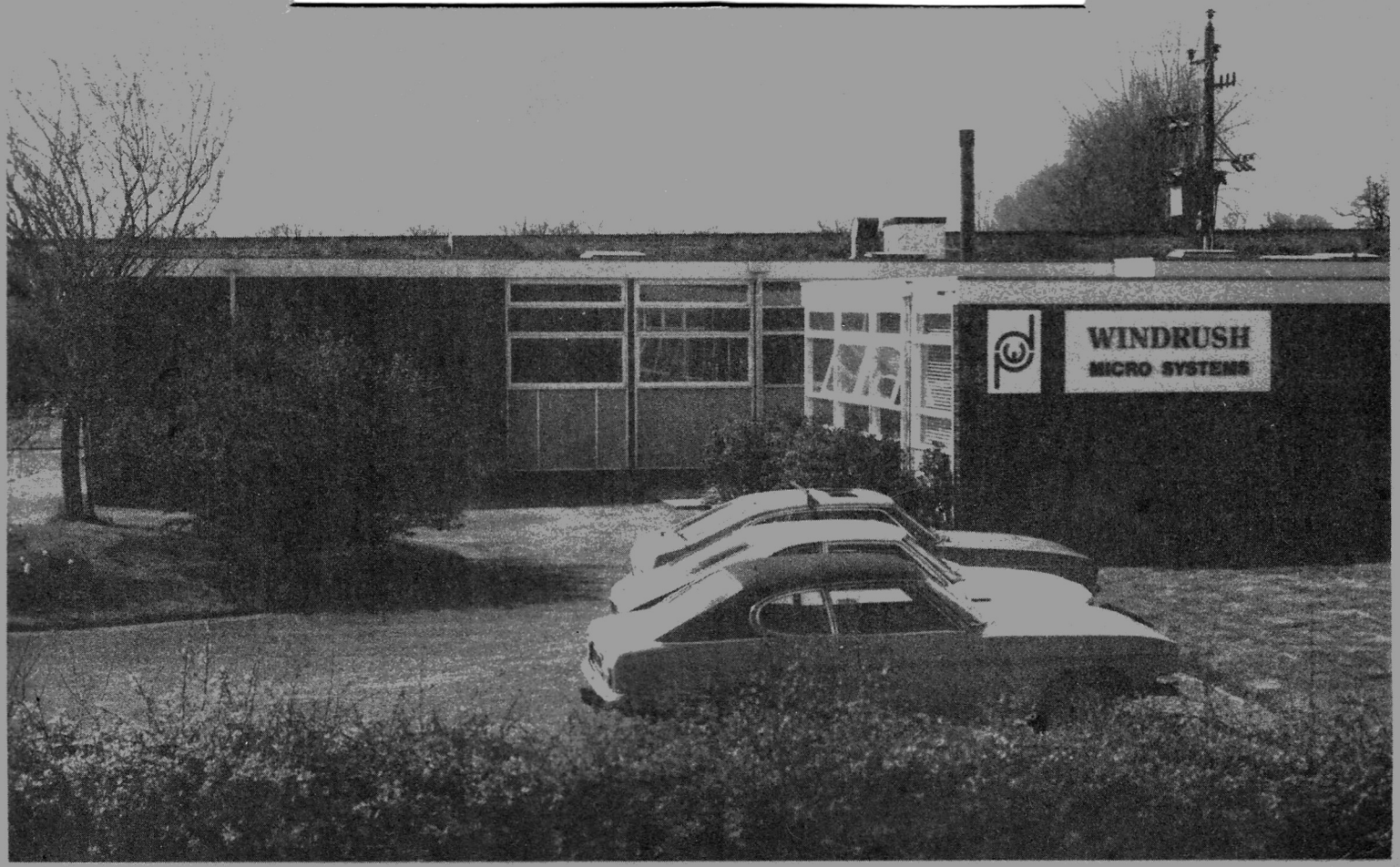
PADGETT NELSON INDUSTRIES, INC.
3684 Hedgewood Drive
Winter Park, Florida 32792
305-678-6776

# WINDRUSH
Micro Systems Ltd.




WORSTEAD LABORATORIES, (Reg. Office), NORTH WALSHAM, NORFOLK NR28 9SA
TELEPHONE (0692) 404086; TELEX 975548 WMICRO G; CABLES 'WINDRUSH' NORTH WALSHAM

# UPROM II

aprii

# UPROM II

# UNIVERSAL EPROM PROGRAMMER



First release: 15 November 1983

## T R A D E M A R K  N O T I C E S

'SCREDITOR III' IS A TRADEMARK OF ALFORD AND ASSOCIATES

'FLEX' IS A TRADEMARK OF TECHNICAL SYSTEMS CONSULTANTS

'OS-9' IS A TRADEMARK OF MICROWARE SYSTEMS CORPORATION

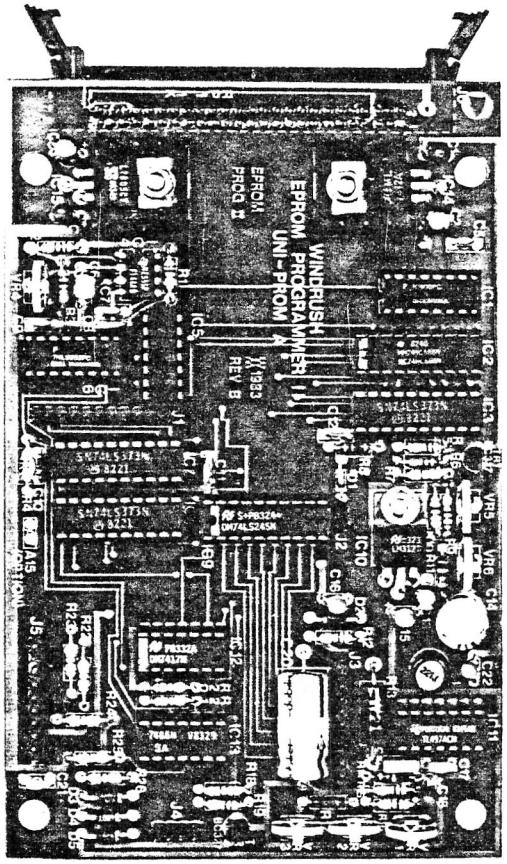
'MDOS' IS A TRADEMARK OF MOTOROLA INCORPORATED

