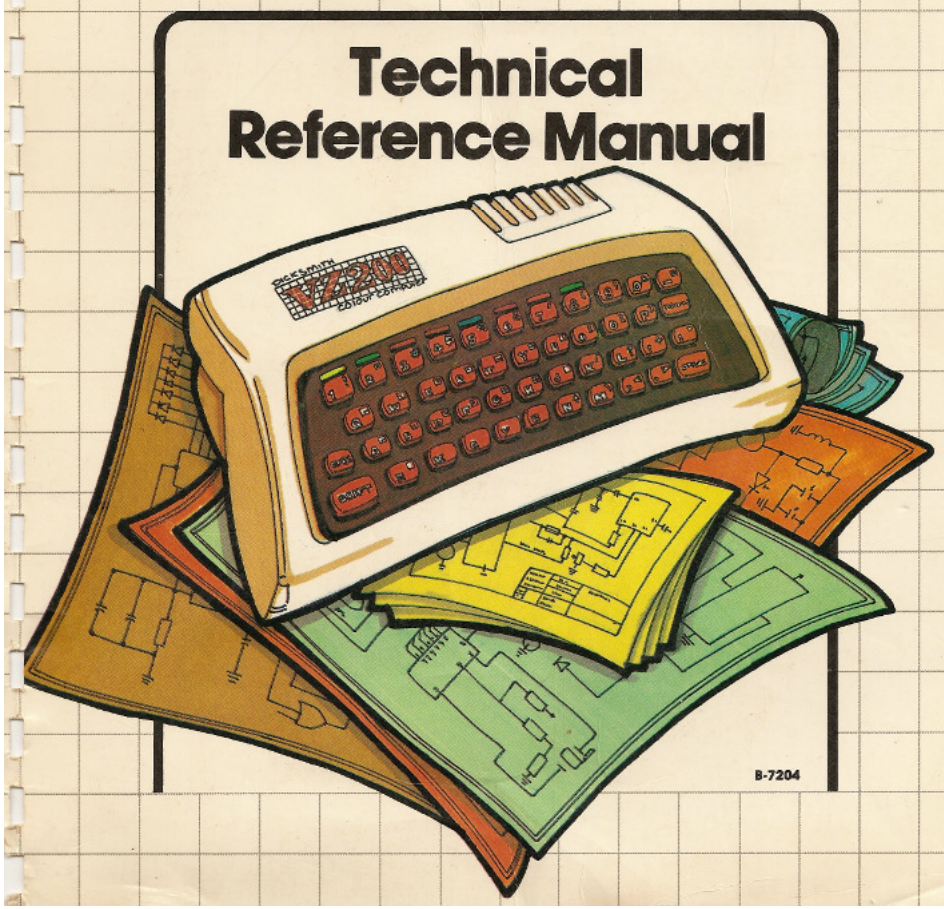
DICK SMITH

# VZ200

Personal
Colour Computer

## Technical Reference Manual




# VZ-200
# Technical Reference Manual

This manual is designed to provide owners of the Dick Smith VZ-200 Personal Colour Computer with additional information to assist in programming, operation and expansion. All reasonable care has been taken to ensure that the information herein is accurate and correct; however no responsibility can be accepted, nor liability assumed for either its accuracy or suitability for any particular purpose. Dick Smith Management Pty Ltd reserves the right to make circuit, programming and/or mechanical changes to the products described herein, without notice, in order to improve performance.

Written and compiled by Jamieson Rowe

PUBLISHED BY DICK SMITH MANAGEMENT PTY LTD
Sydney, Australia 1983

**FIRST EDITION, 1983**
**REPRINT September, 1984**
**National Library of Australia Card No. and ISBN 0 949772 18 6**



**PA865m9.84**



# List of Contents

## 1. THE BASIC COMPUTER

The basic VZ-200 computer employs a Z-80A microprocessor (U4) running at a clock speed of 3.58MHz. Two 8K x 8-bit mask-programmed ROMs (U9,U10) contain the Microsoft BASIC interpreter, while three 8K x 8-bit static RAMs (U2',U3' and U4') provide program memory.

A 6847P-1 video controller chip (U15) and a further 2K x 8-bit static RAM (U7) form the heart of the computer's video section. These are coupled to the processor data bus via an octal bidirectional buffer, U14.

A simple software scanning scheme is used for the keyboard. The keys are arranged in eight rows, each of which can be pulled down to low logic level by diodes connected to the eight least significant address lines (A0-A7). The other sides of the keys are connected to six column lines, which are connected to six of the inputs of a gated octal buffer (U12), and also to six pullup resistors. The outputs of the six corresponding outputs of U12 connect to processor data lines D0-D5.

Cassette input is handled by a simple one-transistor circuit, together with one of the remaining elements of U12 (D6). Cassette output is taken from U1, an 8-bit latch, via outputs Q1 and Q2. Other outputs from this latch are used to operate the internal piezo speaker (Q0 and Q5, used in push-pull), and to control the mode (Q3) and background color (Q4) inputs of the video controller chip.

Simplified decoding is used for selection of the various I/O devices in memory space. The decoding is performed by U2 and U3. The memory address ranges (in hex) occupied are as follows:

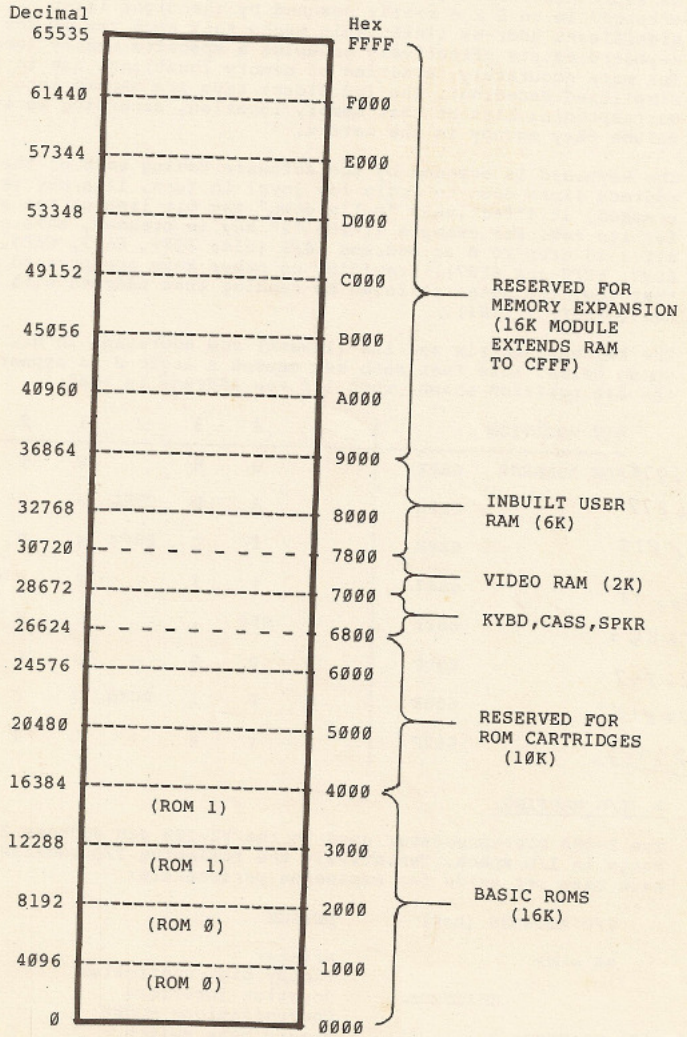
```
0000 - 1FFF:   BASIC ROM 0
2000 - 3FFF:   BASIC ROM 1
4000 - 67FF:   Not occupied in basic unit
               (Used by ROM/game cartridges)
6800 - 6FFF:   Keyboard matrix + cassette input (Read)
               Cassette output, speaker, VDC control (Write)
7000 - 77FF:   Video RAM
7800 - 8FFF:   Inbuilt program RAM
9000 - FFFF:   Not occupied in basic unit
               (Used by expansion RAM modules)
```

Note that due to the simplified addressing, latch U1 serving the cassette output, speaker and video display controller effectively occupies all addresses from 6800-6FFF inclusive. Similarly the keyboard/cassette input buffer U12 also occupies all of this address range, although the individual rows of keys effectively occupy discrete addresses.

A memory map showing both hex and decimal address is shown overleaf.



## 2. VZ-200 MEMORY MAP:



## 3. THE KEYBOARD MATRIX

As explained earlier, the 45 keys of the VZ-200 keyboard are arranged in an 8 x 6 matrix scanned by the eight least significant address lines. This means that each row of the keyboard matrix effectively occupies a specific memory location (or more accurately, a series of memory locations, due to the simplified decoding). The individual keys are mapped into the corresponding bits of that memory location, according to the column they occupy in the matrix.

The keyboard is scanned by the software taking each of the eight address lines down to logic low level in turn. If a key is pressed, it effectively "pulls down" the bit line at the address for its row. For example if the "2" key is pressed, this causes bit 1 to drop to 0 at address 68F7 (also 69F7, 6AF7, 6BF7, 6CF7, 6DF7, 6EF7 and 6FF7). Providing no other keys are pressed in that row, the data retrieved by reading that address will be 3D hex (binary 111101).

The keyboard matrix and its (lowest) row addresses in hex are shown below. Note that each key causes a logic 0 to appear at the bit position shown, when its row address is read.

| BIT POSITION | | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ROW ADDRESS | 68FE | R | Q | E | | W | T |
| | 68FD | F | A | D | CTRL | S | G |
| | 68FB | V | Z | C | SHFT | X | B |
| | 68F7 | 4 | 1 | 3 | | 2 | 5 |
| | 68EF | M | SPC | , | | . | N |
| | 68DF | 7 | 0 | 8 | - | 9 | 6 |
| | 68BF | U | P | I | RETN | O | Y |
| | 687F | J | ; | K | : | L | H |

## 4. I/O MAPPING:

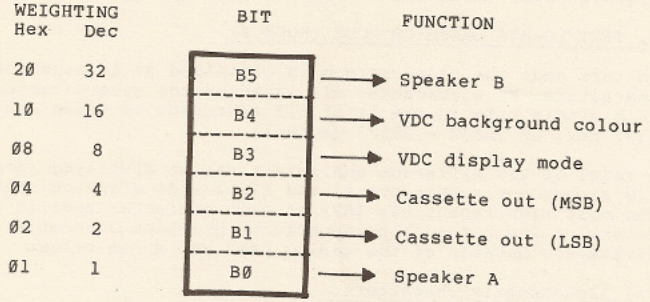
The Z-80A microprocessor used in the VZ-200 can address 256 ports in I/O space. Tentatively the following I/O address ranges have been set aside for expansion peripherals:

| I/O ADDRESS (hex) | DEVICE |
|---|---|
| 00 - 0F | Printer |
| 10 - 1F | Floppy disk controller |
| 20 - 2F | Joystick interface |
| 30 - 3F | Communications MODEM |
| 70 - 7F | Memory bank switch |



## 5. CASSETTE/SPEAKER/VDC OUTPUT LATCH:

As noted earlier, write-only latch U1 is used to provide the cassette output signal, the drive for the internal piezo speaker, and two control signals for the video display controller (VDC) chip. The latch effectively occupies all of the addresses from 6800 - 6FFF (decimal 26624-28671) inclusive. The bit map of the latch is shown below:




A. Speaker

The speaker is driven in push-pull fashion by bits 0 and 5. To make the speaker sound a note, the software should toggle bits 0 and 5 alternately at the required rate. When bit 0 is a logic "1", bit 5 should be logic "0" and vice-versa. Note that when this is done the software should not disturb the other bits of the latch.

