 R10 R11
ALL 1k0
PB2 CLEAR SCREEN
PB3 INITIALISE
INT1
INT2
INT4
INT8 (NOT USED)
INT3 (SPARE TO KEYBOARD)
IC11
ID1
ID2
ID3
IC12

CPU

PROJECT: Computer







VDU section of the Triton.

HOW IT WORKS

IC61, the Thomson CFS VDU control integrated circuit, has a built in clock which generates standard TV synchronisation pulses (line and field sync) on pin 26. Random interlace is used and a simplified field sync train is generated as opposed to the full CCIR specification.

The chip, synchronously with this train of pulses, generates addresses for the VDU RAM so that the correct code of the character is selected as the TV raster spot is traversing the respective part of the television screen. An external "Picture Point Oscillator" (IC55c and d) in conjunction with a divider chain (IC63) sets the horizontal width of a character and steps the address of the control chip, output from pin 12(IC63) to pin 9 (IC61). The inverted output of IC63 pin 15 is used to latch the data being addressed by the controller into IC68 (a seven wide latch), latch the picture point pattern generated by the character generator ROM into the serialiser (IC72) and reset the picture point divider chain (IC63) at the end of each character width.

The picture point width (hence the character width and number of characters per line) is set by the frequency of the oscillator control RV1.

We are using a 7 bit wide RAM to hold the FULL ASCII code — we need this to provide capacity for graphics. The outputs of the latches feed both the standard alpha-numeric character generator (IC69) and a specially programmed ROM (IC70) which contains picture point data for the 64 graphic symbols. We use the EXCLUSIVE OR function (IC62d) on bits 6 and 7 of the ASCII code to select either the graphics or alpha-numeric ROM. The select signals go through further gating (ICs67a and d) to ensure that the integrity of the cursor generating pulse (pin 15 of IC61) is not corrupted.

Three further address lines from the VDU controller (pins 11, 12 and 13) address the picture point data ROWS in both ICs 69 and 70. Due to a limitation caused by the internal operation of IC61 chip the row address code 000 is output for the top row and the bottom four rows of the character cell. Normally rows 0, 8, 9, 10 and 11 are used to provide inter line gaps for alpha-numeric displays while rows 1 to 7 carry alpha-numeric picture point data. We have had to take this into account when designing the font of graphics symbols — some of which cannot fill the complete character cell rectangle on the screen. Look at the table of graphics characters and you can see how we have adjusted the graphics to suit this restriction.

Further complications caused by this limitation are that a graphic must not appear on the topmost line of the television screen if that graphic contains picture points in its top row. IC61 requires there to be zeros present here in order to derive field blanking. This problem could be overcome with extra gating but this would have been at the expense of simplicity.

A similar problem (involving line blanking) is resolved by gating the video output with the INI function (pin 26 of IC61) in IC71b. Without this any graphics symbol having a picture point in its most left hand column would have caused a "wrap around" while a line that interferes with the DC level of the line sync pulse. The only problem that remains in this respect is that you will now get a single "extra" picture point showing to the right of the 64th character down a line if you use a graphic in the most left hand position of a line. This does not happen with all graphics — only those that have picture points in their most left hand column.

The five outputs from the alpha-numeric ROM are wire ORED with five of the eight outputs from the graphics ROM and held high via pull up resistors R22 - 26. They are then fed to the correct positions in the serialiser shift register IC72. Note that the remaining three outputs from the graphics ROM have to be ANDED with a signal defining whether or not the character is a graphic (done by ICs71a, c and d). This is to ensure that if alpha-numerics are printed there is a correct inter-character gap.

So far we have avoided talking about how the VDU RAM is addressed by the control chip. Let's deal with that now.

We are allowing the CPU to memory map the VDU RAM. To do this we have had to allow the MPU to take over addressing control of the VDU RAM. This is done by taking all the address lines from IC61 and their equivalents from the system's busbar to a set of data selectors (ICs64, 65 and 66). If the MPU addresses the VDU memory location (any address between 1000H and 13FFH) the block select line (MAP VDU) is activated. This of course, could happen if ever the address busbar went into a high impedance state (during HOLD etc) so to prevent any spurious pulses affecting the operation we gate the VDU block select line with $\overline{\text{STSTB}}$ which only occurs when valid address information is on the busbar. We do the gating in a D type latch so that during the complete cycle of a VDU memory map the data selectors are set to allow the computer address bus to be transmitted to the inputs of the VDU RAM. At the end of that cycle and at all other times the data selectors hand over address control to IC61.

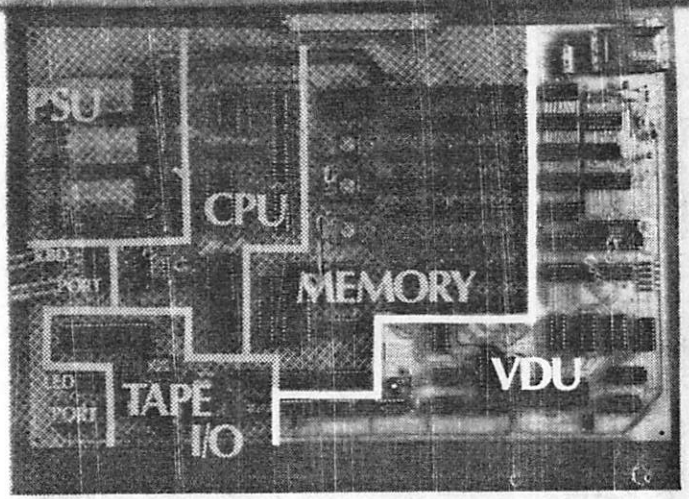
A similar transfer of responsibility takes place between the normal input data to the VDU (which gets to it via an output port) and the main system data bus. In this case the data is selected by ICs59 and 60. These also receive their changeover instruction from the changeover latch IC53. Note that we also have to do a changeover between the internally generated memory write command (pin 17 of IC61) and the MPU's $\overline{\text{MEMW}}$ strobe. This is done within IC60.

It only remains to describe the gates on the VDUs internal data lines and IC54. The former are used to force the ASCII code for "Space" on to the data lines when pin 13 (IC61) is at "0" in coincidence with a writing pulse to the VDU memory. This is to allow for the very useful internal function provided by the IC61 to clear the screen and reset the cursor in one operation.

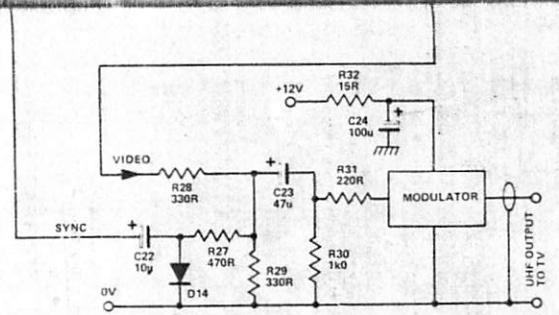
The VDU controller carries out a number of non writing functions as well as entering and addressing data within its memory. By using some of the ASCII codes as control it is possible to do such things as move the cursor in steps to any position on the screen, reset the cursor, carry out a line feed or do a carriage return clearing only the unused part of the line. There are also a couple of control codes that we wish the VDU to ignore — OOH and O4H — respectively these are NUL (or no operation) and EOT (end of text) flags. Recognition of all these special codes is carried out by the VDU CONTROL ROM (IC54). This has had to be specially programmed for the TRITON.

To get best use from the TRITON and its VDU you need to know hexadecimal and decimal values of all the ASCII codes that are used to generate alpha-numerics, graphics, and control characters. You also need to know which of the keyboard keys correspond to each graphic character. To help you we show all the graphics with their respective codes and key names in Fig. 00. Alpha-numeric codes are shown in Fig. 00 and the control codes in Fig. 00.

Normally you may output a character to the VDU for printing in I/0 mode every 8.3mS. The standard TRITON monitor errs on the safe side and has a built in delay which outputs a character roughly every 9mS. If ever you write your own software you must take this speed limitation into account. Furthermore there are two I/0 operations which take a considerably longer time: these are "Clear Screen and Home Cursor" and "Home Cursor". These instructions must be followed by a delay of at least 132mS. Again the TRITON's monitor makes allowance for this but you can get direct access to these functions if you use either the "PRINT CONTROL" or "VDU" commands which exist in BASIC L4.1. If you use these in BASIC you MUST follow them with a delay loop having a time constant greater than 132mS. (In practice we found that a 200 step "FOR — NEXT" instruction was quite safe.)