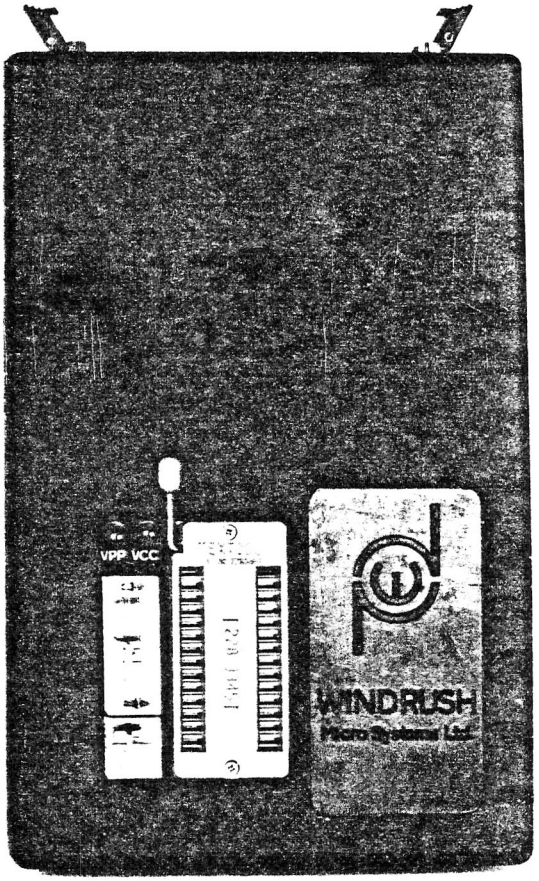
# I N D E X

# I N D E X



## APPENDIX




## 1.0 INTRODUCTION

The 'WINDRUSH UNIVERSAL EPROM PROGRAMMER II', (UPROM-II) for MC6809 development systems has evolved from its predecessor the 'ALL-IN-TWO' which has been in use on SSB DOS (tm), FLEX 2 (tm), FLEX 9 (tm), OS-9 (tm), MDOS (tm) and XDOS (tm) for the past three years. Well over one thousand of the 'ALL-IN-TWO' programmers are in service worldwide.

The software and hardware design of the UPROM-II programmer are substantially the same with four major differences. First the hardware has been simplified by eliminating the provisions for the now obscure tri-volt EPROMS. Second the hardware incorporates a programable digital monostable which ensures accurate programming pulse widths over a very wide range. Third the hardware and the software incorporate the features required to implement the INTEL 'Inteligent Programming Algorithm' (tm). This feature SUBSTANTIALLY reduces the programming times for compatible 2764, 27128, and 27256 EPROMS. Fourth the programmer 'pod', which extends out to the work area, is now housed in an injection moulded case.

The programmer software is supplied on a floppy disk and is available for a variety of operating systems that run under the MC6809. The menu driven software is very 'user friendly' and provides all of the features expected of a professional quality EPROM programmer. FILL, MOVE, EXAMINE/CHANGE, CRC CHECKSUM (device signature), DUMP, FIND, COPY, VERIFY, PROGRAM, DEVICE SELECTION and access to the operating system, primarily for LOAD and SAVE operations.

The host computer only need supply two unregulated D.C. supplies and an MC6821 PIA interface. The programmer pod is completely free standing in that it derives all of its power and intelligence from the host computer.

## TRADEMARK NOTICES

SSB DOS is a trademark of Smoke Signal Broadcasting, FLEX, FLEX 2 and FLEX 9 are trademarks of Technical Systems Consultants, OS-9 is a trademark of Microware Systems Corporation, and MDOS, XDOS, EXORCISER, EXORSET and MICROMODULE are trademarks of Motorola Incorporated. INTELigent is a trademark of Intel Corporation.

## ACKNOWLEDGEMENTS

The digital circuitry for the 'programmable monostable' used in this product was developed from an idea conceived by David Pick.

The FLEX 'LOAD' and 'MAP' utilities included with this package are part of the Windrush FLEX UTILITIES package developed by Neil Jarman.

The core of the hex byte finder routine used in this product was developed by Graham Trott.

## ABOUT THIS MANUAL

The terminology and general descriptions as to use of the EPROM programmer are based on a system running 'FLEX'. Since the majority of sales of this product will be for systems running 'FLEX' we felt that this organization was the most appropriate.

If you have purchased this product for use with SSB DOS, OS-9, MDOS or XDOS you will find addendums for these operating systems at the end of this manual. These sections outline the differences between the standard 'FLEX' version as described in the body of this document and these particular operating systems.

Operation of the programmer and its associated software is virtually identical in all operating systems.

## 2.0 APPLICATIONS

The range of BYTE WIDE EPROMS covered by the programmer and it's associated software are as follows:

| | | |
|---|---|---|
| 2508 (single voltage) (1K) | 2732 (4K) | 2764A (8K) |
| 2758 (single voltage) (1K) | 2732A (4K) | 27128 (16K) |
| 2516 (single voltage) (2K) | 68764/6 (8K) | 27128A (16K) |
| 2716 (single voltage) (2K) | 2564 (8K) | 27256 (32K) |
| 2532 (4K) | 2764 (8K) | |

WARNING: The 2732 is programmed at 25 volts whilst the 2732A is programmed at 21 volts. If you accidentally program a 2732A at 25 volts it won't live through the experience! The 2764 and 27128 are programmed at 21 volts whilst the 2764A and 27128A are programmed at 12.5 volts. DON'T MIX THEM UP!

NOTE: Compatible CMOS EPROMS may also be programmed. They will generally require the use of 'DUMB' programming. Consult section 10.0 for the static and dynamic levels used with the various devices to determine compatibility. If you are still in doubt as to the compatibility of a particular manufacturers device send us a copy of the manufacturers data sheet and we will advise you whether it can be programmed or not.

Selection between the various EPROM types is made via a device selection MENU in the associated software. The pin-out, signal level and VPP (programming voltage) incompatibility of the various devices is sorted out by a series of switches on the programmer. When a particular device is selected you will be informed how to set the switches for the device you wish to program. This approach completely eliminates the need for cumbersome 'PERSONALITY MODULES' which are easy to lose and even easier to damage.