B. Cassette output

Bits 1 and 2 are used to generate the cassette recording signal, which is approximately 175 millivolts peak-to-peak.

C. VDC display mode

The VDC display mode is controlled by bit 3. If bit 3 is a logic "0", the VDC will operate in its text/low-res mixed mode. If bit 3 is taken to logic "1", the VDC operates in its hi-res graphics only mode.

D. VDC background colour control

Bit 4 is used to control the VDC background colour. In text/low-res mode (mode 0), a "0" in bit 4 gives a green background colour while a "1" in bit 4 gives an orange background. In hi-res mode (mode 1) a "0" in bit 4 gives a green background, while a "1" gives a buff background.

## 6. VIDEO DISPLAY MODES

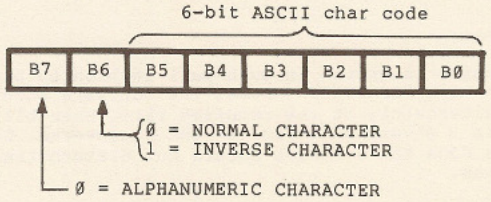
The VZ-200 has a total of 2K bytes of video display RAM, located in memory addresses 7000 - 77FF hex (28672 - 30719 decimal) inclusive. When the video display controller (VDC) chip is operating in its text/lo-res mode (mode 0), only the first 512 bytes of this video RAM are used. This is the default mode, entered on power-up. When the VDC is switched to its hi-res graphics mode (mode 1), all of the video RAM is used.

### A. TEXT/LO-RES GRAPHICS MODE (MODE 0)

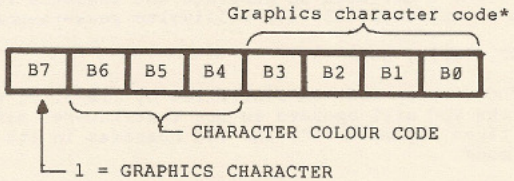
In this mode the video screen is organised as 16 lines of 32 characters. Each character displayed on the screen corresponds to a single byte in the first 512 addresses of video RAM (7000 - 71FF hex, or 28672 - 29183 decimal).

A total of 256 different characters can be displayed in mode 0: 128 alphanumeric characters, and 128 lo-res graphics characters. The most significant bit (B7) of each character code is used to determine whether an alphanumeric or graphics character is displayed. Details of the coding used are shown below:

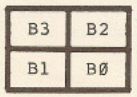
(i) Alphanumeric characters:




(ii) Programmable (lo-res) graphics characters:




*The graphics character coding used is quite simple. Each bit corresponds to one cell of a 2 x 2 pixel matrix which occupies the same screen area as an alphanumeric character:



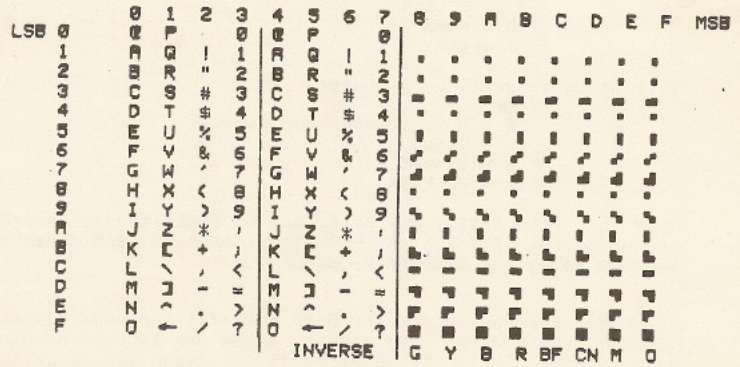



The 3-bit colour code is as follows:

| B6 | B5 | B4 | HEX | COLOUR |
|---|---|---|---|---|
| 0 | 0 | 0 | 00 | Green |
| 0 | 0 | 1 | 10 | Yellow |
| 0 | 1 | 0 | 20 | Blue |
| 0 | 1 | 1 | 30 | Red |
| 1 | 0 | 0 | 40 | Buff |
| 1 | 0 | 1 | 50 | Cyan |
| 1 | 1 | 0 | 60 | Magenta |
| 1 | 1 | 1 | 70 | Orange |

To summarise, the total mode (0) character set available is shown below. Note that these are the characters displayed if the codes concerned are written directly into the screen RAM -- i.e., using POKE statements or from a machine code routine.

VZ-200 SCREEN CHARACTER SET

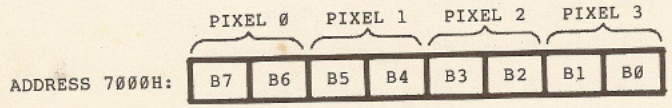



Note that when the PRINT CHR$ (N) command is used, rather than POKE, the BASIC interpreter performs code conversion to make the display conform more closely to normal ASCII standards. Hence decimal codes 0 - 31 become non-printing control codes, while codes 64-95 become those for the normal alphabetic capitals. As there are no lower-case characters available, codes 96-127 duplicate the characters for codes 32-63. Codes 192-255 are used for the inverse alphanumerics, and codes 128-143 for graphics (with colour set by the COLOR M,N command).

## B. HI-RES GRAPHICS MODE (MODE 1)

In this mode the screen is organised as 64 rows of 128 pixels, giving a total of 8192 pixels. Each pixel can be displayed in one of four colours, one of which is the background colour. This means that for each of the two possible background colours, each pixel can be either "turned off" (i.e., the same colour as the background), or displayed in one of three colours.

The video RAM coding scheme used for this display mode uses each byte to encode four adjacent pixels. This means that each pixel is encoded in two bits. To illustrate this, here is the coding for the first four pixels on the screen, up in the top left-hand corner:




The next four pixels along the line are stored in location 7001H, and so on. The 2-bit colour coding used for each pixel is shown below:

(i) Backbround colour 0 (green):

00 = GREEN (background colour)
01 = YELLOW
10 = BLUE
11 = RED

(ii) Background colour 1 (buff):

00 = BUFF (background colour)
01 = CYAN
10 = ORANGE
11 = MAGENTA

Note that from BASIC, any pixel may be individually turned on or off using the SET(x,y) and RESET(x,y) command, and given various colours using the COLOR(m,n) command.

Video display worksheets for both mode (0) and mode (1) are given at the rear of this manual. These can be very handy for planning the display screens, menus etc when you are writing programs. Feel free to photocopy these worksheets, so you can use the photocopies in this way.



## 7. VZ-200 SCREEN CONTROL CODES

The following codes can be used for screen control from BASIC:

| | | | |
|---|---|---|---|
| Cursor left | PRINTCHR$(8) | Cursor right | PRINTCHR$(9) |
| Cursor up | PRINTCHR$(27) | Cursor down | PRINTCHR$(10) |
| Rubout | PRINTCHR$(127) | Insert | PRINTCHR$(21) |
| Home | PRINTCHR$(28) | Clear screen | PRINTCHR$(31) |

## 8. SYSTEM POINTERS, ETC

Here are some of the main system pointers and variable storage locations of interest to VZ-200 programmers:

| POINTER OR VARIABLE | HEX LOC | DECIMAL |
|---|---|---|
| Top of Memory (ptr) | 78B1/2 | 30897/8 |
| Start of BASIC program (ptr) | 78A4/5 | 30884/5 |
| End of BASIC program (ptr)<br>(also start of simple variables table) | 78F9/A | 30969/70 |
| Start of dim. variables table (ptr) | 78FB/C | 30971/2 |
| End of BASIC's stack (ptr)<br>(also start of string variable storage area) | 78A0/1 | 30880/1 |
| Execute address for USR program (ptr)<br>(note: high byte of address must go in 788F) | 788E/F | 30862/3 |
| Interrupt exit (called upon interrupt) | 787D/E/F | 30845/6/7 |
| Start of BASIC line input buffer<br>(buffer is 64 bytes long -- 2 screen lines) | 79E8 | 31208 |
| Copy of output latch | 783B | 30779 |
| Cursor position | 78A6 | 30886 |
| Output device code<br>(0 = video, 1 = printer, -1 = cassette) | 789C | 30876 |

The contents of the BASIC stack pointer stored in 78A0/1 are basically equal to the contents of the 'top of memory' pointer stored in 78B1/2, less a figure equal to the number of bytes reserved for string storage. The default value for string storage space is 50 bytes; this can be modified from within a BASIC program by using the CLEAR command -- i.e., CLEAR 1000 will increase the string space to store 1000 bytes.

The VZ-200 printer interface uses I/O port address 0E for the ASCII character code data and strobe output, and address 00H for the busy/ready-bar status input (bit 0).

## 9. RESERVING SPACE FOR A MACHINE CODE PROGRAM

There are a number of ways to reserve memory space for a machine code program, from within a BASIC program. But before details of these methods are given, we should clarify the way that BASIC normally organises RAM memory space.