VDU section of the Triton.



HOW IT WORKS

IC61, the Thomson CFS VDU control integrated circuit, has a built in clock which generates standard TV synchronisation pulses (line and field sync) on pin 26. Random interlace is used and a simplified field sync train is generated as opposed to the full CCIR specification.

The chip, synchronously with this train of pulses, generates addresses for the VDU RAM so that the correct code of the character is selected as the TV raster spot is traversing the respective part of the television screen. An external "Picture Point Oscillator" (IC55c and d) in conjunction with a divider chain (IC63) sets the horizontal width of a character and steps the address of the control chip, output from pin 12(IC63) to pin 9 (IC61). The inverted output of IC63 pin 15 is used to latch the data being addressed by the controller into IC68 (a seven wide latch), latch the picture point pattern generated by the character generator ROM into the serialiser (IC72) and reset the picture point divider chain (IC63) at the end of each character width.

The picture point width (hence the character width and number of characters per line) is set by the frequency of the oscillator control RV1.

We are using a 7 bit wide RAM to hold the FULL ASCII code — we need this to provide capacity for graphics. The outputs of the latches feed both the standard alpha-numeric character generator (IC69) and a specially programmed ROM (IC70) which contains picture point data for the 64 graphic symbols. We use the EXCLUSIVE OR function (IC62d) on bits 6 and 7 of the ASCII code to select either the graphics or alpha-numeric ROM. The select signals go through further gating (ICs67a and d) to ensure that the integrity of the cursor generating pulse (pin 15 of IC61) is not corrupted.

Three further address lines from the VDU controller (pins 11, 12 and 13) address the picture point data ROWS in both ICs 69 and 70. Due to a limitation caused by the internal operation of IC61 chip the row address code 000 is output for the top row and the bottom four rows of the character cell. Normally rows 0, 8, 9, 10 and 11 are used to provide inter line gaps for alpha-numeric displays while rows 1 to 7 carry alpha-numeric picture point data. We have had to take this into account when designing the font of graphics symbols — some of which cannot fill the complete character cell rectangle on the screen. Look at the table of graphics characters and you can see how we have adjusted the graphics to suit this restriction.

Further complications caused by this limitation are that a graphic must not appear on the topmost line of the television screen if that graphic contains picture points in its top row. IC61 requires there to be zeros present here in order to derive field blanking. This problem could be overcome with extra gating but this would have been at the expense of simplicity.

A similar problem (involving line blanking) is resolved by gating the video output with the INI function (pin 26 of IC61) in IC71b. Without this any graphics symbol having a picture point in its most left hand column would have caused a "wrap around" while a line that interferes with the DC level of the line sync pulse. The only problem that remains in this respect is that you will now get a single "extra" picture point showing to the right of the 64th character down a line if you use a graphic in the most left hand position of a line. This does not happen with all graphics — only those that have picture points in their most left hand column.

The five outputs from the alpha-numeric ROM are wire ORED with five of the eight outputs from the graphics ROM and held high via pull up resistors R22 - 26. They are then fed to the correct positions in the serialiser shift register IC72. Note that the remaining three outputs from the graphics ROM have to be ANDED with a signal defining whether or not the character is a graphic (done by ICs71a, c and d). This is to ensure that if alpha-numerics are printed there is a correct inter-character gap.

So far we have avoided talking about how the VDU RAM is addressed by the control chip. Let's deal with that now.

We are allowing the CPU to memory map the VDU RAM. To do this we have had to allow the MPU to take over addressing control of the VDU RAM. This is done by taking all the address lines from IC61 and their equivalents from the system's busbar to a set of data selectors (ICs64, 65 and 66). If the MPU addresses the VDU memory location (any address between 1000H and 13FFH) the block select line (MAP VDU) is activated. This of course, could happen if ever the address busbar went into a high impedance state (during HOLD etc) so to prevent any spurious pulses affecting the operation we gate the VDU block select line with $\overline{\text{STSTB}}$ which only occurs when valid address information is on the busbar. We do the gating in a D type latch so that during the complete cycle of a VDU memory map the data selectors are set to allow the computer address bus to be transmitted to the inputs of the VDU RAM. At the end of that cycle and at all other times the data selectors hand over address control to IC61.

A similar transfer of responsibility takes place between the normal input data to the VDU (which gets to it via an output port) and the main system data bus. In this case the data is selected by ICs59 and 60. These also receive their changeover instruction from the changeover latch IC53. Note that we also have to do a changeover between the internally generated memory write command (pin 17 of IC61) and the MPU's $\overline{\text{MEMW}}$ strobe. This is done within IC60.

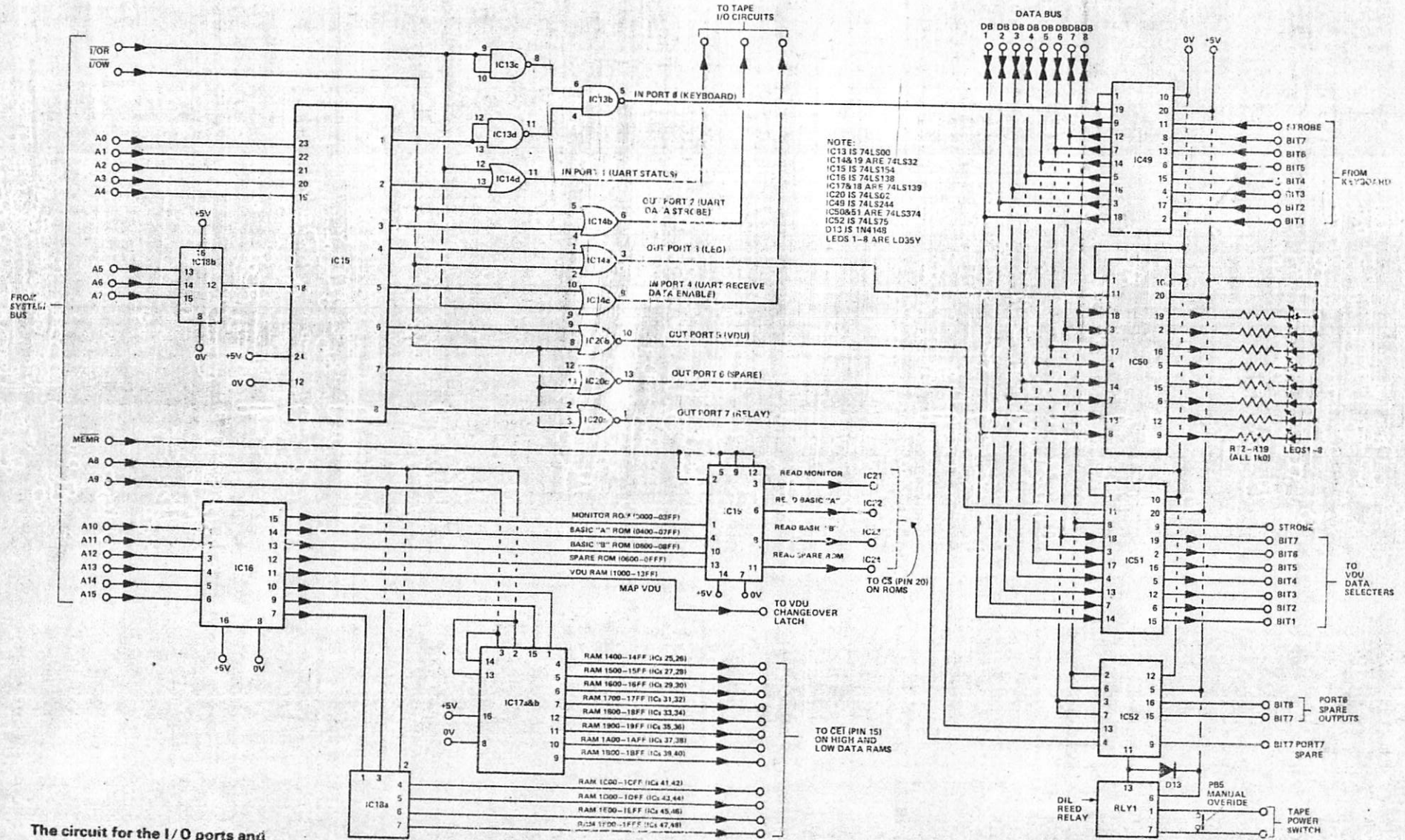
It only remains to describe the gates on the VDUs internal data lines and IC54. The former are used to force the ASCII code for "Space" on to the data lines when pin 13 (IC61) is at "O" in coincidence with a writing pulse to the VDU memory. This is to allow for the very useful internal function provided by the IC61 to clear the screen and reset the cursor in one operation.