Another comforting thought is that there is no combination of switch settings that will damage the programmer provided that an EPROM is not in the socket.

### W A R N I N G

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                             *
* The WINDRUSH UNIVERSAL EPROM PROGRAMMER has been designed for use in        *
*                                                                             *
* development system environments where it will be used in intermittent duty  *
*                                                                             *
* service. It is NOT intended for production programming of EPROMS. Ensure    *
*                                                                             *
* that the programmer is allowed to cool for at least two minutes between     *
*                                                                             *
* programming operations otherwise damage to the +12/21/25 volt switching     *
*                                                                             *
* regulator and its associated circuitry may result.                          *
*                                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

## 3.0 SOFTWARE

The software provided with this programmer has been continuously improved over a period of three years so that it would be be as user friendly as possible. The software is MENU driven for the most part which will enable you to master its many features in short order. To protect you from yourself there are error traps in any prompt that requests data from you. This approach has enabled us to produce a software package that cannot be crashed in normal operation.

Consequently the program resides in slightly less than 9K between $8000 and $A1FF. About 7K of this is actual program code and the remainder contains messages and prompts that are presented to the operator during use.

The software was written in highly optimized 6809 assembly language and is therefore extremely fast (and virtually impossible to port to the 6800!).

## 3.1 SOFTWARE MEMORY MAP

The memory map of the standard 'FLEX' software package, as provided, is as follows:

$0000 - $7FFF EPROM BUFFER AREA

$8000 - $A1FF EPROM PROGRAMMER SOFTWARE
$A200 - $A2FF PROGRAM VARIABLES AND HARDWARE STACK

$E090 - $E094 MC6821 PIA INTERFACE TO EPROM PROGRAMMER (default address)

NOTE: These addresses are approximate only.

The software is not relocatable in it's binary form. A series of binary patches are available which enable the advanced user to modify some of the features of the software to suit his/her own needs. General details are provided in the advanced programmers section.

## 3.2 MEMORY REQUIREMENTS

This package assumes that you have FLEX running in a system with 56K of RAM located between $0000 and $DFFF with FLEX running between $C000 and $DFFF. The user may have programs of his own or extensions to FLEX resident in memory between $A400 and $BFFF as this area of system memory is not used by the programmer software.

## 3.3 SOFTWARE SOURCE FILES

The source code of the EPROM programmer software is available direct from Windrush Micro Systems (it is not available through our dealers). The source code, which has been prepared with our 'MACE' assembler, uses label names 8 characters in length so it may not be compatible with all 6809 assemblers. The source code files will be supplied on disk for $75.00 only after you have completed and returned a non-disclosure agreement.

## 4.0 WHAT YOU SHOULD GET

1. A floppy disk with the following files:

   FLEX VERSION

   UPRII .CMD ... the executable object file
   UPRII-EQ.ASM ... the software tables which may be modified by the user
   LOAD .CMD ... a binary file offset loader for FLEX
   MAP .CMD ... a binary file mapper for FLEX
   READ-ME .TXT ... tells you what is on the disk

   (see addendums for SSB DOS, OS-9 and MDOS/XDOS versions)

2. A PROGRAMMING POD

3. THIS MANUAL

4. AN INTERFACE CABLE (optional)

5. AN INTERFACE BOARD (optional)

   We supply five different interface boards for use with this programmer:

   a. S-30
   b. S-50
   c. EXORCISER/EXORSET/MICROMODULE
   d. EURO-3X (Windrush 3U Eurocard bus)
   e. EURO-6X (Windrush 6U Eurocard bus)

If you have decided to build your own interface board or integrate the programmer into your own hardware you must build an interface similar to the one detailed in the schematic diagrams. Please note that the POD expects to receive unregulated +5 volts at about +8 to +10 volts and unregulted +12 volts at about +14 to +18 volts. If your computer can only supply regulated +5 and regulated +12 you must modify the programmer pod (i.e. by-pass the internal regulators). Please note that any such modifications will void the warranty.

## D I S C L A I M E R

We cannot, and will not, be held responsible for any disasterous side effects of the users attempts to use our programmer pod with an interface or cable assembly not supplied by ourselves. Any attempts to modify the programmer pod, interface board or cable assembly supplied will void the warranty and is done at the risk of the purchaser. This includes, but is not limited to, the users attempts to implement field modification notices approved by Windrush Micro Systems.

## 4.1 GETTING IT UP

To use the EPROM programmer follow these instructions step-by-step:

1. Turn the power on your computer off.

**W A R N I N G**

CONNECTING THE EPROM PROGRAMMER INTERFACE OR POD WITH THE POWER APPLIED WILL (and we mean WILL and not MAY) CAUSE PERMANENT DAMAGE TO THE PROGRAMMER OR THE INTERFACE.

2. Insert the EPROM programmer interface into an appropriate bus slot in your computer.

3. Connect one end of the ribbon cable to the interface board. Observe the polarity key!

4. Route the free end ribbon cable out of your computer and connect it to the programmer pod. Again observe the polarity key!

5. Switch the power to your computer 'ON'.

6. Boot up your disk operating system.

7. Format a blank floppy disk.

8. Copy all of the files from the disk we have supplied onto the freshly formatted disk. Then put the original disk away in a safe place.

**W A R N I N G**

NEVER USE THE DISK WE SUPPLY FOR ANY PURPOSE OTHER THAN MAKING A WORKING COPY OF ITSELF ON A KNOWN GOOD MACHINE. IF YOU CRASH IT, REGARDLESS OF THE REASON, WE WILL CHARGE YOU THE FULL COST OF A NEW DISK TO RESTORE IT!

9. The 'READ-ME' file on the disk you just created will tell you what files you should copy to your normal 'system disk'. Copy these files onto your system disk now.

10. The following section assumes that you have located the interface board at its standard location (refer to section eight for details of standard system configurations).

11. Invoke the eprom programmer software by typing 'UPRII <RETURN>'.

12. A start-up banner should appear informing you of the version number of the software ... proceed to section 5.0 overleaf.