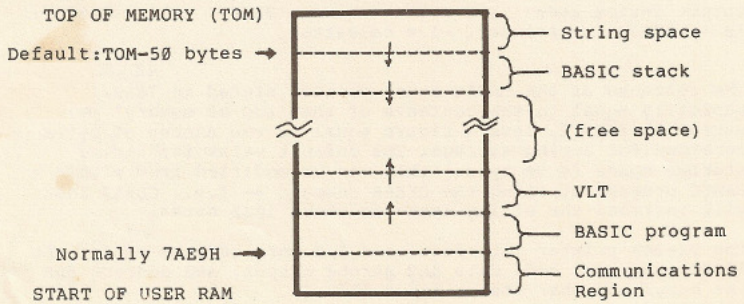
A range of addresses at the bottom of user RAM is reserved for system pointers and variables. This section is often termed the "communications region". It includes locations which store pointers to the boundaries of the various regions in upper RAM, like the 'Top of Memory' pointer, the 'start of BASIC program' pointer and so on. The latter pointer is stored at 78A4/5 (decimal 30884/5).

Normally the BASIC program itself is stored next, in locations starting at address 7AE9 hex. At the end of the BASIC program text, the system stores a table containing the program's variables. This is known as the 'variable list table' (VLT). This is divided into two sections: first the simple variable table, containing simple numeric variables and pointers to the simple string variables, and then the subscripted variable table containing dimensioned variables.

As the BASIC program text changes in length, the VLT is moved up or down in memory so that it always begins at the end of the program. The pointer to the start of the VLT is stored in location 78F9/A, and the pointer to the start of the subscripted variable table in location 78FB/C.

The remaining major regions extend <u>downward</u> from the top of user RAM. Normally at the very top of RAM is the string storage area, extending down from the top of RAM (pointer stored at 78B1/2) by either the default figure of 50 bytes, or a different amount established by a CLEAR N command. The BASIC interpreter's stack then extends downward in memory from the bottom of the string area (pointer stored in 78A0/1).

The space between the top of the VLT region and the bottom of the stack is not used, and is designated 'free space'. So that normally, the RAM organisation looks like this:






<u>Method 1:</u> This method of reserving space for a machine code program involves shifting the BASIC program area upward in memory from its normal start at 7AE9, creating a space immediately above the communications region. The machine code program can then be loaded into this space, probably by POKEing it from your main BASIC program.

Needless to say, the BASIC program area can only be shifted up <u>before</u> your main program is loaded into it (if it were done afterwards, the start of the program would be lost). But the shifting is quite easy to do, because all that is required is (a) to change the 'Start of BASIC program' and 'End of Program/Start of VLT' pointers, together with (b) creation of a new 'null program' at the start of the new program area.

This can be done quite easily by a small BASIC program which is fed into the computer ahead of your main program. Here is what it looks like if you want to reserve say 128 bytes:

```
10 POKE 31593,0:POKE 31594,0:POKE31595,0
20 POKE 30884,105:POKE 30885,123
30 POKE 30969,107:POKE 30970,123
```

Here line 10 pokes a 'null program' of 3 zero bytes into the start of the new program area (which starts at 7B69H, or 31593). Line 20 pokes the decimal equivalents of the low and high bytes of this new starting address of the program area into its pointer address, while line 30 pokes in the corresponding values for the new EOP/VLT pointer.

Note that this shifting program 'self destructs' -- once you run it, the BASIC interpreter loses all knowledge of its existence in memory. So if you then try to LIST or RUN, nothing will happen because as far as the interpreter is concerned, it now has nothing in its (new) program storage area.

Once the program has run, however, any BASIC program you load in will start at the new, higher address (here 128 bytes up), leaving the space immediately above the communications area free for a machine language routine or program.

Needless to say you can vary the above program to adjust the amount of space reserved. You'll need to change both the values poked into the pointer locations in lines 20 and 30, and the poke addresses in line 10.

Don't forget that if you use this method, the 'reserving' program will have to be loaded and run ahead of the main program, every time you want to use it. The reserving operation can't be done from within the main program itself.

This is one disadvantage of this method; another is that it is not easy to load in both your main basic program and the machine language program directly from tape.

Method 2: With this method of reserving space for a machine language program, you create the required space in between the end of the main BASIC program and the start of the VLT, by shifting the VLT upward in memory.

This is simpler to achieve than Method 1, because all that is required is to change the 'End of BASIC program/Start of VLT' pointer stored in 78F9/A hex (decimal 30969/70). In effect, we 'fool' the BASIC interpreter into thinking that the BASIC program is longer than it really is.

How do you work out the new value for this EOP/VLT pointer? Probably the best approach is to PEEK at the value of the pointer when your main program is loaded in normally, and then add to this figure the amount you need for the machine language routine -- plus a small amount (say 64 bytes) for safety margin.

Let's say you again want to reserve 128 bytes. First load in your main BASIC program, then key in this command:

```
PRINT PEEK(30969) + 256*PEEK(30970)
```

The answer you get is the current value of the EOP/VLT pointer, in decimal. In other words, it represents the actual end of your BASIC program. So add say 192 to this (128 plus a safety margin), to get the new EOP/VLT pointer value.

Say the value you get is 32800. Now find the decimal equivalents of the high and low pointer bytes for this figure, by keying in this line:

```
P=32800:PRINT INT(P/256), P-(256*INT(P/256))
```

The first number you get is the pointer high byte (in this case 128), while the second is the pointer low byte (here 32). Obviously if you get a different value from 32800, key this into the above line to get the corresponding values.

Now all you have to do is fit these values into a pair of POKE statements at the very start of your main BASIC program:

```
1 POKE 30969,32 :POKE 30970,128
```

This line must be right at the start of your program, so that the EOP/VLT pointer is moved before the program introduces or uses any variables. Otherwise the variables would be 'lost'.

This method allows you to load, save and run the BASIC program normally, without any prior preparation. Once you have loaded the machine language program into the reserved space between the BASIC program and its VLT, you can also save and re-load it along with the BASIC program, automatically. This is because the CSAVE and CLOAD routines use the EOP/VLT pointer to indicate the end of the BASIC program.

Note that the 64-byte 'safety margin' allows for the small increase in program length when you add line 1 above.



Method 3: This method of reserving space for a machine language program involves changing the 'Top of Memory' (TOM) pointer so that it points to an address lower than the actual top of memory. This forces the BASIC interpreter to move its string storage area and stack downward, leaving a space for your machine language program at the top.

Like Method 2, this is quite easy to do and it can be done from within your BASIC program.

First, you need to PEEK the current value of the TOM pointer. This is found quite easily:

```
PRINT PEEK(30897) + 256*PEEK(30898)
```

This will normally give you 36863 for a basic VZ-200, or 53247 if you have the 16K Memory Expansion Module plugged in.

Then you simply subtract from this figure the amount of space you want to reserve for the machine language program, to give a new TOM address. Then it's simply a matter of poking the low and high byte figures for this address into the TOM pointer, at the start of your program.

For example, say you want to reserve 256 bytes, and you have a basic VZ-200 so the normal TOM is 36863. So the new artificial TOM will be 36863-256, giving 36607. To work out the two new pointer bytes in decimal, type in:

```
T=36607:PRINT INT(T/256), T-(256*INT(T/256))
```

The first number you get is the pointer high byte (here 142), while the second is the low byte (here 255). If you have a different value for TOM (T), you'll get corresponding values.

Having found these values, all you need to do is add the following line to the start of your program:

```
1 POKE 30897,255:POKE 30898,142
```

The pointer must be changed before the program uses string variables and the stack, or the system could 'crash'.

Note that this method allows your BASIC program to be loaded, saved and run normally. However it does not allow the machine language program to be loaded directly into the reserved area at the same time. The machine code must be loaded either separately, or POKEd into the reserved area by the BASIC program itself -- after the pointer is changed.

## 10. FINDING THE TOP OF YOUR VZ-200's MEMORY

This is quite an easy one -- simply type in the line:

```
PRINT PEEK(30897) + 256*PEEK(30898)
```

## 11. CALLING A MACHINE CODE ROUTINE FROM BASIC

The standard way of calling a machine language program or routine from BASIC is to use the USR(X) command. But before this command can be used, the starting address of the machine language routine must be loaded into the USR program pointer, stored at address 788E/F hex (decimal 30862/3). This can be done using POKE statements.

To illustrate this, let's look at an example. Let's say you want to use the INKEY$ function in your BASIC program, to accept input character-by-character. But you'd like the VZ-200 to give its usual 'beep' each time to register input, and the INKEY$ function doesn't provide this.

As it happens, the BEEP subroutine in VZ-200's BASIC ROM can easily be called to do this, using the USR(X) command. The calling address for the routine is 3450 hex, so the decimal figures for the USR pointer bytes are 80 (low byte, equal to 50 hex) and 52 (high bytes, equal to 34 hex).

So if you want to produce a 'beep' at various places in your BASIC program, all you need to do is put this line near the start of the program (before the first beep is needed):

```
20 POKE 30862,80:POKE 30863,52
```

This sets up the USR pointer. Then wherever you want a 'beep' in your program, simply use the command:

```
X=USR(X)
```

Note that before control is passed to the user routine at the designated address, the value of the argument variable X is stored in locations 31009/31010 (7921/2 hex). So this can be used to 'pass' a parameter value to the user routine. If the routine doesn't need any parameters (like the 'beep' routine above), simply use a 'dummy' variable name like X, as shown.

The same general technique is used for calling other machine code routines, whether they are located in ROM or RAM. It's simply a matter of poking the start address of the routine into 30862/3, and then using the USR command.

You aren't limited to calling a single machine code routine, by the way. You can call a number of routines in turn, simply by poking each routine's start address into 30862/3 before you use the USR command to call it. Just remember to POKE the right routine address into the pointer each time!