The VDU controller carries out a number of non writing functions as well as entering and addressing data within its memory. By using some of the ASCII codes as control it is possible to do such things as move the cursor in steps to any position on the screen, reset the cursor, carry out a line feed or do a carriage return clearing only the unused part of the line. There are also a couple of control codes that we wish the VDU to ignore — OOH and O4H — respectively these are NUL (or no operation) and EOT (end of text) flags. Recognition of all these special codes is carried out by the VDU CONTROL ROM (IC54). This has had to be specially programmed for the TRITON.

To get best use from the TRITON and its VDU you need to know hexadecimal and decimal values of all the ASCII codes that are used to generate alpha-numerics, graphics, and control characters. You also need to know which of the keyboard keys correspond to each graphic character. To help you we show all the graphics with their respective codes and key names in Fig. 00. Alpha-numeric codes are shown in Fig. 00 and the control codes in Fig. 00.

Normally you may output a character to the VDU for printing in I/0 mode every 8.3mS. The standard TRITON monitor errs on the safe side and has a built in delay which outputs a character roughly every 9mS. If ever you write your own software you must take this speed limitation into account. Furthermore there are two I/0 operations which take a considerably longer time: these are "Clear Screen and Home Cursor" and "Home Cursor". These instructions must be followed by a delay of at least 132mS. Again the TRITON's monitor makes allowance for this but you can get direct access to these functions if you use either the "PRINT CONTROL" or "VDU" commands which exist in BASIC L4.1. If you use these in BASIC you MUST follow them with a delay loop having a time constant greater than 132mS. (In practice we found that a 200 step "FOR — NEXT" instruction was quite safe.)

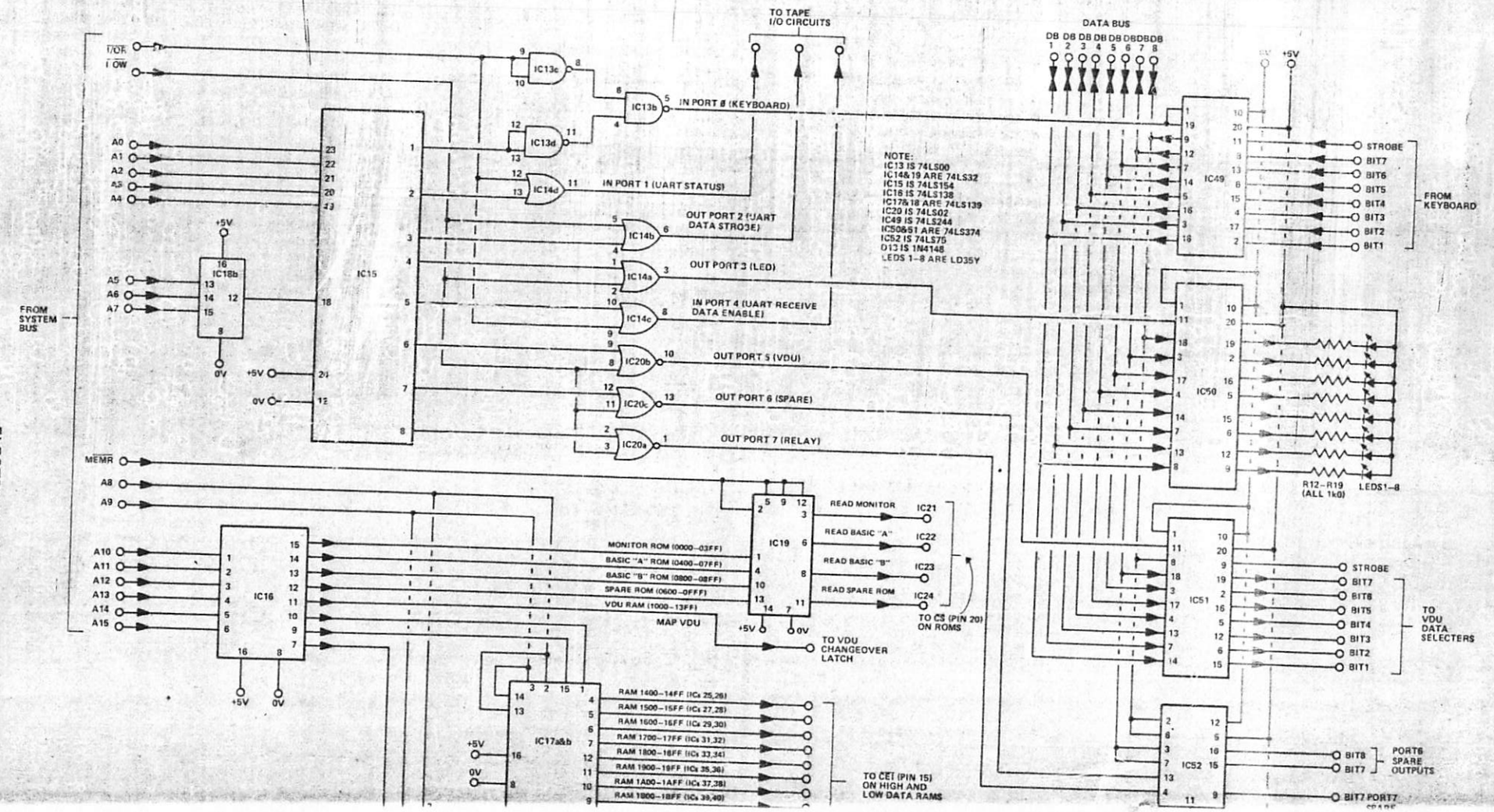
KBD PORT



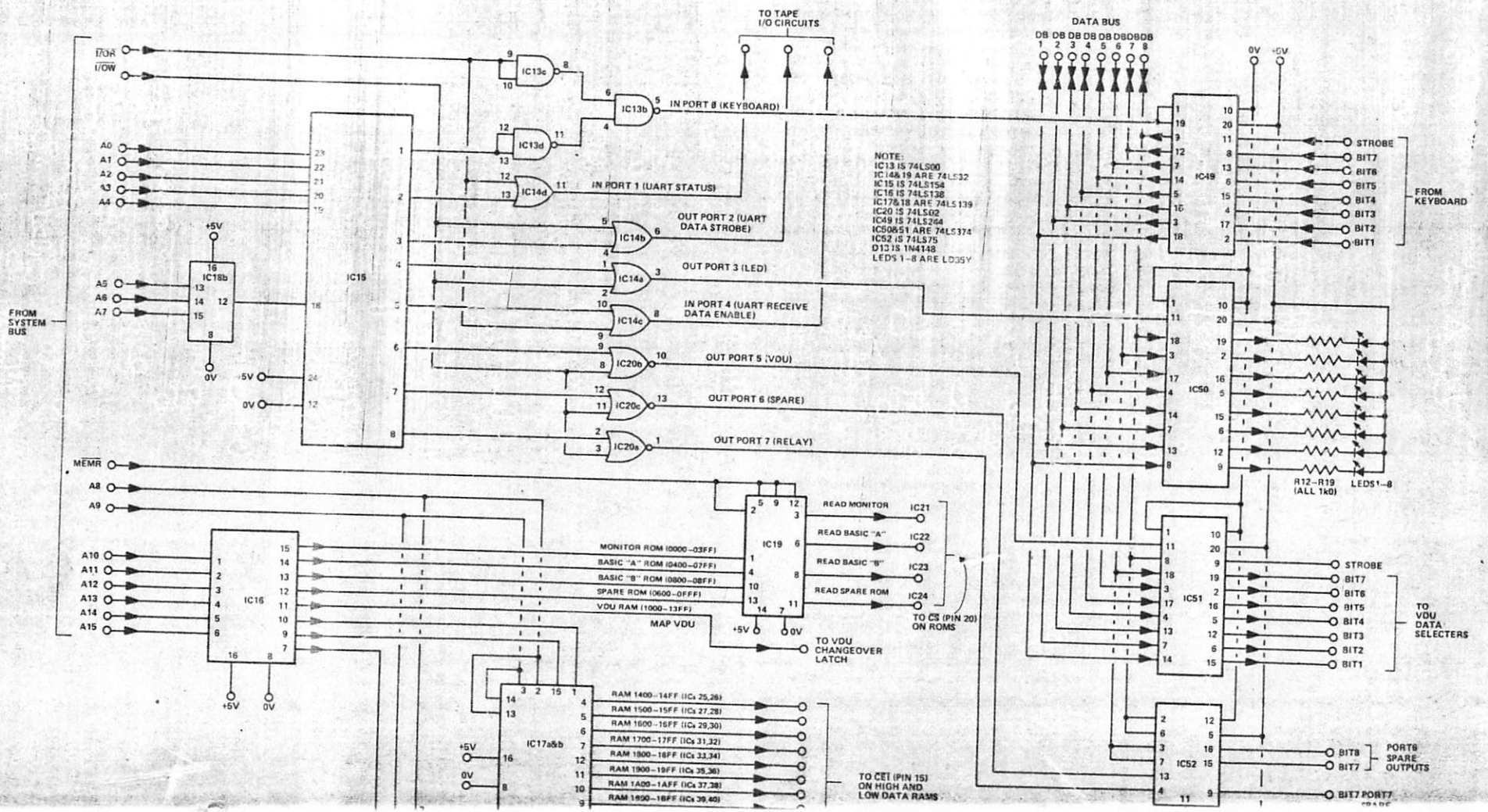
The circuit for the I/O ports and memory select.