13. READ SECTIONS 5.0 AND SECTIONS 5.1 BEFORE ATTEMPTING TO USE THIS PRODUCT.

## 5.0 OPERATION

The software used to drive the EPROM programmer is controlled by two menus which prompt the operator for an activity or information.

The software is forgiving of errors and will inform you, in PLAIN ENGLISH, what error has occurred and then re-prompt you for the information.

Here are four general guidelines to note:

1. Always remember that no matter what you are doing, no matter what corner you may have managed to paint yourself into, you can INSTANTLY escape from it by simply hitting ^C (CONTROL-C). The CONTROL-C key will be called the <BREAK> key throughout the remainder of this document.

   There are only two exceptions to this. The first is when you have initiated a programming cycle. The second is when you are accessing the operating system. The only way to abort these activities is to use the RESET button on the computer (cringe!). Unfortunately the real-time nature of these activities precludes polling the keyboard for <BREAK>.

2. There are a set of defaults for most prompts. Each of these defaults will be explained in the following sections. The default answer is the most commonly given response to a particular question. Hitting the <RETURN> key signifies that you wish to accept the default. If you do not wish to accept the default value you must specifically type the alternate entry.

   The use of defaults is designed to speed up the activities involved and eliminate some of the repetition (which can become very tedious after a while) when working with a batch of EPROMS.

   The use of defaults makes the assumption that you have gained a little experience with the programmer. We have found from past experience with our other professional products that if we treat the operator like an idiot and prompt him for a specific answer to each question operation of the software becomes very tedious once the operator becomes familiar with the responses expected of him. We designed the software on the assumption that the operator is willing to make a few mistakes initially in exchange for a product that is easier to use in the long term.

3. All references to the buffer (the area in memory where the data associated with the EPROM is being stored) are relative to the EPROM addresses and not necessarily the physical addresses used in the computers memory.

   This feature GREATLY simplifies working with EPROMS as it allows you to forget all about the physical addresses in memory and concentrate on the addresses in the EPROM.

   The buffer address $0000 corresponds to the first byte in the EPROM, i.e. the byte addressed when all address lines to the EPROM are at logic '0'. It just so happens that buffer address $0000 and memory address $0000 are the same in FLEX and SSB DOS systems. This has been done to make use of the binary offset loader supplied a bit easier. This equivalence is not the case in the OS-9 and MDOS/XDOS versions however.

4. NEVER INSERT AN EPROM IN THE ZIF SOCKET UNLESS PROMPTED TO DO SO AND NEVER INSERT OR REMOVE AN EPROM IF EITHER OF THE LEDS ON THE POD ARE ILLUMINATED.

## 5.1 ERROR MESSAGES

1. Most of the prompts for HEX ranges are provided with defaults. The defaults supply the most commonly used answers. The defaults are entered by hitting <RETURN> when the prompt appears. If no default is available you will be so informed by the message:

NO DEFAULTS AVAILABLE

2. Responses to any prompt for a 'BYTE' or an 'ADDRESS' must be given in two or four character HEX respectively. Back-spaces are not allowed during HEX entry. If you accidentally hit a non-hex character (i.e. anything other than 0 - 9, and A - F [or a - f] ) you will be so informed with the message:

NOT A HEX NUMBER

3. When prompted for a START and an END address the END address must be higher in memory, or equal to the START address. If you fail to follow this rule the following message will be issued:

ILLEGAL ADDRESSES

4. When prompted for an EPROM programming range you must enter addresses within the range of the device being programmed. If you fail to do this the following message will be issued:

EPROM LIMITS EXCEEDED

5. When supplying buffer ranges they must fall within the current buffer limits. These limits are usually $0000 to $7FFF. Any prompt that requests buffer address limits will ALWAYS be preceeded by a banner indicating the current buffer limits. Answering any address prompt with an address outside of the current buffer limits will be greeted with the message:

BUFFER LIMITS EXCEEDED

6. When using the MOVE command with the move operation restricted to the buffer, any move that would result in a buffer overflow condition will also give the message:

BUFFER LIMITS EXCEEDED

7. When using the buffer memory EXAMINE/CHANGE or the DUMP function any attempt to go outside of the current buffer limits will result in the message:

BUFFER LIMITS EXCEEDED

8. If there are any problems in the computers RAM you may encounter the next error message when you are using the MEMORY - EXAMINE/CHANGE operation. This error is caused by the data written to the RAM not matching the data read back immediately after the write operation:

MEMORY ERROR

## 5.2 START-UP BANNER

When you cold start the FLEX version of the EPROM programmer by typing 'UPRII <RETURN>' when in FLEX or by executing a 'JUMP' to $8000 when in the SYSTEM MONITOR the following message will appear on the screen.

```
+-----------------------------------------+
|                                         |
|     WINDRUSH MICRO SYSTEMS LIMITED      |
|                                         |
|                                         |
| * * * * * * * * * * * * * * * * * * * * |
| *                                     * |
| *   UNIVERSAL EPROM PROGRAMMER II    * |
| *                                     * |
| * * * * * * * * * * * * * * * * * * * * |
|                                         |
|            VERSION:  X.XX               |
|                                         |
|                                         |
| MC6821  RESIDES  AT: $E090              |
|                                         |
| BUFFER  RESIDES  AT: $0000  -> $7FFF    | <- (varies)
| SOFTWARE RESIDES AT: $8000  -> $A1FF    | <- (varies)
| VARIABLES RESIDE AT: $A200  -> $A2FF    | <- (varies)
|                                         |
| HIT ANY KEY TO CONTINUE!                |
|                                         |
+-----------------------------------------+
```

The startup banner informs you of several things.

1. In this example the default address of the PIA ($E090) has been accepted and the PIA and POD have been proven to be present at the address shown. (or something that looks like a PIA and the POD are present!!)

2. The absolute range of the programmer buffer is presented.

3. The absolute range of the programmer software is presented.

4. The absolute range of the programmer variable storage and stack is presented.

These last three items are there to serve warning that the programmer expects to find RAM in all of these locations.

Hitting any key on the keyboard at this point will take you into the EPROM TYPES MENU described in section 6.0. If you hit <BREAK> you will return to FLEX.

If the screen does not look like the above presentation refer to section 5.3.