## 12. USEFUL ROM SUBROUTINES FOR ASSEMBLY PROGRAMMING

### A. KEYBOARD SCANNING ROUTINE

The keyboard scanning routine resides at 2EF4 hex. This routine scans the keyboard once and returns. If a key is pressed, the A register will contain the code for that key; otherwise this register will contain zero. Registers AF, BC, DE and HL are all modified by the routine, so if the contents of these registers must be preserved they should be pushed onto the stack before the routine is called. The following example shows how the routine would be used to wait for the RETURN key to be pressed:

```
SCAN   CALL 2EF4H       ;scan kybd once
       OR   A 10        ;any key pressed?
       JR   Z,SCAN      ;back if not
       CP   0DH 13      ;was it RETN key?
       JR   NZ,SCAN     ;back if not
       ...              ;otherwise continue
```

### B. CHARACTER OUTPUT SUBROUTINE

A routine which outputs a single character to the video display is located at 033A hex. The code for the character to be displayed must be in the A register, while the character will be displayed on the screen at the position corresponding to the current value of the cursor pointer. All registers are preserved. Here is how the routine would be called to display the word 'HI', followed by a carriage return:

```
       LD  A,'H'        ;load A reg with code
       CALL  033AH      ;& display
       LD  A,'I'        ;same for I
       CALL  033AH
       LD  A, 0DH       ;now load A with CR code
       CALL  033AH      ;& update screen
```

### C. MESSAGE OUTPUT SUBROUTINE

A very useful subroutine located at 28A7 hex can display a string of character codes as a message on the screen. The string of character codes must end with a zero byte. The HL register pair must be set to the start of the string before the subroutine is called. All registers are used by the subroutine. Here is how it is used:

```
       LD  HL,MSG       ;load HL with start of strg
       CALL 28A7H       ;& call print subroutine
       ....

MSG    DEFM 'READY'     ;main message string
       DEFB 0DH         ;carriage return
       DEFB 0           ;null byte to terminate
```

D. COMPARE SYMBOL (EXAMINE STRING) -- RST 08H

A routine which is called using the RST 08H instruction can be used to compare a character in a string pointed to by the HL register, with the value in the location following the RST 08 instruction itself. If there is a match, control is returned to the instruction 2 bytes after the RST 08, with the HL register incremented by one and the next character of the string in the A register. This allows repeated calling to check for an expected sequence of characters. Note that if a match is NOT found, the RST 08 routine does not return from where it was called, but jumps instead to the BASIC interpreter's input phase after printing the SYNTAX ERROR message. Here is how the routine is used to check that the string pointed to by the HL register is 'A=B=C':

```
RST 08H       ;test for 'A'
DEFB 41H      ;hex value of A for comparison
RST 08H       ;must have found, so try for '='
DEFB 3DH      ;hex value of '='
RST 08H       ;OK so far, try for 'B'
DEFB 42H
RST 08H       ;now look for second '='
DEFB 3DH
RST 08H       ;finally check for 'C'
DEFB 43H
....          ;must have been OK, so proceed
```

E. LOAD & CHECK NEXT CHARACTER IN STRING -- RST 10H

The RST 10H instruction may be used to call a routine which loads the A register with the next character of a string pointed to by the HL register, and clears the CARRY flag if the character is alphabetic, or sets the flag if it is alphanumeric. Blanks and control codes 09H and 0BH are skipped automatically. The HL register is incremented before each character is loaded, therefore on the first call the HL register should be set to point to the address BEFORE the location of the first string character to be tested. The string must be terminated by a null byte.

Here is an example of this routine in use. Note that if it is used immediately after the RST 08H instruction as shown, the HL register will automatically be incremented to point to the next character in the string:

```
RST 08H           ;test for '='
DEFB 3DH
RST 10H           ;fetch & check next char
JR  NC, VAR       ;will go to VAR if alpha
...               ;continues if numeral
```



F. COMPARE DE & HL REGISTER PAIRS -- RST 18H

The instruction RST 18H may be used to call a routine which compares the contents of the DE and HL register pairs. The routine uses the A register only, but will only work for unsigned or positive numbers. Upon returning, the result of the comparison will be in the status register:

```
HL <  DE : carry set
HL >  DE : no carry
HL <> DE : NZ
HL =  DE : Z
```

Here is an example of its use. Assume the DE pair contains a number and we want to check that it falls within a certain range -- say between 100 and 500 (decimal):

```
LD   HL,500    ;load HL with upper limit
RST  18H       ;& call comparison routine
JR   C,ERR     ;carry means num > 500
LD   HL,100    ;now set for lower limit
RST  18H       ;& try again
JR   NC,ERR    ;no carry means num < 100
....           ;if still here, must be OK
```

G. SOUND DRIVER

Located at 345C hex is a routine which can be used to produce sounds via the VZ-200's internal piezo speaker. Before calling the routine, the HL register pair must be loaded with a number representing the pitch (frequency) of the tone to be produced, while the BC register pair must be loaded with the number of cycles of the tone required (i.e., the duration in cycles). All registers are used. The frequency coding used is inversely proportional to frequency, i.e., the smaller the number loaded into the HL register pair, the higher the frequency. As a guide, the low C produced by VZ-200's SOUND command in BASIC can be produced using the decimal number 526, the middle C using 259 and the high C using 127. Here is how you would call the routine to get say 75 cycles of the middle C:

```
LD    HL, 259     ;set frequency code
LD    BC, 75      ;set number of cycles
CALL  345CH       ;& call sound routine
....
```

H. 'BEEP' ROUTINE

The routine which is used by BASIC to produce the short 'beep' when a key is pressed is located at address 3450 hex. It disturbs all registers except the HL pair. All you have to do to produce a beep is call it:

```
CALL  3450        ;make a 'beep'
```

J. CLEAR SCREEN

A routine located at Ø1C9 hex may be used to clear the video screen, home the cursor and select display mode (Ø). It disturbs all registers. Again it is used simply by calling it:

```
CALL  Ø1C9        ;clear screen, home cursor etc
```

K. PRINTER DRIVER

The printer driver routine is located at Ø58D hex. To send a character to the printer, load the character's ASCII code into the C register and call the driver. After printing the character code will be returned in both the A and C registers. All other registers are disturbed. For example to print the letter 'a' (ASCII code 97 decimal), you would use:

```
LD   C,97         ;set up code in C reg
CALL Ø58DH        ;& call printer driver
...
```

A line feed character (ØAH) is automatically inserted after a carriage return (ØDH). If the driver is called with a null byte in the C register, it will simply check printer status and return with bit Ø of the A register either set or cleared. The routine does check for a BREAK key depression, and if one is detected, it will return with the carry flag set.

L. CHECK PRINTER STATUS

A routine to check printer status is located at Ø5C4 hex. When called it loads the printer status (I/O port ØØH) into the A register and returns. Bit Ø will be set (1) if the printer is busy, or cleared (Ø) if it is ready. No other registers are disturbed. An example:

```
TEST  CALL  Ø5C4H      ;check is printer is ready
      BIT   Ø,A        ;test bit Ø
      JR    NZ,TEST    ;loop if busy
      ...              ;must be ready
```

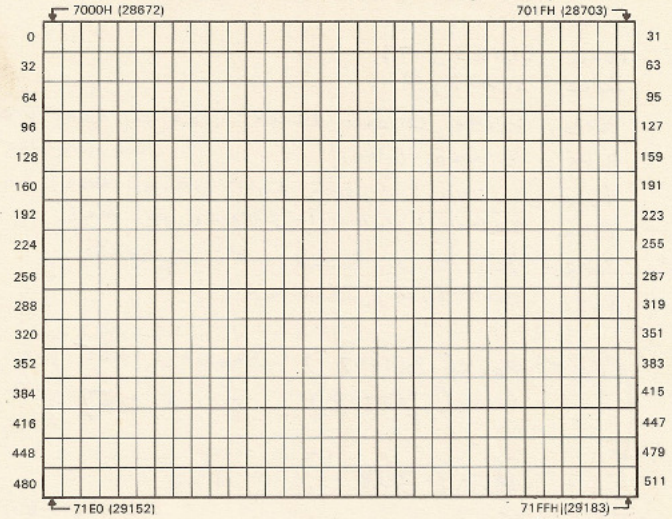
M. SEND CR-LF TO PRINTER

A routine located at 3AE2 hex may be used to send a carriage return and line feed combination to the printer. No registers need be set up before calling, but all registers are disturbed. If the BREAK key is pressed while printing occurs (or while the printer driver is waiting for the printer to signal 'ready'), the routine will return with the carry flag set:

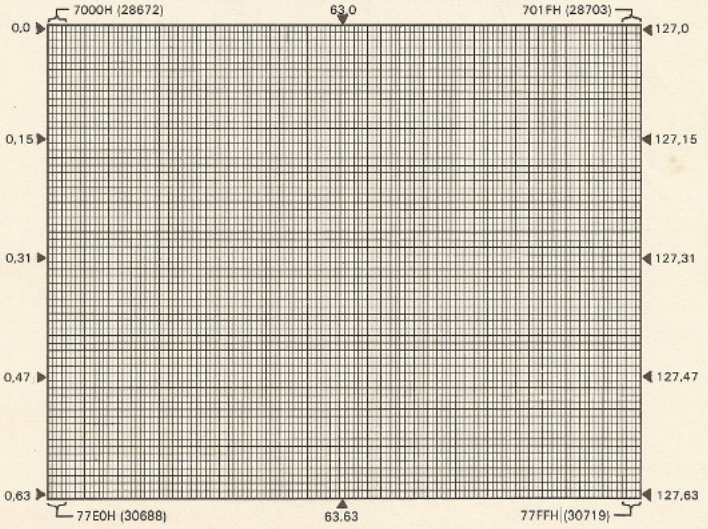
```
CALL  3AE2          ;go send CR-LF to printer
JP    C,BRK         ;check if BREAK key pressed
...                 ;apparently not
```



VZ-200 VIDEO DISPLAY WORKSHEET (MODE 0)




VZ-200 VIDEO DISPLAY WORKSHEET (MODE 1)



NOTES



FROM REAR

TOP RIGHT

P1

| | |
|---|---|
| 1 | NC |
| 2 | NC |
| 3 | NC |
| 16 | NC |
| 17 | NC |
| 18 | NC |
| 20 | NC |
| 4 | +5V |
| 19 | |
| 15 | |
| 30 | |
| 28 | A0 |
| 26 | A1 |
| 11 | A2 |
| 13 | A3 |
| 27 | A4 |
| 10 | A5 |
| 12 | A6 |
| 29 | A7 |
| 9 | D0 |
| 24 | D1 |
| 8 | D2 |
| 6 | D3 |
| 21 | D4 |
| 22 | D5 |
| 7 | D6 |
| 23 | D7 |
| 5 | $\overline{IORQ}$ |
| 25 | $\overline{RD}$ |
| 14 | $\overline{WR}$ |

I/O EXPANSION CONNECTOR

TOP LEFT

P2

| | |
|---|---|
| 22 | NC |
| 21 | +5V |
| 1 | |
| 12 | A1 |
| 11 | A2 |
| 10 | A3 |
| 9 | A4 |
| 8 | A5 |
| 7 | A6 |
| 6 | A7 |
| 5 | A8 |
| 4 | A9 |
| 3 | A10 |
| 13 | D2 |
| 14 | D7 |
| 2 | $\overline{RESET}$ |
| 15 | $\overline{RFSH}$ |
| 16 | $\overline{M1}$ |
| 17 | $\overline{WAIT}$ |
| 18 | $\overline{NMI}$ |
| 19 | $\overline{RD}$ |
| 20 | $\overline{IORQ}$ |