KBD PORT

PROJECT: Computer



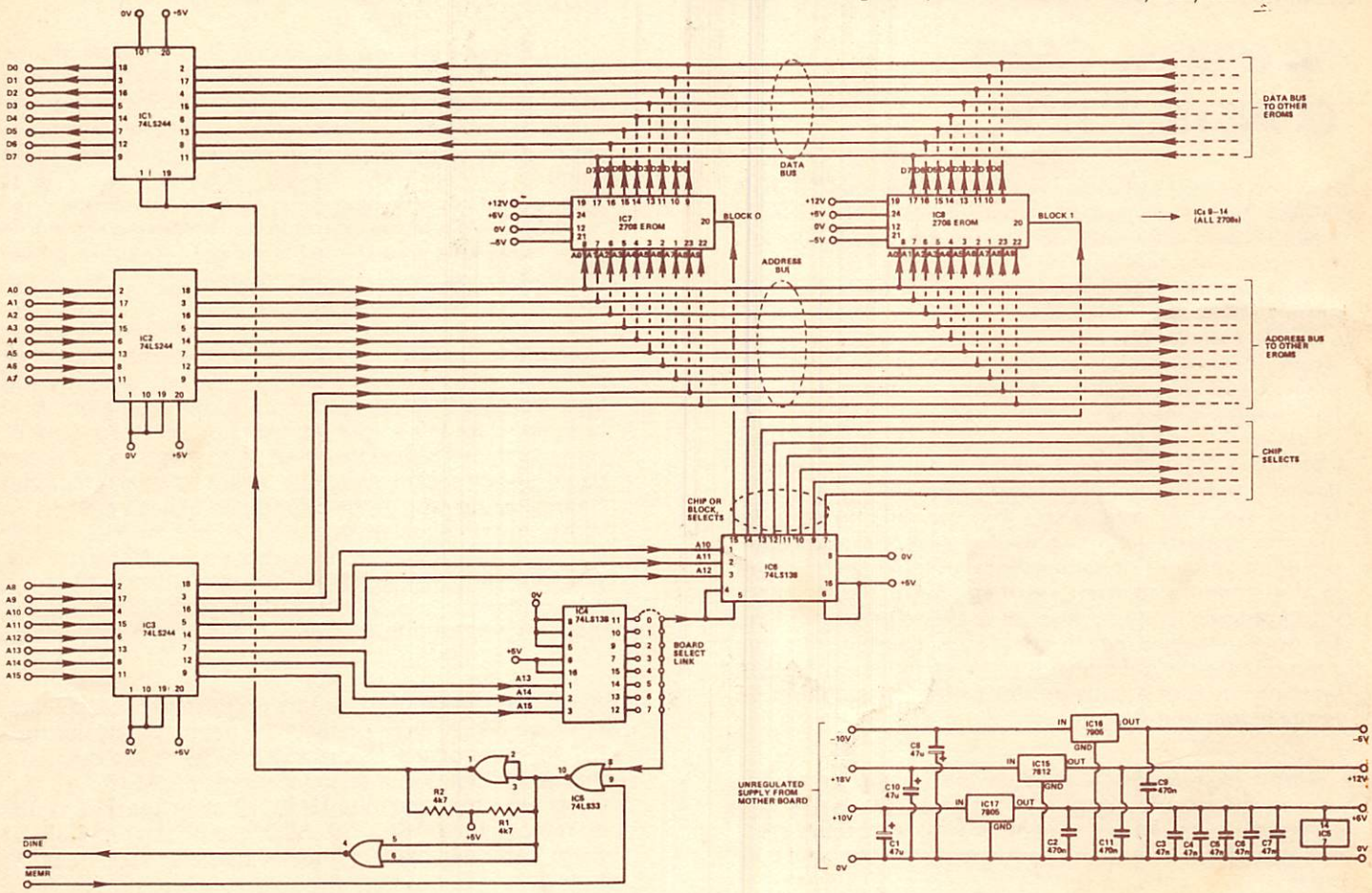
KBD PORT



With the EROM card plugged into the motherboard apply power to it and then switch Triton on. Press L for LIST and then enter the starting address of the Region that you have selected for the EROM card (e.g. for Region 2 you should enter 4000 and for Region 7 the start address would be E000). When you initiate LIST you should see the same data that you had originally noted. By carefully calculating the starting address of each block on the EROM card you can use BASIC "A" to test out each EROM socket in this manner.

When you have finished your tests do not forget to replace BASIC "A" back into its correct socket and give it a quick run to check that all is well.

At the time of writing we understand that Transam Components Ltd. will be offering a much larger and more versatile BASIC which could be housed on this board. This board, together with the 8K RAM card make Triton an extremely powerful machine and there is no reason (apart from financial!) why you should limit yourself to a single EROM card. Provided you use unallocated Regions you can add as many as you like.



HOW IT WORKS

The object of this board is to provide facility for the Triton to carry a much larger degree of resident firmware than is catered for on the main board. This extension card will provide space for a full 8K of ultra-violet erasable read only memory (using 2708 chips) which could be used to carry a much larger BASIC Interpreter or, maybe, an Assembler.

The board's position within the 64K memory map of Triton can be selected with a board select link to be any 8K region. Memory organisation on the board is in 1K blocks — each block contained in a single 2708 EROM.

It circuit is designed to interface directly with the Triton motherboard busbar from which it draws power to provide the three regulated voltage rails (+12V, +5V and —5V). The board also generates the correct DINE signal to control the enabling of the motherboard data bus buffer.

If you have already made the 8K RAM extension card you will notice a close similarity in this circuit. ICs 2 and 3 buffer the address bus on to the board and the address lines used exactly match the data sheet designations for the 2708 — there are, therefore, no ambiguities in address line nomenclature. IC1 is a similar 8 bit buffer which carries data coming from the EROMs and applies this to the motherboard busbar when it is enabled. As this card contains *only* ROM this latter buffer need only be unidirectional. Its output is enabled when the board select decoder recognises that its address is being interogated by the computer.

IC4 decodes the 8 memory regions of the 64K memory map and provides eight options for the board's starting address. These are user selectable by means of a jumper lead. When this board select signal is gated with MEMR it carries out three functions: (a) enables the output of the data bus buffer, (b) provides the DINE signal and (C) enables IC6 which decodes which block of 1K is being selected on the board.

Although the board will carry eight 2708 chips it is not necessary to have a full complement on board to start with. Care should be taken to ensure that the right EROM is plugged into its correct designation block number otherwise major computer brainstorms will ensue! IC7 corresponds to block "O" which is the lowest order address position on the board; IC8 is the next higher and so on.