N O T E

Cold starting the software does not affect the contents of the programmer buffer. Thus you may move from the programmer software to FLEX or your system monitor without worry about corrupting the contents of the buffer.

Hitting <RESET> will not affect the programmer buffer providing that your reason for using <RESET> is not to recover from an accident involving the 'MOVE' operation. You can use <RESET> at any time with the knowledge that the data currently in the EPROM programmer buffer will be preserved.

## 5.3 STARTUP ERROR MESSAGES

There are two possible error conditions that may exist when the programmer is fired up.

A. The software is unable to find a PIA or its associated POD at the specified default address. If an EPROM is in the ZIF socket when the POD is being initialized the software may be unable to find the POD. If this is the case the following error message will appear:

NO PIA PRESENT AT $XXXX

If you hit any key other than <BREAK> at this point you will get the following prompt:

BASE ADDRESS OF PIA? $

Entering the correct or alternative address of the programmer interface board should then result in the screen display described in the previous section. If it doesn't there is either a fault in the interface board, the interconnecting cable or the programmer POD.

If you hit <BREAK> at this point you will return smoothly to FLEX.

B. If your system does not have 48K of user memory (or at least have memory up to $9FFF) or you or some program you have been running has reset the FLEX 'MEMEND' pointer at $CC2B to a memory location below $9FFF you will get the following message:

TOO CLOSE TO MEMEND! ... PROGRAM ABORTED!

Control of the system will then return to FLEX. The above message will be posted only if RAM exists between $8000 and $80FF as a minimum. FLEX will have attempted to load the entire program however. This will result in anything that was being protected by MEMEND being ovelayed by the EPROM programmer software. If you get this message it is highly advisable to re-boot the system before continuing.

This message is not implemented in the SSB DOS, MDOS/XDOS and OS9 versions of the software.

## 6.0 EPROM TYPES MENU

The EPROM TYPES MENU provides a convenient method for EPROM type selection. Simply look up the EPROM type number in the table and type in the figure (0-9, A-B) adjacent to it. You may also hit the <RETURN> key to accept the default device pointed to by the symbol: (->). The default device pointer may be altered by the user. General information is given in the advanced programmers section.

```
+-------------------------------+
|  SINGLE VOLTAGE DEVICES ONLY! |
|                               |
|        EPROM TYPES MENU       |
|                               |
|     0 = 2758   (INTEL)  1K    |
|     1 = 2716   (INTEL)  2K    |
|     2 = 2532   (TEXAS)  4K    |
|     3 = 2732   (INTEL)  4K    |
|     4 = 2732A  (INTEL)  4K    |
|     5 = 68764  (M'OLA)  8K    |
|     6 = 2564   (TEXAS)  8K    |
| ->  7 = 2764   (INTEL)  8K    |
|     8 = 2764A  (INTEL)  8K    |
|     9 = 27128  (INTEL) 16K    |
|     A = 27128A (INTEL) 16K    |
|     B = 27256  (INTEL) 32K    |
|                               |
|  SELECT TYPE BY NUMBER:       |
+-------------------------------+
```

You will note that the above menu does not contain all of the devices indicated earlier. This has done for the sake of brevity and to improve the appearance of the display. The following is an expansion of specific items in the menu for those who are not familiar with the compatibility of various type numbers:

(0) 2758 (INTEL) ... or ... 2508 (TEXAS)

(1) 2716 (INTEL) ... or ... 2516 (TEXAS)

(5) MCM68764 ... or ... MCM68766 (MOTOROLA)

All CMOS pin compatible devices, notably the 27C16, 27C32, and 27C64 may be used in place of their NMOS counterparts.

A quick note is in order on 2716 EPROMS. As you may, or may not, be aware there is a bit of confusion in the industry over this type number. We have opted to use the INTEL designations instead of the TEXAS designations. This choice is based on popularity of the low cost Hitachi, Mitsubishi and Fujitsu second sources for the INTEL devices.

Make sure that the 2716 EPROM is a single voltage device not a TRI-VOLT device (+5, -5, and +12). The TRI-VOLT types are seldom found in equipment designed in the past two or three years but were quite common in equipment designed in the late 1970's. TRI-VOLT DEVICES ARE NOT SUPPORTED These include the TEXAS TMS2708 and the TEXAS TMS2716, et al.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* USING TRI-VOLT EPROMS MAY PERMANENTLY DAMAGE THE PROGRAMMER *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

## 6.1 SETTING UP THE PROGRAMMER FOR A PARTICULAR DEVICE

**W A R N I N G**

**NEVER INSERT OR REMOVE AN EPROM IF EITHER LED ON THE PROGRAMMER IS ILLUMINATED!**

**NEVER INSERT AN EPROM UNLESS PROMPTED TO DO SO.**

As soon as you enter one of the menu item numbers/letters or <RETURN> (to accept the default EPROM pointed to thus: -> ) the software will prompt you for one of two things.

If you have selected device types (0) through (6) you will be only prompted to set up the mode selection switches.

If you select devices (7), (8), (9), (A) or (B) you will be asked whether you wish to use inteligent (tm) programming or not. Then you will be prompted to set up the mode selection switches.

## 6.2 INTELIGENT PROGRAMMING OPTION

If you select the 2764, 2764A, 27128, 27128A or 27256 devices you will be prompted as follows:

```
+-------------------------------+
|                               |
|      2764   $0000 - $1FFF     |
|      ==================       |
|                               |
|  INTELIGENT PROGRAMMING (TM)  |
|  IS AVAILABLE FOR THIS DEVICE.|
|                               |
|DO YOU WANT TO USE IT? (Y/N) Y | <- default is 'Y'
|                               |
+-------------------------------+
```

If the manufacturer of the device is Intel you should answer the prompt by hitting <RETURN> (the default answer is YES) or 'Y'. You should consult the data sheets of other manufacturers of these devices to determine if the inteligent (tm) programming algorithm can be used. If you are not sure it is best to answer the this question 'N'. Generally speaking CMOS devices must be programmed using the 'DUMB' programming option, i.e. answer the prompt 'N'.

**W A R N I N G**

It is important to ensure that the manufacturer of the device indicates that it can be programmed using Intel's inteligent (tm) programming algorithm as VCC is raised to six (6) volts during programming. If the device is not designed for this level of VCC it may be permanently destroyed if you attempt to use the intelligent programming mode of operation.