MEMORY EXPAN



NEXT ASSY

USED ON

FIRST APPLICATION

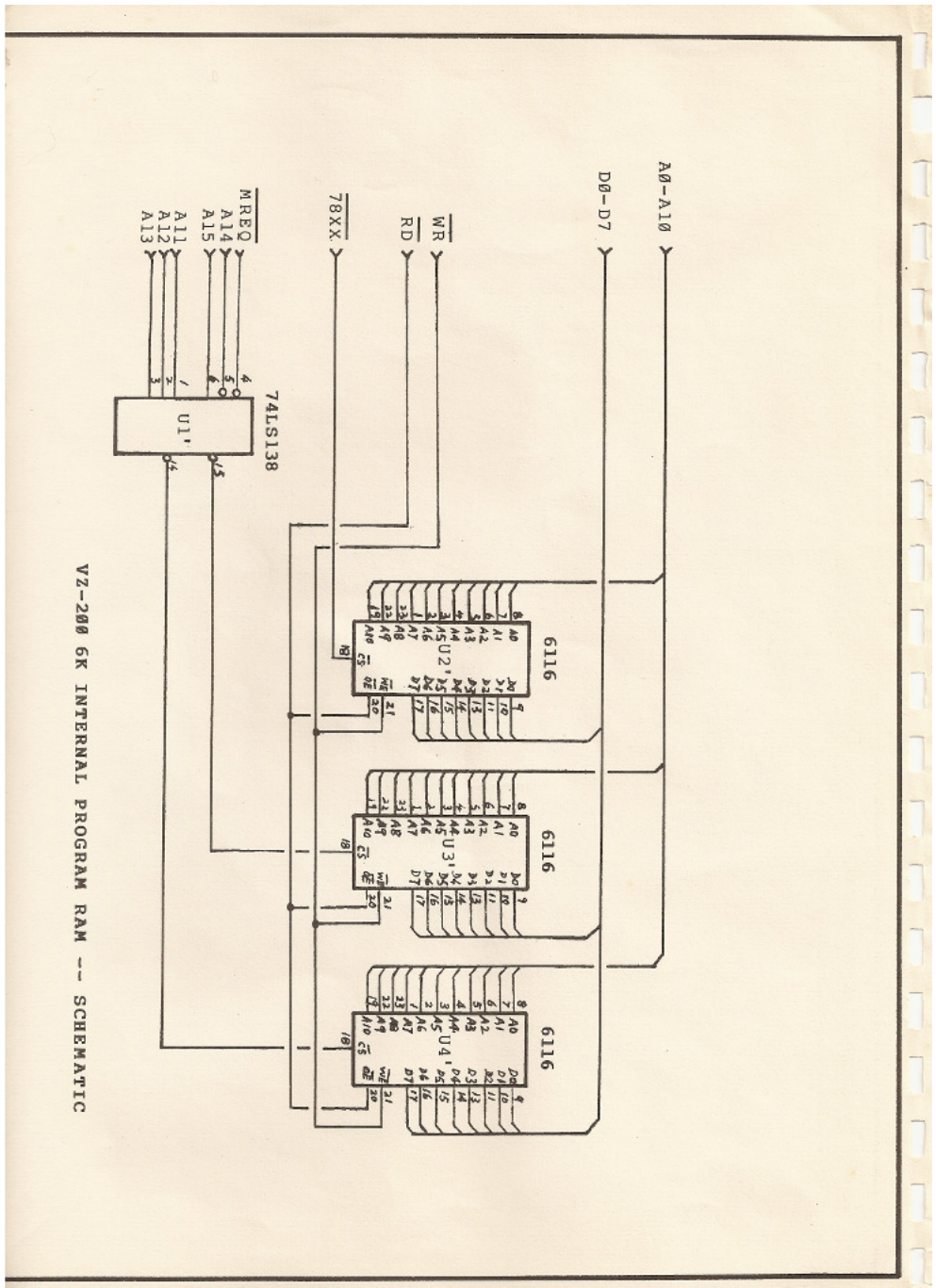
| REVISION | | | | | |
|---|---|---|---|---|---|
| REV | ZONE | DESCRIPTION | DRAWN | APPD | DATE |

P2

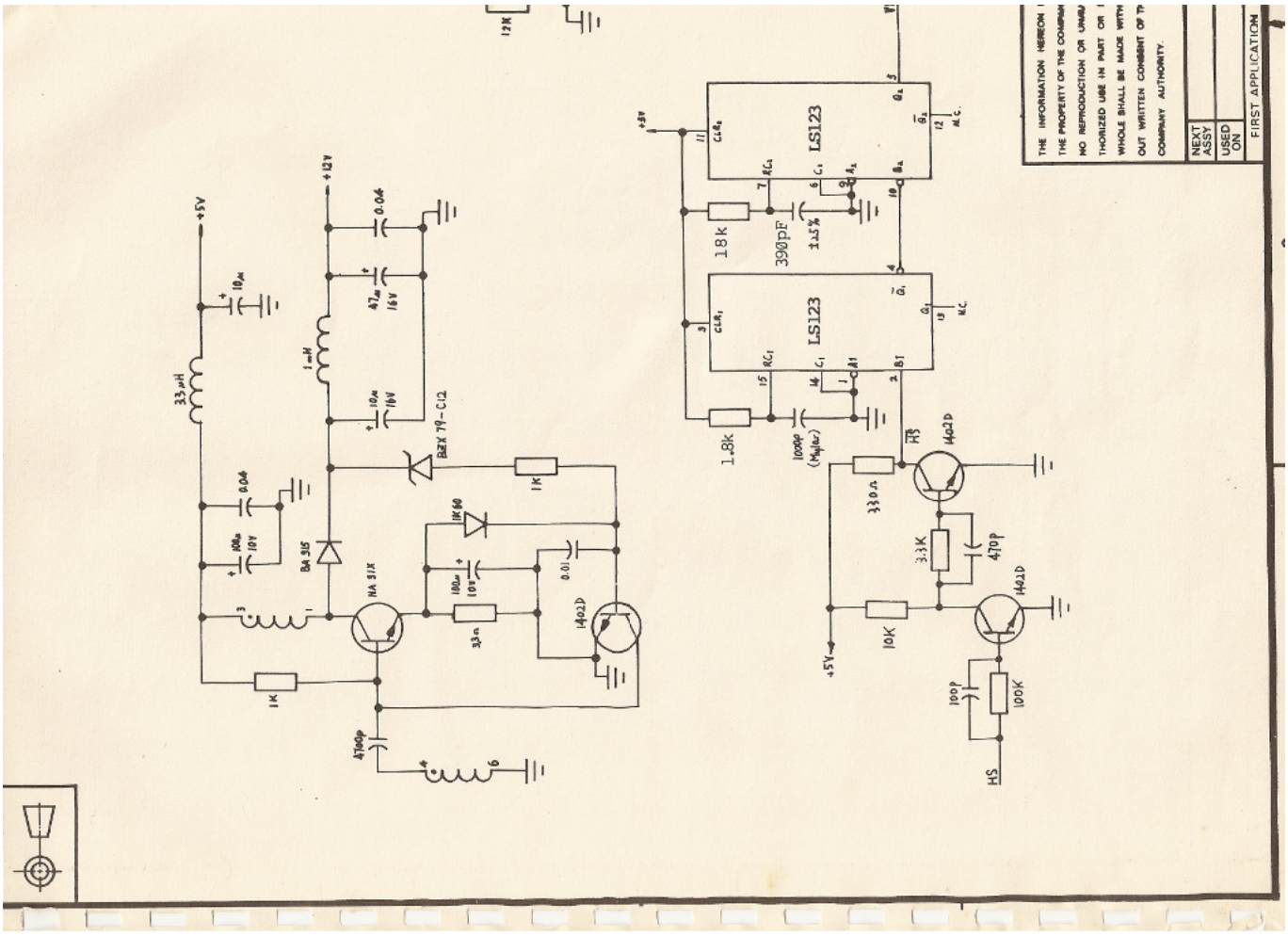
| | |
|---|---|
| 34 | NC |
| 42 | NC |
| 44 | NC |
| 43 | +9V |
| 23 | ⏚ |
| 24 | A11 |
| 25 | A12 |
| 26 | A13 |
| 27 | A14 |
| 28 | A15 |
| 35 | A0 |
| 36 | D0 |
| 37 | D1 |
| 31 | D3 |
| 30 | D4 |
| 32 | D5 |
| 33 | D6 |
| 29 | CLK |
| 38 | $\overline{INT}$ |
| 39 | $\overline{HALT}$ |
| 40 | $\overline{MERQ}$ |
| 41 | $\overline{WR}$ |