Although you can position this board within any 8K region in Triton's memory architecture you must avoid region "O" which is completely spoken for by the main board. You must, also, make sure that you do not use a region that is already allocated to one or more RAM cards. We would suggest that any large scale firmware should occupy the top end of memory to prevent it interfering with the continuity of the RAM but this is for the individual user to choose.

PARTS LIST

RESISTORS (all ¼W 5%)

R1, 2	4k7

CAPACITORS

C1, 8, 10	47u 25V electrolytic
C2, 9, 11	470n polyester
C3-7	47n ceramic

SEMICONDUCTORS

IC1, 2, 3	74LS244
IC4, 6	74LS138
IC5	74LS33
IC7-14	2708 EROMs suitably programmed
IC15	+12V 1A regulator
IC16	—5V 1A regulator
IC17	+5V 1A regulator

MISCELLANEOUS

PCB (Transam Components Ltd.)
64 pin Eurosocket (right angled pins)
heatsink
3 off 20 pin DIL sockets
3 off 16 pin DIL sockets
1 off 14 pin DIL sockets
8 off 24 pin DIL sockets
1 off 16 pin DIL header

TRITON 8K EPROM CARD

8K EPROM CARD CONSTRUCTION

A prime design criterion for the whole of our TRITON project has been ease of construction. This EPROM extender card is no exception and needs very few words to help you on your way.

It is wise to start by soldering in all the integrated circuit sockets — ONE AT A TIME — checking all soldered connections as you go. Start, as usual, with the larger sockets and work downwards in size; this will prevent you putting a small socket in where it should have been a large one!

When soldering hold the iron in place slightly longer than for a single sided PCB. This will allow the solder to flow through the hole reinforcing the plating through. Do not, however, overdo this and certainly do not hold the iron in place for more than 3 seconds. Apply the minimum amount of solder that will make a clean wet joint. Remember to insert a sixteen pin DIL socket where the board select jumper is located. Go on to solder in the 64 pin Eurosocket which should be inserted from the component side of the board. Its fixing holes might need opening up with a suitable drill and then it should be firmly bolted to the board.

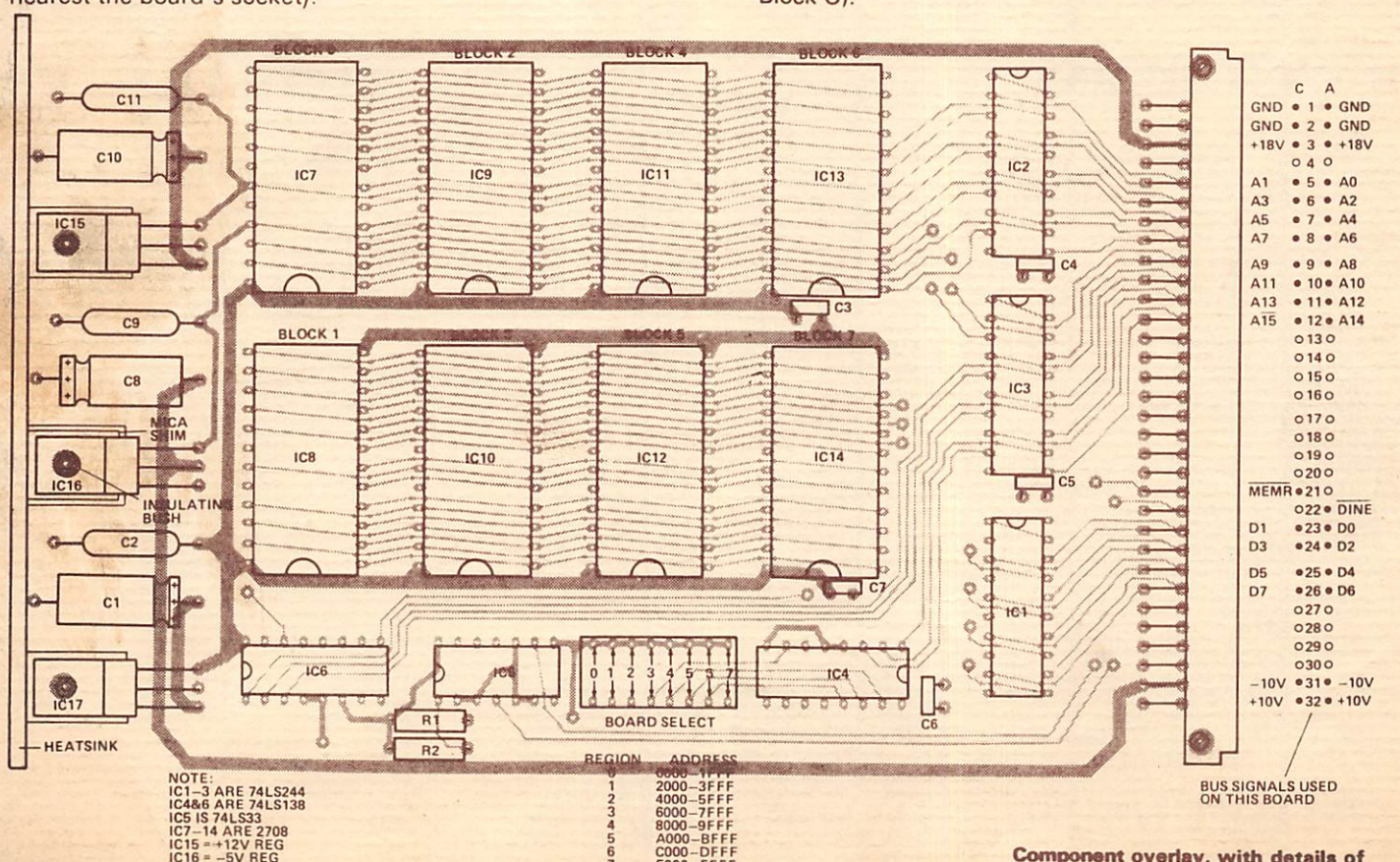
Insert all resistors and capacitors giving particular note to the orientation of C1, C8 and C10 — C8 is a different way round from the other two! Be careful, also, that you insert C7 into the correct pair of holes (the pair nearest the board's socket).

For obvious reasons do not mix up the three voltage regulators (ICs 15 to 17). Hold them so that their fixing holes line up with the large holie in the PCB and then put a right angled form on their leads so that they will neatly pass through their respective connection holes. Allow about $^1/_{16}$" gap between their backs and the PCB to allow space for the heat sink tabs to be sandwiched between them and the board. When soldering them into place make sure that the leads do not touch the power distribution tracks which they bridge over. Note that a mica shim and insulating bush is needed for IC16.

The heatsink should be cut and bent from 14 gauge aluminium. Before drilling holes in it slide it into place and mark drilling centres through the regulators' fixing tabs. Doing it this way will ensure a perfect matching.

Finally solder a link across the selected pair of contacts on a 16 pin DIL header and insert this into the board select socket and then insert all the integrated circuits taking note of the orientation of each one.

At the time of testing you might not have any firmware available to run on this board. None-the-less you can still test the board through Triton's Monitor. All you need is a 2708 EROM with some known program written into it. Every Triton user has such an EROM that can be used for this purpose in the shape of one of the BASIC chips on the main board. Firstly use the LIST function of the Monitor and write down the contents of the first dozen or so locations of BASIC "A" (to do this LIST from location 0400). Power the Triton off and remove the BASIC "A" EROM and insert it into the EROM extender card in the location for IC7 (that is Block 0).