## 6.3 WHAT IS INTELIGENT PROGRAMMING

In the past the standard method of programming an EPROM was to apply a 50 millisecond programming pulse to each EPROM location to be programmed. If you were programming a 1K device this meant that you had to apply programming pulses to 1028 locations x .050 seconds = 51.25 seconds. This was fine when devices were only 1K or 2K in size as the programming time was only one or two minutes respectively. This technique became annoying when the 8K devices became available (7 minutes programming time) and absolutely aggravating when the 16K devices became available (14 minutes programming time), and completely impossible with the arrival of the 32K devices (28 minutes programming time)!

As you can see each new generation of devices required a doubling of the programming time. Software driven programmers were able to shorten the times a bit by skipping over any data equal to $FF (no cells require programming). This was not enough. Fortunately Intel Corporation, one of the industry leaders in EPROM design and manufacture, recognized this problem and decided to do something about it.

The Intel engineers noted that over 95% of the cells to be programmed would program in only eight milliseconds. It was only the remaining 5% of the cells that took up to 45 milliseconds to program. Thus the 'DUMB' programming algorithm of applying a 50 millisecond pulse was not really necessary for 95% of the cells being programmed!

Armed with this knowledge Intel Corporation developed the 'INTELigent programming algorithm' which can reduce programming time AND improve the reliability of the programming at the same time. The technique works like this:

a. All intelligent programming is done with VCC elevated to +6 volts. This shifts the threshold of the internal circuitry which senses whether a cell is a logical 0 or a logical 1. This makes the device recognize a logical zero 'later' than it will with VCC at the normal +5 volts. Thus if you read a logical zero when VCC is at +6 volts it will POSITIVELY read logical zero when VCC is in the normal range of +4.75 to +5.25. This provides a higher degree of reliability in the recognition of the point at which a bit becomes programmed to logical zero. (All bits erase to logical one).

b. The programming cycle consists of first applying a nominal 1 millisecond programming pulse. The data is then 'read' from the EPROM and compared with the data we are attempting to program into it. If the data matches we proceed to apply the overprogram pulse.

c. If it fails to match we keep repeating the program-read-compare cycle until either that device programs or a pre-set limit is reached. The software keeps track of the number of 1 millisecond pulses being applied as they will enter the calculations for the overprogram pulse.

d. If the pre-set limit is reached with a 2764 or a 27128 the overprogram pulse will be applied and then the data will be verified one last time. In the case of the 2764A, 27128A and the 27256 reaching the pre-set limit only result in the data being verified one last time. If either device fails to verify at this point it will be rejected.

e. When a given location has been programmed an overprogram pulse is then applied. This is an insurance policy to ensure that the location is programmed well below the 0/1 threshold. The length of the overprogram pulse is determined by the number of 1 millisecond pulses already applied multiplied by a constant. Thus the locations that took the longest to 'take' initially get the largest overprogram pulse.

## 6.3 WHAT IS INTELLIGENT PROGRAMMING (continued)

The inteligent (tm) programming algorithm offers the potential of very substantially reducing programming times and greatly improving programming margin. Since every single location in the device is verified to have programmed correctly and is ensured to have an adequate overprogram margin a completely closed programming loop exists.

In comparison the conventional approach not only increases the programming time but it is also an 'open loop' in respect of the overprogram margin. The overprogram margin relies on the manufacturers to screen out devices that do not program within 45 milliseconds. Since manufacturers do this by batch sampling (100% testing is prohibitively expensive) it is not unheard of having an EPROM 'forget' it's program after 2 or 3 years due to inadequate over-program margin.

The table below illustrates some of typical programming times that can be achieved using INTEL's 'inteligent programming algorithm' and compares them with the times required by the conventional 'dumb' programming algorithm.

| DEVICE | DUMB PROGRAM TIME | INTELIGENT PROGRAM TIME |
|---|---|---|
| 2758 | 1 minute | not available |
| 2716 | 1 minute, 45 sec | not available |
| 2732 | 3 minutes, 30 sec | not available |
| 2764 | 7 minutes | 1 minute, 15 seconds |
| 27128 | 14 minutes | 2 minutes, 30 seconds |
| 27256 | 28 minutes | 5 minutes |

NOTE 1: The intelligent programming times indicated are typical for 2.0 MHZ FLEX systems only. Some devices may take less time to program and some make take longer. OS-9 level 2 systems will be particularly slow due to the system task switching overheads.

It is not unusual for approximately 5% of the devices that you program using the intelligent programming option to take LONGER than the 'DUMB' programming times indicated.

INTELLIGENT PROGRAMMING WORKS ON THE LAW OF AVERAGES!

NOTE 2: Pin compatible CMOS EPROMS must generally be programmed using the 'DUMB' programming operation.