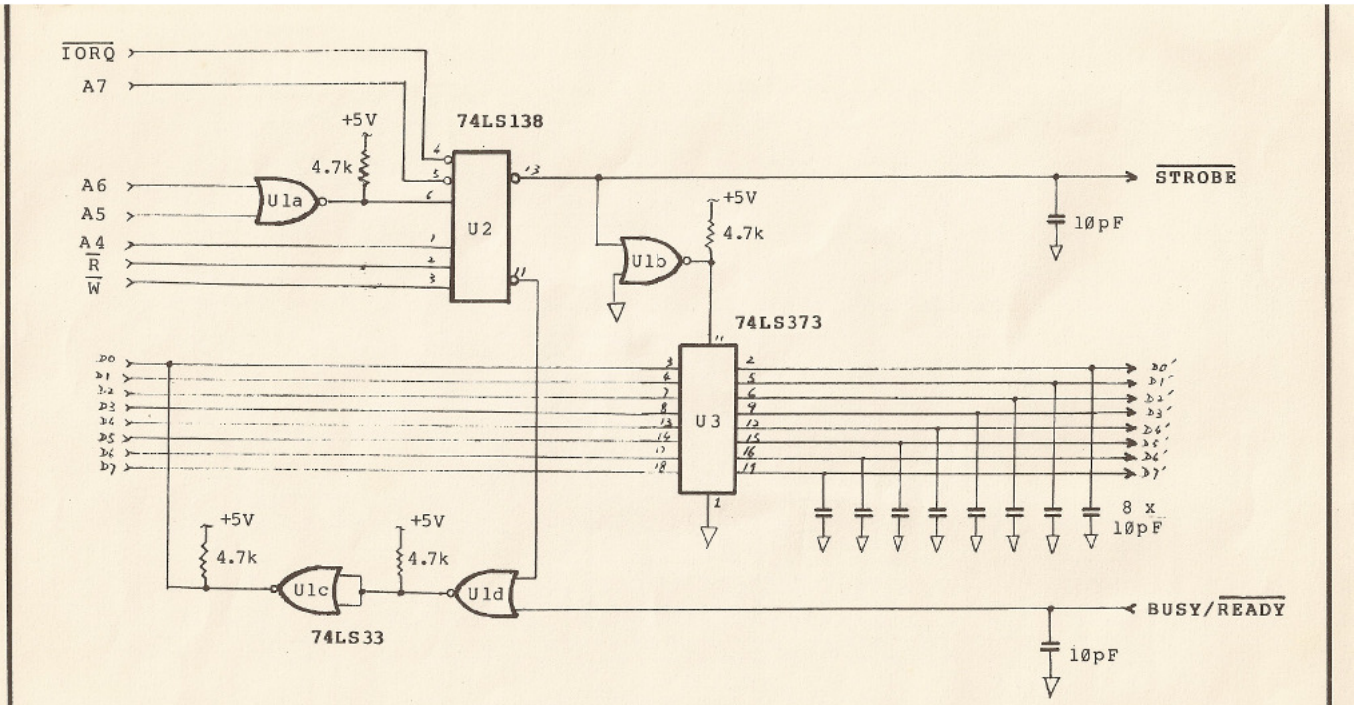
ION CONNECTOR

| UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE .XXX .XX ANGULAR ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING | SIGNATURE | DATE | DICK SMITH ELECTRONICS |
|---|---|---|---|
| | DWN. BY | | TITLE |
| | CHECK. BY | | VZ-200 PERSONAL COLOUR COMPUTER |
| | ENGR | | |
| | ENGRG AUTH | | |
| TERIAL | SIZE A3 | CODE IDENT | DWG NO 65-0237-00 |
| ISH | SCALE | | SHEET 4 OF 4 |