NOTE:
IC1–3 ARE 74LS244
IC4&6 ARE 74LS138
IC5 IS 74LS33
IC7–14 ARE 2708
IC15 = +12V REG
IC16 = –5V REG
IC17 = +5V REG

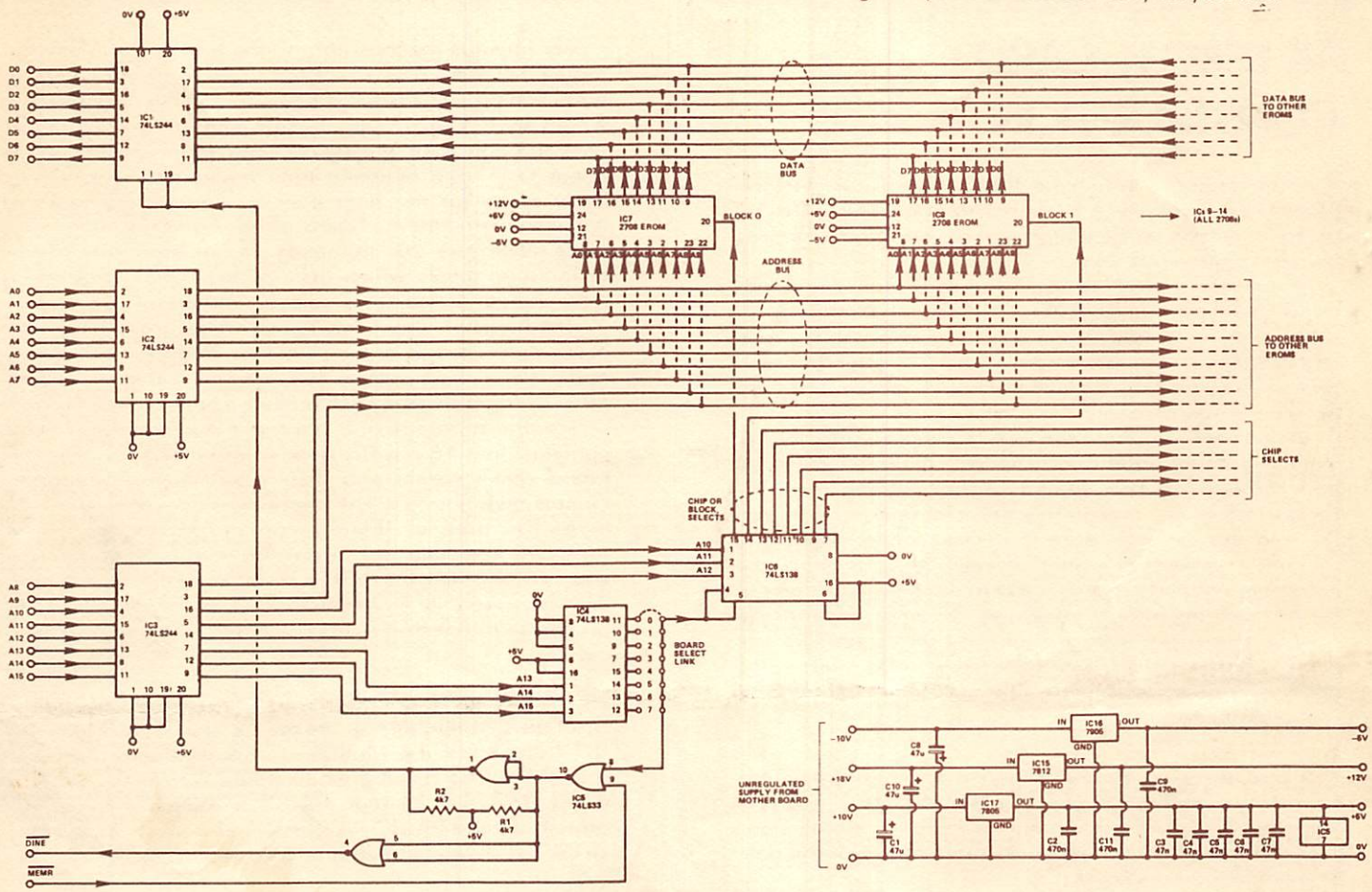
REGION	ADDRESS
0	0000–1FFF
1	2000–3FFF
2	4000–5FFF
3	6000–7FFF
4	8000–9FFF
5	A000–BFFF
6	C000–DFFF
7	E000–FFFF

Component overlay, with details of the bus signals used on the board.

With the EROM card plugged into the motherboard apply power to it and then switch Triton on. Press L for LIST and then enter the starting address of the Region that you have selected for the EROM card (e.g. for Region 2 you should enter 4000 and for Region 7 the start address would be E000). When you initiate LIST you should see the same data that you had originally noted. By carefully calculating the starting address of each block on the EROM card you can use BASIC "A" to test out each EROM socket in this manner.

When you have finished your tests do not forget to replace BASIC "A" back into its correct socket and give it a quick run to check that all is well.

At the time of writing we understand that Transam Components Ltd. will be offering a much larger and more versatile BASIC which could be housed on this board. This board, together with the 8K RAM card make Triton an extremely powerful machine and there is no reason (apart from financial!) why you should limit yourself to a single EROM card. Provided you use unallocated Regions you can add as many as you like.



HOW IT WORKS

The object of this board is to provide facility for the Triton to carry a much larger degree of resident firmware than is catered for on the main board. This extension card will provide space for a full 8K of ultra-violet erasable read only memory (using 2708 chips) which could be used to carry a much larger BASIC Interpreter or, maybe, an Assembler.

The board's position within the 64K memory map of Triton can be selected with a board select link to be any 8K region. Memory organisation on the board is in 1K blocks — each block contained in a single 2708 EROM.

It circuit is designed to interface directly with the Triton motherboard busbar from which it draws power to provide the three regulated voltage rails (+12V, +5V and —5V). The board also generates the correct DINE signal to control the enabling of the motherboard data bus buffer.

If you have already made the 8K RAM extension card you will notice a close similarity in this circuit. ICs 2 and 3 buffer the address bus on to the board and the address lines used exactly match the data sheet designations for the 2708 — there are, therefore, no ambiguities in address line nomenclature. IC1 is a similar 8 bit buffer which carries data coming from the EROMs and applies this to the motherboard busbar when it is enabled. As this card contains *only* ROM this latter buffer need only be unidirectional. Its output is enabled when the board select decoder recognises that its address is being interogated by the computer.

IC4 decodes the 8 memory regions of the 64K memory map and provides eight options for the board's starting address. These are user selectable by means of a jumper lead. When this board select signal is gated with MEMR it carries out three functions: (a) enables the output of the data bus buffer, (b) provides the DINE signal and (C) enables IC6 which decodes which block of 1K is being selected on the board.

Although the board will carry eight 2708 chips it is not necessary to have a full complement on board to start with. Care should be taken to ensure that the right EROM is plugged into its correct designation block number otherwise major computer brainstorms will ensue! IC7 corresponds to block "O" which is the lowest order address position on the board; IC8 is the next higher and so on.

Although you can position this board within any 8K region in Triton's memory architecture you must avoid region "O" which is completely spoken for by the main board. You must, also, make sure that you do not use a region that is already allocated to one or more RAM cards. We would suggest that any large scale firmware should occupy the top end of memory to prevent it interfering with the continuity of the RAM but this is for the individual user to choose.

PARTS LIST

RESISTORS (all ¼W 5%)

R1, 2	4k7

CAPACITORS

C1, 8, 10	47u 25V electrolytic
C2, 9, 11	470n polyester
C3-7	47n ceramic

SEMICONDUCTORS

IC1, 2, 3	74LS244
IC4, 6	74LS138
IC5	74LS33
IC7-14	2708 EROMs suitably programmed
IC15	+12V 1A regulator
IC16	—5V 1A regulator
IC17	+5V 1A regulator

MISCELLANEOUS

PCB (Transam Components Ltd.)
64 pin Eurosocket (right angled pins)
heatsink
3 off 20 pin DIL sockets
3 off 16 pin DIL sockets
1 off 14 pin DIL sockets
8 off 24 pin DIL sockets
1 off 16 pin DIL header

TRITON 8K EPROM CARD

8K EPROM CARD CONSTRUCTION

A prime design criterion for the whole of our TRITON project has been ease of construction. This EPROM extender card is no exception and needs very few words to help you on your way.

It is wise to start by soldering in all the integrated circuit sockets — ONE AT A TIME — checking all soldered connections as you go. Start, as usual, with the larger sockets and work downwards in size; this will prevent you putting a small socket in where it should have been a large one!

When soldering hold the iron in place slightly longer than for a single sided PCB. This will allow the solder to flow through the hole reinforcing the plating through. Do not, however, overdo this and certainly do not hold the iron in place for more than 3 seconds. Apply the minimum amount of solder that will make a clean wet joint. Remember to insert a sixteen pin DIL socket where the board select jumper is located. Go on to solder in the 64 pin Eurosocket which should be inserted from the component side of the board. Its fixing holes might need opening up with a suitable drill and then it should be firmly bolted to the board.