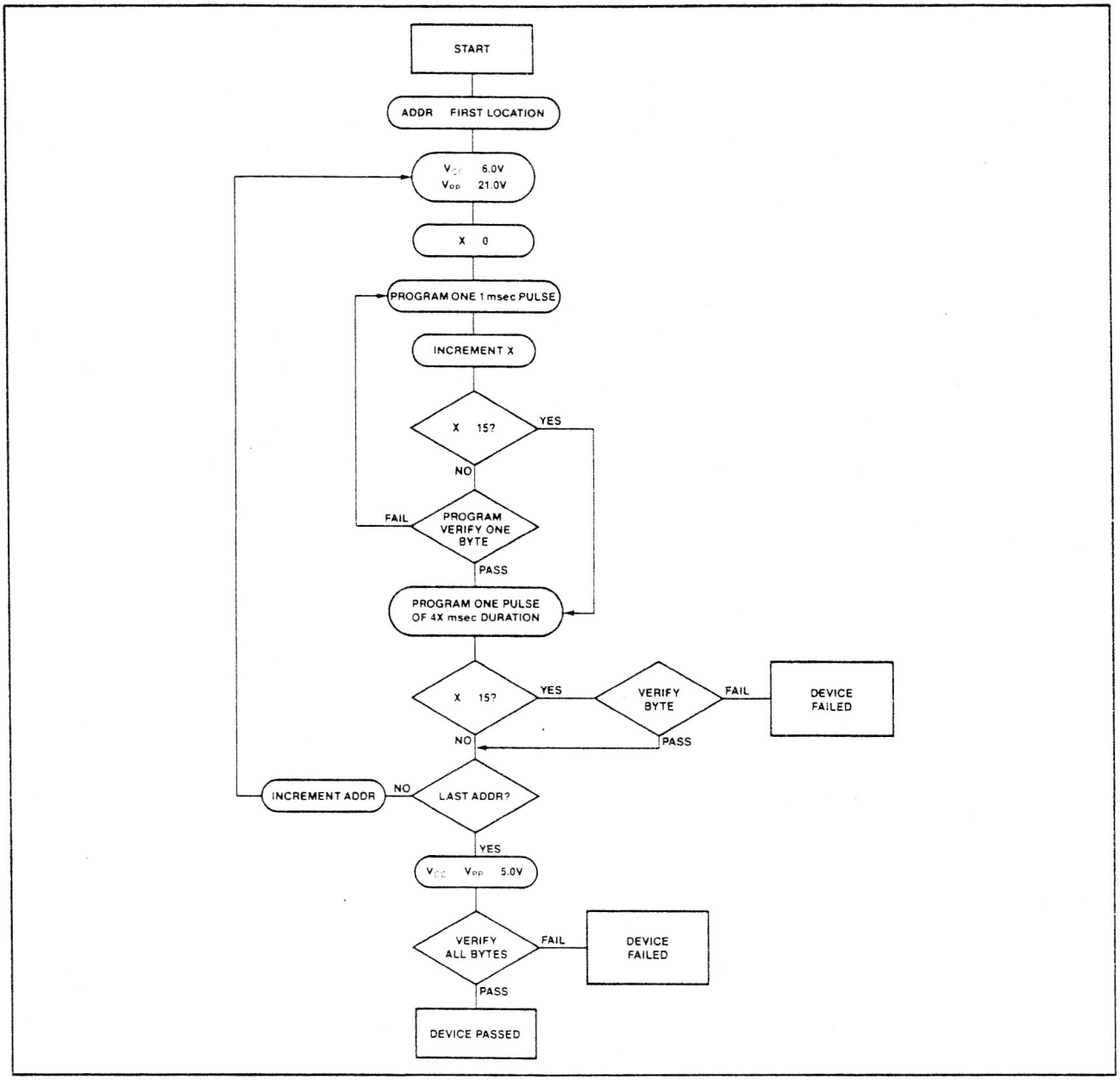
### W A R N I N G

If you are using devices other than those manufactured by INTEL you should consult the data sheets to determine if you can use 'intelligent programming'.

If you wish to learn more about the INTEL 'inteligent programming algorithm' (tm) it is described in detail in the March 17, 1983 and the October 27, 1983 issues of EDN. Further information can be obtained from the INTEL data sheets covering the 2764, 2764A, 27128, 27128A and 27256.

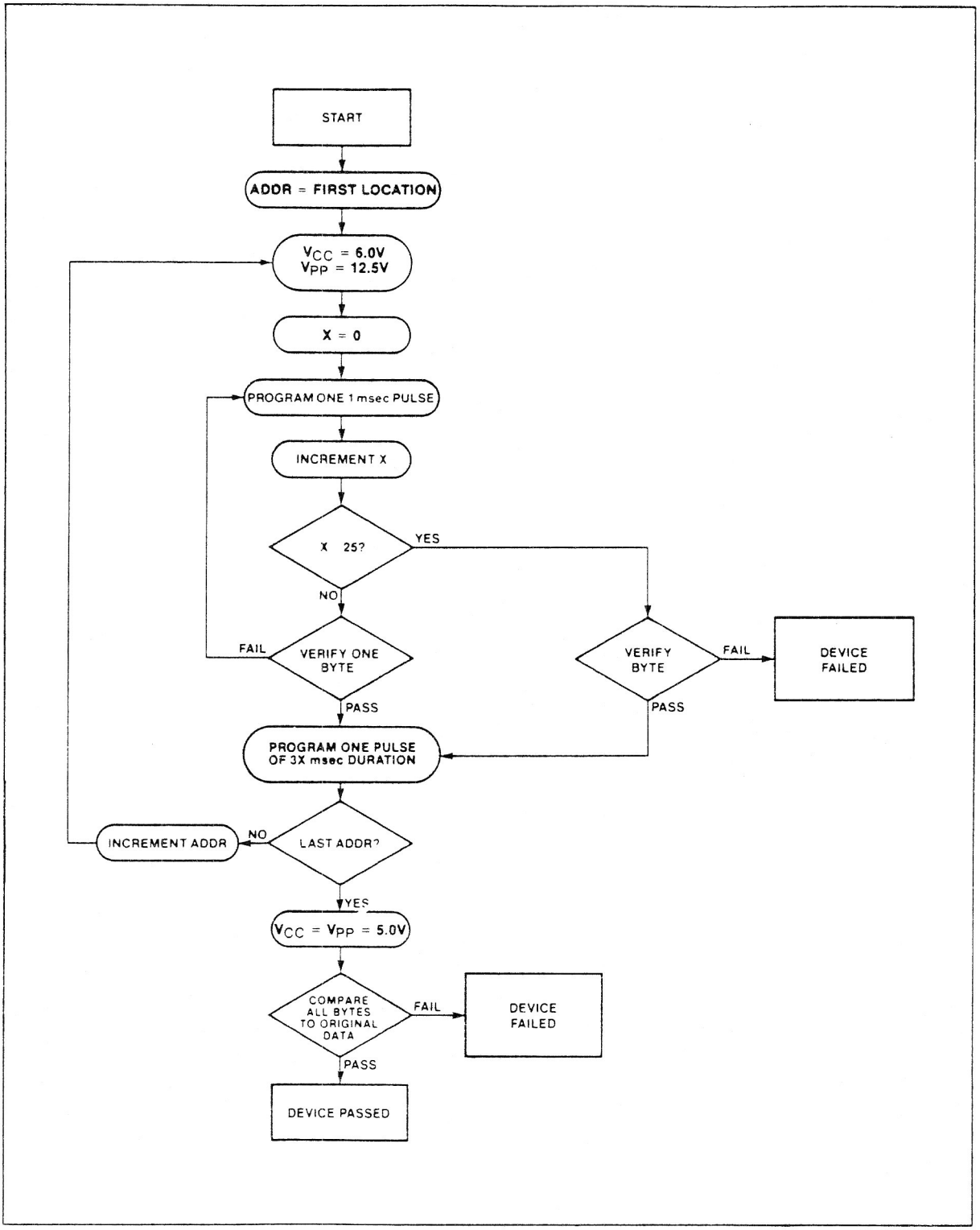
## 6.4 THE 2764 AND 27128 INTELIGENT PROGRAMMING ALGORITHM

The basic algorithms used by this programmer are those specified by INTEL and are illustrated on the following two pages. This information is re-printed with the permission of INTEL who assume no liability for its accuracy and reserve the right to alter the design of their products without prior notice.