VZ-200 6K INTERNAL PROGRAM RAM -- SCHEMATIC

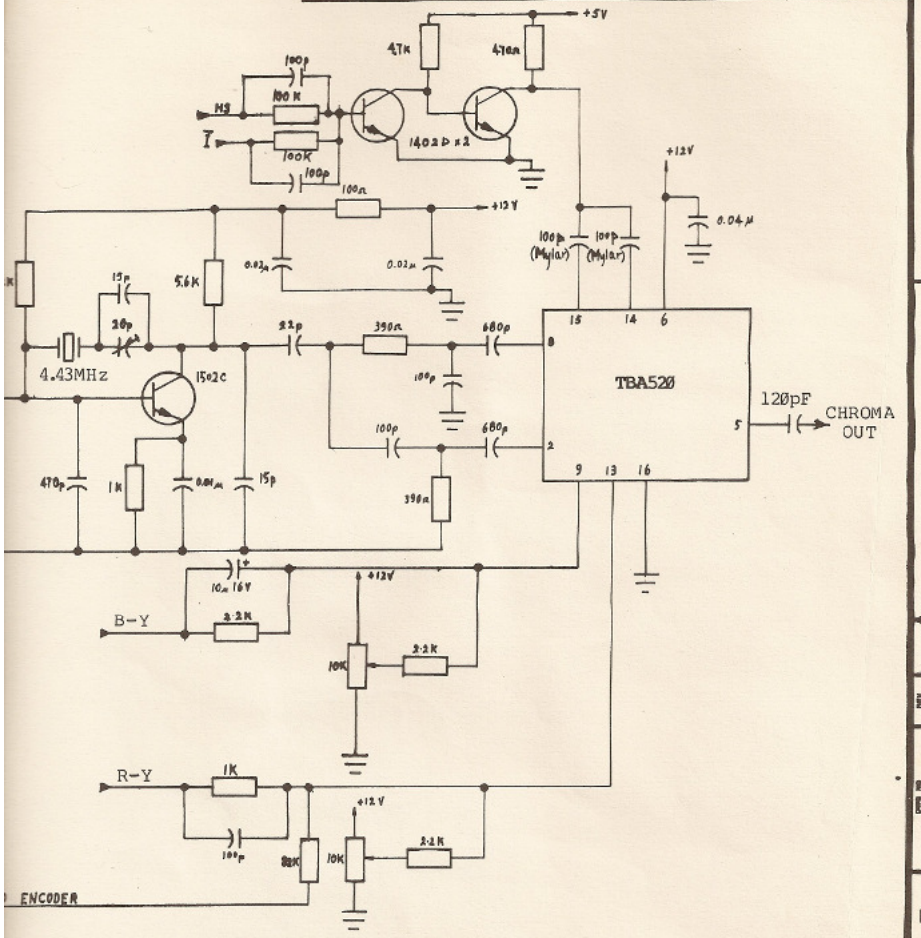


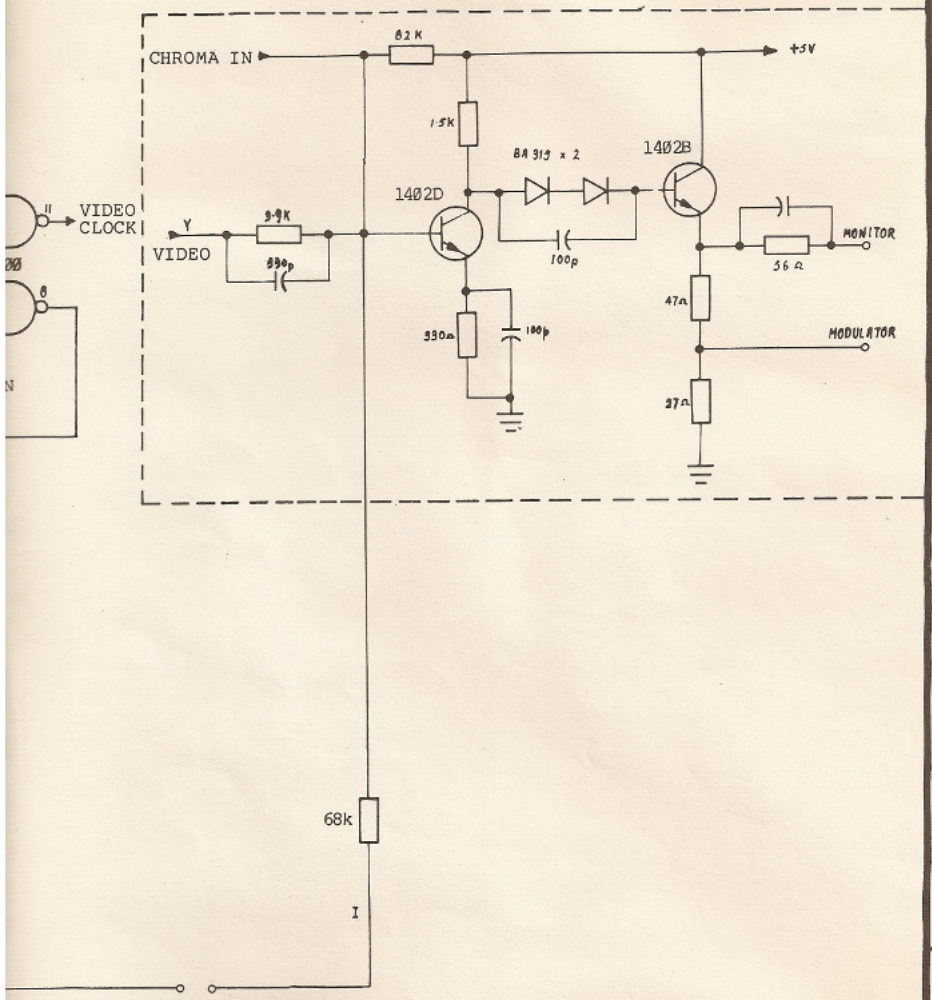




VZ-200 PRINTER INTERFACE -- SCHEMATIC

DICK SMITH ELECTRONICS PTY LTD

| REVISION | | | | | |
|---|---|---|---|---|---|
| REV | ZONE | DESCRIPTION | DRAWN | APPD | DATE |



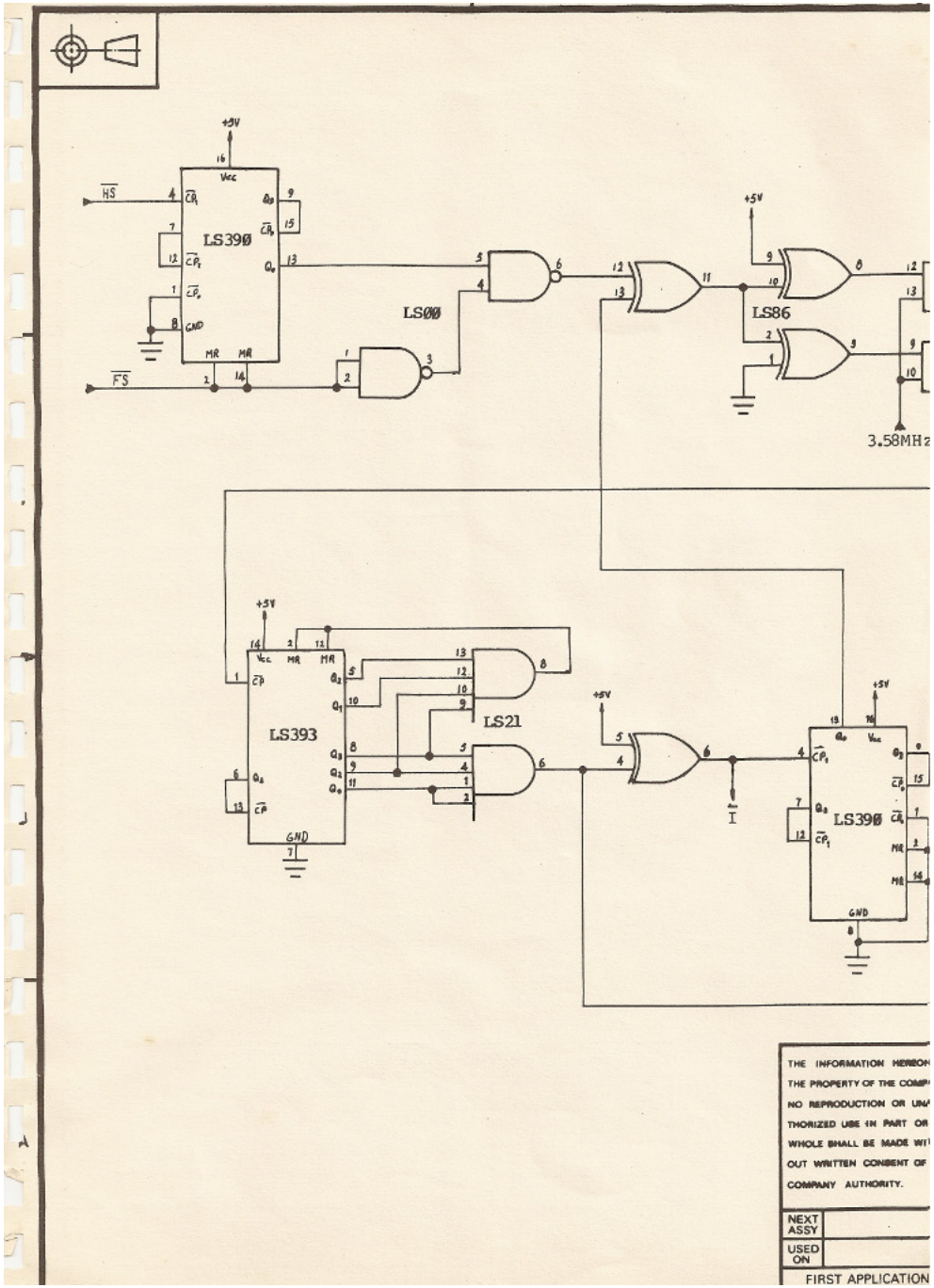



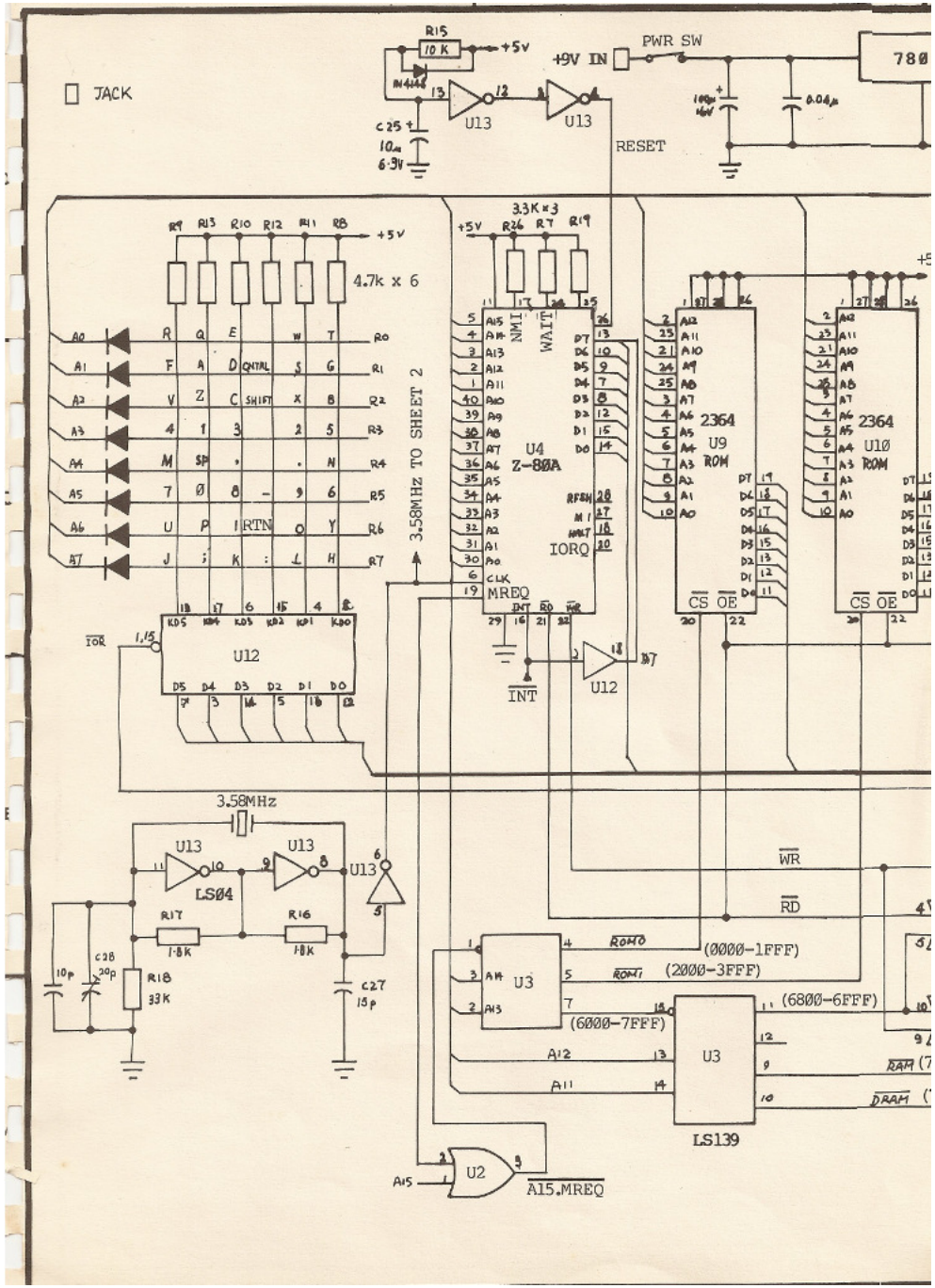

| UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE xxx xx ANGULAR ± ± ± ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING | SIGNATURE | DATE | DICK SMITH ELECTRONICS |
|---|---|---|---|
| | DWN. BY | | TITLE: VZ-200 PERSONAL COLOUR COMPUTER |
| | CHECK BY | | |
| | ENGR | | |
| | ENGRG AUTH | | |

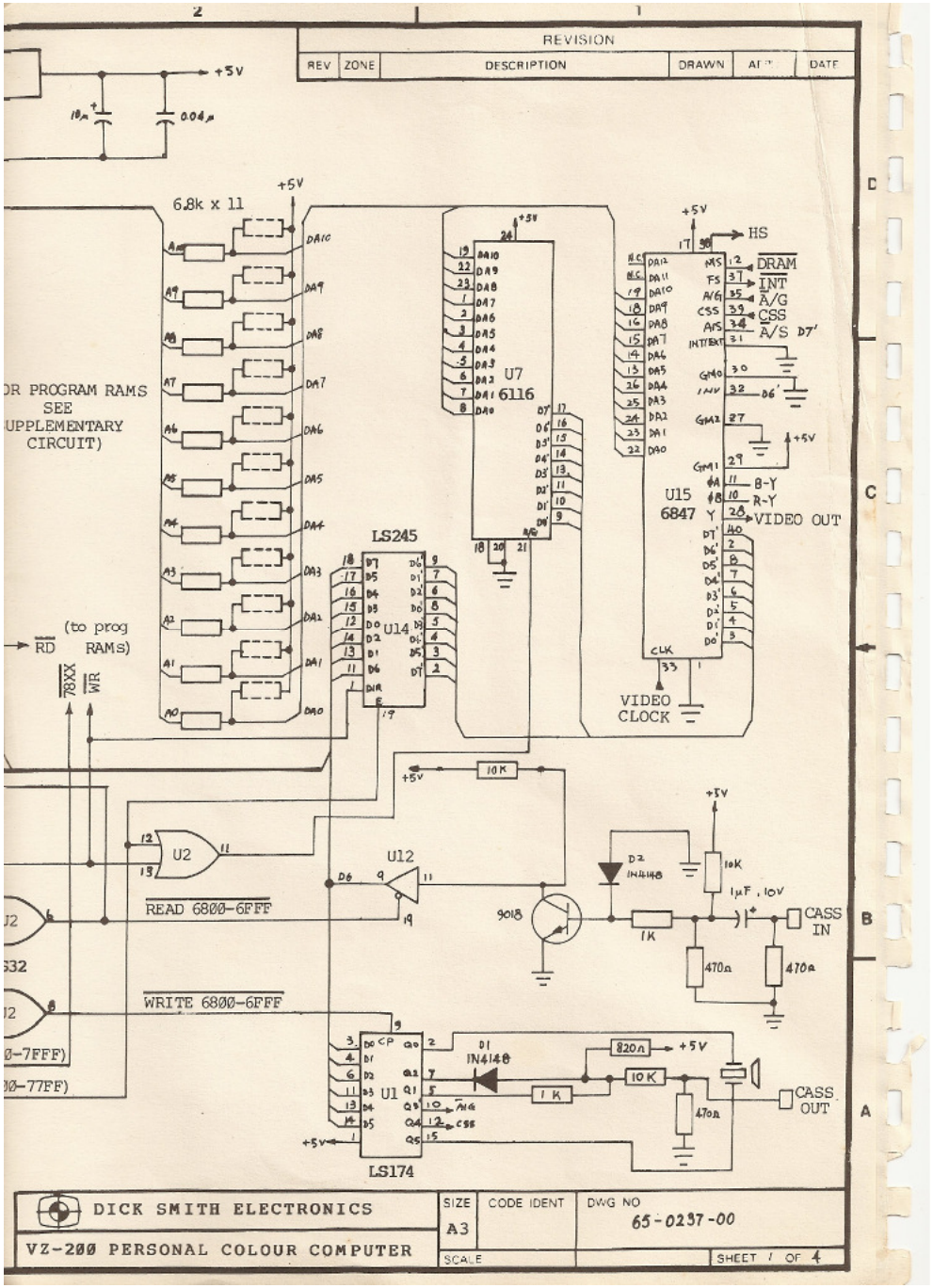
| MATERIAL | SIZE | CODE IDENT | DWG NO |
|---|---|---|---|
| FINISH | A3 | | 65-0237-00 |
| | SCALE | | SHEET 3 OF 4 |