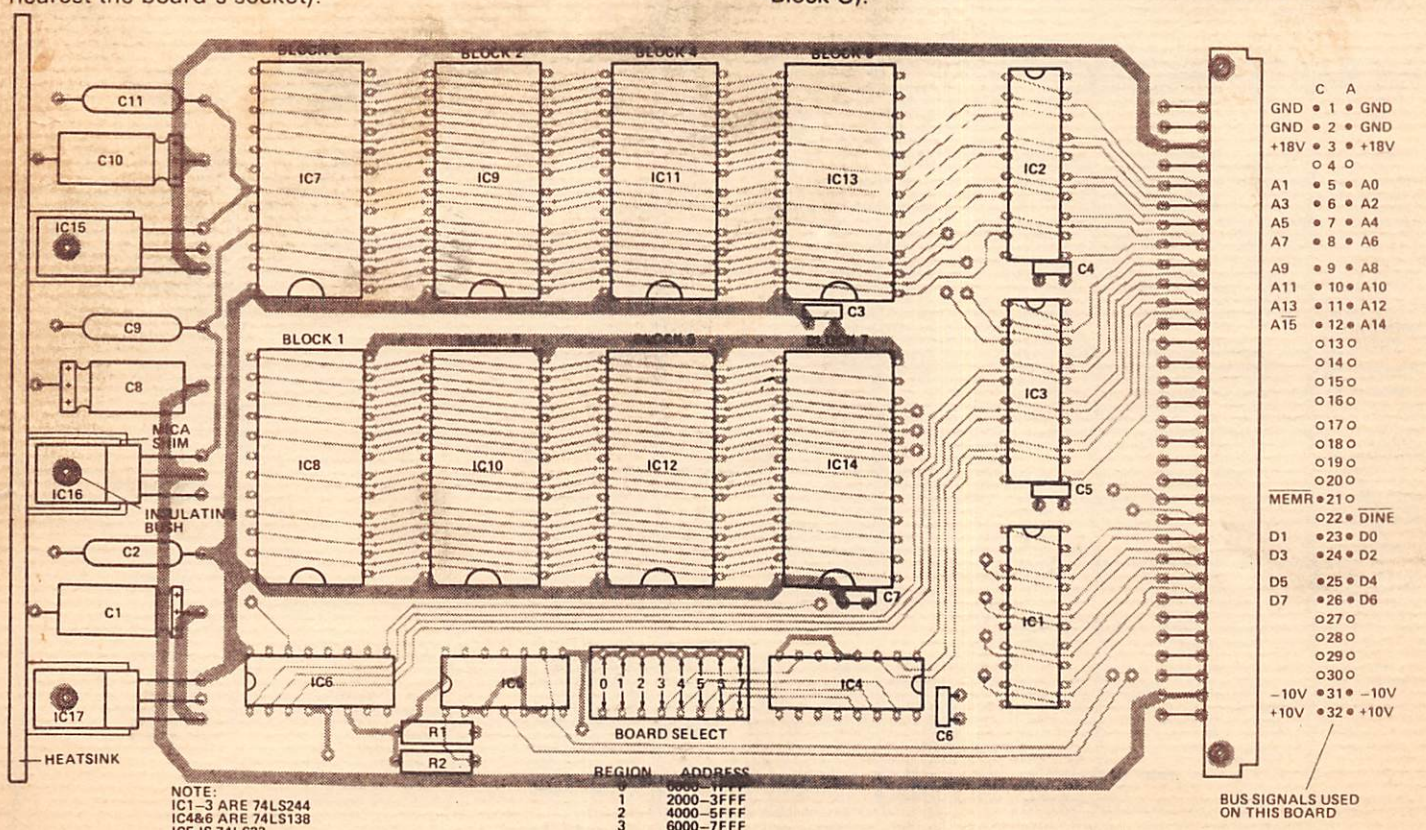
Insert all resistors and capacitors giving particular note to the orientation of C1, C8 and C10 — C8 is a different way round from the other two! Be careful, also, that you insert C7 into the correct pair of holes (the pair nearest the board's socket).

For obvious reasons do not mix up the three voltage regulators (ICs 15 to 17). Hold them so that their fixing holes line up with the large holie in the PCB and then put a right angled form on their leads so that they will neatly pass through their respective connection holes. Allow about $^1/_{16}$" gap between their backs and the PCB to allow space for the heat sink tabs to be sandwiched between them and the board. When soldering them into place make sure that the leads do not touch the power distribution tracks which they bridge over. Note that a mica shim and insulating bush is needed for IC16.

The heatsink should be cut and bent from 14 gauge aluminium. Before drilling holes in it slide it into place and mark drilling centres through the regulators' fixing tabs. Doing it this way will ensure a perfect matching.

Finally solder a link across the selected pair of contacts on a 16 pin DIL header and insert this into the board select socket and then insert all the integrated circuits taking note of the orientation of each one.

At the time of testing you might not have any firmware available to run on this board. None-the-less you can still test the board through Triton's Monitor. All you need is a 2708 EROM with some known program written into it. Every Triton user has such an EROM that can be used for this purpose in the shape of one of the BASIC chips on the main board. Firstly use the LIST function of the Monitor and write down the contents of the first dozen or so locations of BASIC "A" (to do this LIST from location 0400). Power the Triton off and remove the BASIC "A" EROM and insert it into the EROM extender card in the location for IC7 (that is Block 0).



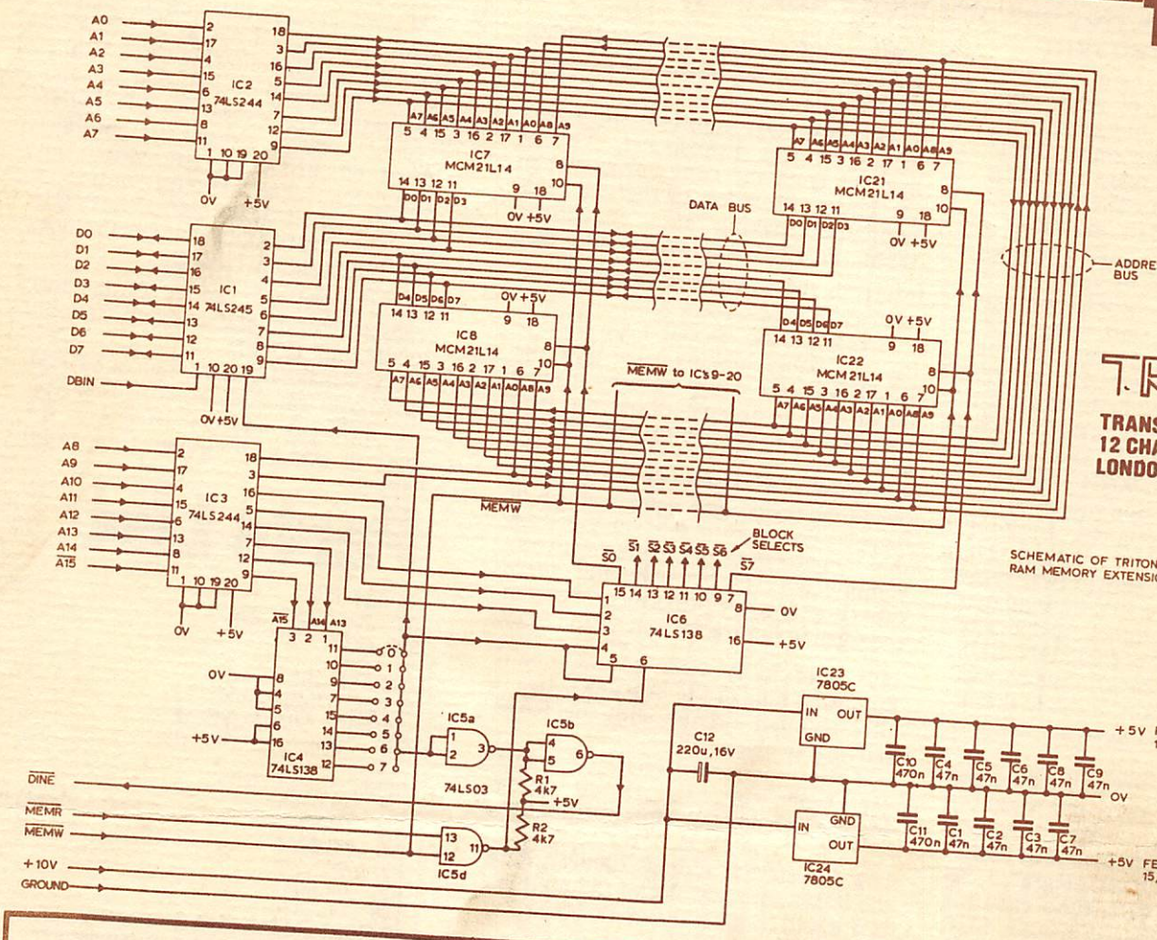
NOTE:
IC1–3 ARE 74LS244
IC4&6 ARE 74LS138
IC5 IS 74LS33
IC7–14 ARE 2708
IC15 = +12V REG
IC16 = –5V REG
IC17 = +5V REG

REGION	ADDRESS
1	2000–3FFF
2	4000–5FFF
3	6000–7FFF
4	8000–9FFF
5	A000–BFFF
6	C000–DFFF
7	E000–FFFF

Component overlay, with details of the bus signals used on the board.