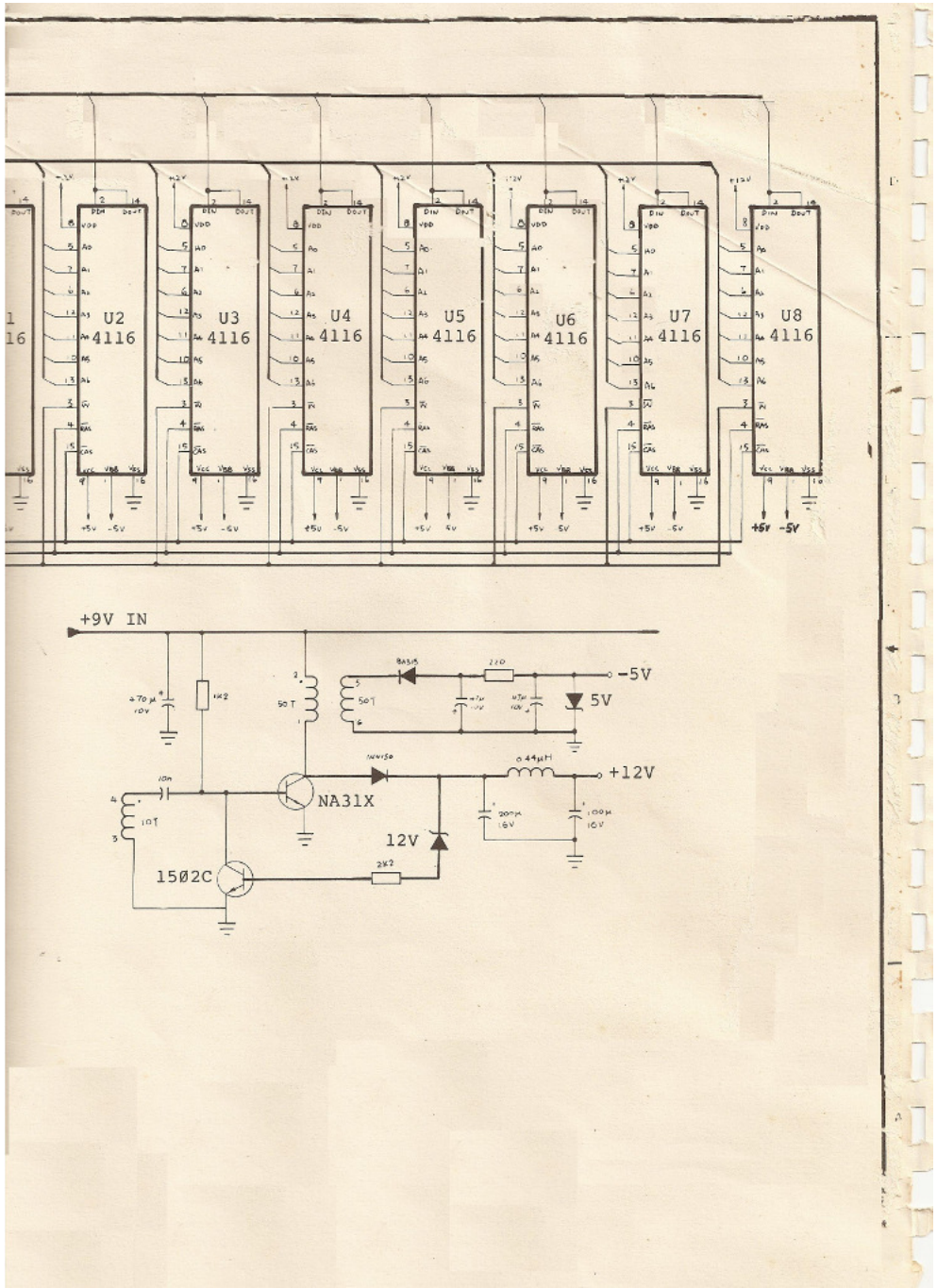

| UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN MM INCHES AND TOLERANCES ARE xxx xx ANGULAR ± ± ± ALL MACHINED SURFACE √ REMOVE BURRS BREAK ALL SHARP CORNERS DO NOT SCALE DRAWING | SIGNATURE | DATE | DICK SMITH ELECTRONICS |
|---|---|---|---|
| | DWN. BY | | TITLE: VZ-200 PERSONAL COLOUR COMPUTER |
| | CHECK BY | | |
| | ENGR | | |
| | ENGRG AUTH | | |

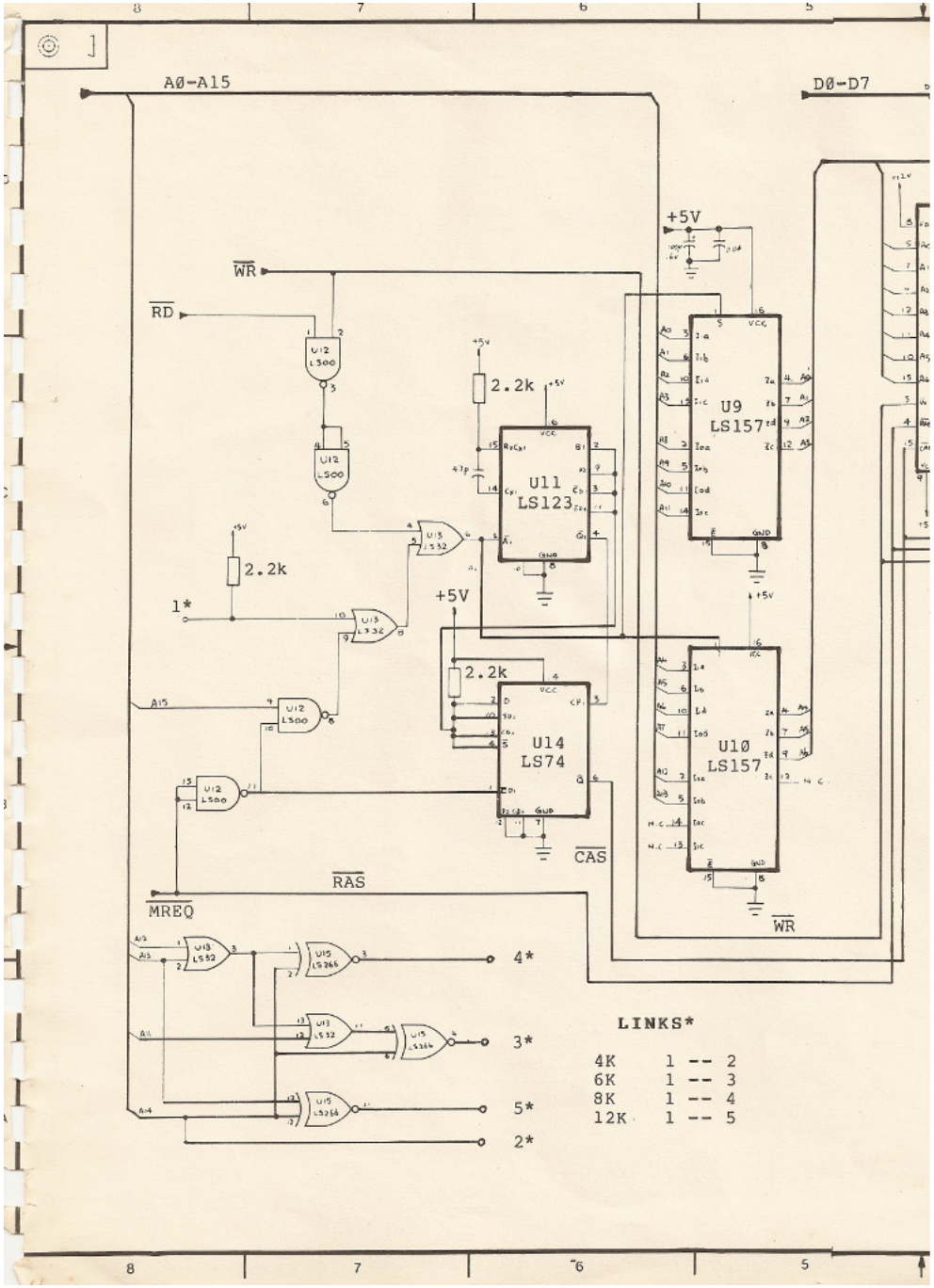

| MATERIAL | SIZE | CODE IDENT | DWG NO |
|---|---|---|---|
| FINISH | A3 | | 65-0237-00 |
| | SCALE | | SHEET 2 OF 4 |


LS390
LS00
LS86
LS21
LS393
LS390
3.58MHz
HS
FS
I
THE INFORMATION HEREON
THE PROPERTY OF THE COMP
NO REPRODUCTION OR UNA
THORIZED USE IN PART OR
WHOLE SHALL BE MADE WI
OUT WRITTEN CONSENT OF
COMPANY AUTHORITY.
NEXT ASSY
USED ON
FIRST APPLICATION


JACK
+9V IN
PWR SW
RESET
U13
4.7k x 6
3.3K x 3
3.58MHz TO SHEET 2
U4
Z-80A
NMI
WAIT
INT
MREQ
IORQ
U12
2364
U9
ROM
U10
CS OE
3.58MHz
LS04
R17
1.8K
R16
18K
R18
33K
C28
C27
15p
WR
RD
(0000-1FFF)
(2000-3FFF)
(6000-7FFF)
(6800-6FFF)
U3
LS139
A12
A11
U2
A15.MREQ


REVISION
REV
ZONE
DESCRIPTION
DRAWN
DATE
+5V
10µ
0.04µ
6.8k x 11
OR PROGRAM RAMS
SEE
SUPPLEMENTARY
CIRCUIT)
U7
6116
LS245
U14
U15
6847
HS
DRAM
INT
A/G
CSS
A/S
B-Y
R-Y
VIDEO OUT
VIDEO
CLOCK
(to prog
RAMs)
RD
WR
78XX
10K
U2
U12
READ 6800-6FFF
WRITE 6800-6FFF
9018
D2
IN4148
1K
1µF, 10V
CASS
IN
470Ω
U1
LS174
D1
IN4148
820Ω
CASS
OUT
DICK SMITH ELECTRONICS
VZ-200 PERSONAL COLOUR COMPUTER
SIZE
A3
CODE IDENT
DWG NO
65-0237-00
SCALE
SHEET 1 OF 4


U2
4116
U3
4116
U4
4116
U5
4116
U6
4116
U7
4116
U8
4116
+9V IN
-5V
5V
+12V
NA31X
12V
15Ø2C


A0-A15
D0-D7
+5V
WR
RD
U12 LS00
2.2k
47p
U11 LS123
U13 LS32
1*
U9 LS157
U10 LS157
U14 LS74
A15
CAS
RAS
MREQ
WR
U15 LS266
4*
3*
5*
2*
LINKS*
4K 1 -- 2
6K 1 -- 3
8K 1 -- 4
12K 1 -- 5