TRANSAM

TRITON



SCHEMATIC OF TRITON 8k STATIC RAM MEMORY EXTENSION CARD

How It Works

IC1 is a bidirectional buffer for the 8 bit data busbar and receives and transmits data to the motherboard. Its direction of operation is controlled by the DBIN signal but its outputs are not enabled unless the board recognises its own address (decoded by IC4) — this enabling signal is a "0" on pin 19 of IC1. The 16 bits of the address busbar are buffered by ICs 2 and 3 which are permanently enabled.

The three highest order bits of the address busbar (A15, A14 and A13) are used to decode which 8K region the board sits in. There are 8 possibilities so we are using a three to eight line decoder. Only one of the eight possible outputs will go to "0" for a given high order address and we can select one of these outputs to identify the board's starting address. This is done by means of a wire link. We anticipate that all our memory extension cards will be of 8K capacity therefore it is convenient to think of the 8 possible regions of 8K being designated 0, 1, 2 etc. up to 7. The memory map of the TRITON system can then be simply defined by the following chart:

Region	*Address limits*	*Description*
0	0000H—1FFFH	TRITON main board embracing EROM, VDU and RAM
1	2000H—3FFFh	User definable
2	4000H—5FFFH	User definable
3	6000H—7FFFH	User definable
4	8000H—9FFFH	User definable
5	A000H—BFFFH	User definable
6	C000H—DFFFH	User definable
7	E000H—FFFFH	User definable

Region 0 is already spoken for by the TRITON main board so none of the extension cards must sit in this area. We would expect that most people will wish to have continguous memory (i.e. an unbroken sequence of addresses) starting from 1600H which is the start of work area on the main board so the first RAM extension card should be situated in Region 1, the next in region 2 etc. Any specialised memory could then be sited at the high order end of the map (regions 6 and 7). These comments are, of course, generalisations and individual users can arrange their memory in any order they wish using the 8K regions as building bricks.

In the case of this static RAM card we are using pairs of 1K × 4 memory chips type MCM21L14 (note we are specifying the low power version to economise on total current consumption — the higher power versions CAN be used but board dissipation might get uncomfortably near the limit of the voltage regulators). This allows us to organise the memory within the board into 8 blocks of 1K apiece. These can be conveniently and individually selected by the three next lower address lines (A12, A11 and A10) through a further three to eight line decoder (IC6). We have to use the board select signal to enable this decoder so that the blocks are selected ONLY when the board as a whole is addressed. This enabling signal is fed into pins 4 and 5 of IC6. We have designated the eight block selects signals (active low) as S0 to S7. S0 represents the lowest order block of the board so for contiguous memory you should insert memory chips into locations IC7 and 8 in the first instance and then works upwards (that is if you do not intend to put a complete load of RAM into the board in the first instance).

As the RAM chips are organised as 4 bit wide "nibbles" we have to use two Integrated circuits for each block. Odd number component designations (IC7,9 11 etc) represent the low order nibbles and even numbers (IC8, 10 12 etc) the high order.

The ten lowest order address lines are paralleled to all the RAM chips on the board in the usual way but please note that we are NOT using the data sheet's designations for the addresses. The reason for this is tied into the convenience of the board's layout. It makes no difference to the operation but some people might query why we have done this. A small amount of logic is needed to organise the MEMR and MEMW command signals to interface these with the RAM's writing and select inputs. This is provided for by the NAND gate (IC5d). The logic ensures that a select line becomes active whenever either the MEMR or MEMW lines go low but the WRITE input of the RAM is only taken low when MEMW is active.

ICs 5a and 5b take the board select signal and output this as an active low through the open collector gate (IC5b). This is the DINE (Data In Enable) which can be wire ored with similar signals from other boards on to the motherboard busbar. This is used to enable the motherboard's data bus buffer during memory or I/O read operations.

Two 1A regulators are used to provide +5V.

TRITON RAM Card

Designed by Mike Hughes, complete kits will be available from Transam

This 8K static RAM card contains 8192 contiguous bytes of read write memory organised in blocks of 1K. For reasons of economy some people might wish to expand their system in easy stages in which case the chips can be added to this board in units of 1K. The board itself can have its start point address selected by means of a jumper wire (or DIL switch) and can be positioned at the start of any 8K region in a 64K memory map. In the case of TRITON the lowest order 8K region is entirely taken up by the main board so it is assumed that the first RAM extension card should be positioned to start at 2000H to make it contiguous with existing memory.

The card is designed to plug directly into the TRITON MOTHERBOARD and draws, typically, 1A from the +10V unregulated power bus. It is designed specifically to interface with the TRITON BUS configuration and provides the correct $\overline{\text{DINE}}$ signal; no claim is made that it will operate in other systems without modification.

It is built on a specially designed double sided Euro card and assembly is extremely straightforward — average building time being about 3 hours.

Insert and solder all integrated circuit sockets one at a time ensuring that no connections are missed. Start with the larger sockets — this will prevent you accidentally soldering in an 18 pin socket where there ought to be one having 20 pins etc! We suggest using a 16 pin DIL socket to carry the board select jumper wire so that this can be altered to reposition the board's address should the need arise. Where expense is no object and for development systems you could, as we did, use plug in DIL switches in this socket.

When all sockets are in position make sure you solder in the two pull up resistors.

The next stage is to bolt and solder in the 64 way Eurosocket. This is inserted from the component side of the board. Move on to the two +5V regulators making sure you insert them the right way round and then insert all the capacitors checking the polarity of C12.

Assemble the special heat sink to the regulators making sure that the fixing bolts are really tight — quite a lot of heat is dissipated and the heat sink size is a bare minimum. If you make your own from our drawing we recommend that you paint it black to increase its radiating efficiency. Check that no parts of the heatsink touch any components (particularly the three capacitors in close proximity) or any of the topside foil tracks (one of the +5V rails goes fairly close to the board mounting hole of the heatsink). Insert all integrated circuits with the RAM starting at block "0".

To test the board make up a DIL pin header with a jumper wire bridging the "1" position (this positions the board to start at address 2000H). Plug the board into any slot of the motherboard and connect the motherboard to TRITON.

Test with Triton

Do not apply power to TRITON at this stage but switch on the motherboard's supply. Check that +5V exists on both main rails of the RAM card and wait a few minutes for the heatsink to get up to temperature. It will run fairly hot — up to about 50°C.

We will assume that you have completely filled the 8 blocks of RAM of the card for the following tests. While the motherboard is switched on you should now apply power to TRITON. (NOTE always switch the motherboard on BEFORE you switch on TRITON). You should notice a slightly longer delay before the VDU screen is cleared and you get the normal initialisation message.

If all is well you can try writing and reading within the memory region 2000H and 3FFFH.

If you make up more than one RAM board you must make sure that you use different address selections for them or else very strange things will happen!

Parts List

RESISTORS
R1 4k7 10% ¼W
R2 4k7 " "

CAPACITORS
C1-C9 47n ceramic
C10-C11 470u polyester
C12 220u 16V electrolytic

SEMICONDUCTORS
IC1 74LS245
IC2 74LS244
IC3 74LS244
IC4 74LS138
IC5 74LS03
IC6 74LS138
IC7-IC22 MCM21L14 (or similar)
IC23-IC24 7805C (+5V 1A regulators)

MISCELLANEOUS
PCB
64 way Eurosocket
Heat sink for regulators (see drawing)
3 off 20 pin DIL sockets
16 off 18 pin DIL sockets
3 off 16 pin DIL sockets
1 off 14 pin DIL socket
16 pin DIL header (for wire link)