27128 intₑligent Programming™ Flowchart

## 6.5 THE 2764A, 27128A AND 27256 INTELIGENT PROGRAMMING ALGORITHM




27256 int$_e$ligent Programming™ Flowchart

## 6.6 SWITCH SETTING INSTRUCTIONS

The variety of devices supported by this product requires some method of catering for the differences in pin-outs, signal levels and programming voltages of the various devices. Rather than have a personality module for each device (which can get lost or damaged ... and usually do!) we have incorporated a set of switches to provide the necessary personality switching facilities.

There are two switch groups used. One, designated 'SWITCH A' comprises 10 SPDT changeover switches, and the other, designated 'SWITCH B' comprises 4 SPST switches. SWITCH A provides the pin-out switching facilities and SWITCH B provides the program voltage (VPP) switching facilities.

A typical set of instructions looks like this:

```
+-------------------------------+
|     2764   $0000 - $1FFF      |
|   SET SWITCHES AS INDICATED   |
|   SWITCH A          SWITCH B  |
|                               |
|  (1)     -->       (1)  <--   |
|  (2)  <--          (2)  <--   |
|  (3)     -->       (3)  <--   |
|  (4)     -->       (4)     -->|
|  (5)  <--                     |
|  (6)  <--                     |
|  (7)  <--                     |
|  (8)     -->                  |
|  (9)     -->                  |
| (10)  <--                     |
|                               |
|HIT ANY KEY TO CONTINUE.       |
+-------------------------------+
```

You should have noticed by now that ever since you made your device selection the device type and its range appear on the top of every prompt presented to you. This is provided as a reminder of what the current device is.

Confirm that you have selected the correct device and then move each of the switches into the positions indicated. The left (<--) and right (-->) designations assume that the end of the POD with the cable entering it is uppermost.

### W A R N I N G

Make sure you get all of the switches in the correct positions. If even one switch in the wrong position it can result in the destruction of the EPROM being programmed, the programmer, or BOTH!

### C A U T I O N

It is important to note that various flags are set in software when a particular device is selected. Even if you could remember all of the different switch settings you should NEVER reset the switches to different settings without first going through the selection menu.

## 7.0 OPERATIONS MENU

After you have set up the EPROM personality switches you will enter this menu. This menu may also be entered by executing a JUMP to the warm start entry address if you are in your system monitor. See section 7.0.9 for further details.

```
+-----------------------------------+
|                                   |
|       2764   $0000 - $1FFF        |
|                                   |
|          OPERATIONS  MENU         |
|                                   |
|  0  FILL  BUFFER WITH A HEX CHAR  |
|  1  MOVE  DATA WITHIN THE BUFFER  |
|  2  EXAMINE - CHANGE THE BUFFER   |
|  3  DUMP  BUFFER IN  HEX & ASCII  |
|  4  CRC   CHECKSUM OF THE BUFFER  |
|  5  COPY  AN EPROM TO THE BUFFER  |
|  6  VERIFY  EPROM AGAINST BUFFER  |
|  7  PROGRAM    EPROM FROM BUFFER  |
|  8  EPROM  (TYPE SELECTION MENU)  |
|  9  MONITOR   (USE WITH CAUTION)  |
|  A  ALL DONE    (RETURN TO FLEX)  |
|  F  FIND   A STRING OF HEX BYTES  |
|  /  FLEX  (EXECUTE FLEX COMMAND)  |
|                                   |
|    SELECT OPERATION BY NUMBER     |
|                                   |
+-----------------------------------+
```

There are thirteen operations available. Each operation will be discussed in detail in the following sections. To select a particular operation simply type in the number (0-9) or the character (A, F or /) adjacent to the description of the operation you require. Any key other than these will result in the menu being re-displayed.

When you have selected a particular operation a banner will appear confirming the operation has been selected. The software will then prompt you for further information or provide further instructions.

The SSB DOS and MDOS versions do not have the '/' operation. The OS9 version does not have operation '9' but it does have two additionals opererations: 'R' (READ) and 'W' (WRITE). The the addendums for further information.

If you accidentally select an operation and wish to return to the OPERATIONS MENU all you have to do is hit the <BREAK> key (see note). You may do this at any time you desire to do so!

This 'escape' facility is provided for those circumstances where you accidentally press the wrong key and end up in some part of the menu you don't really want to be.

N O T E

For the sake of brevity we will be calling the 'CONTROL-C' key the <BREAK> key and the 'CARRIAGE-RETURN' key the <RETURN> key throughout this document.

## 7.0.0 FILL BUFFER AREA WITH A HEX CHARACTER

This operation is provided to fill any portion of EPROM buffer area between $0000 and $7FFF with a specified HEX byte. It is most commonly used to initialize the buffer to $FF before loading the buffer with data stored on cassette. This operation is also available in the cassette menu for this reason.

When you select this operation you will be prompted for the HEX FILL BYTE, the START ADDRESS, and the END ADDRESS as follows:

```
+-----------------------------------+
|     2764   $0000 - $1FFF          |
|     ====================          |
|                                   |
|          FILL  BUFFER             |
|                                   |
| BUFFER RANGE = $0000 -> $7FFF     |
|                                   |
| ENTER HEX FILL BYTE:  $FF         | <- default is $FF
| ENTER START ADDRESS:  $0000       | <- default is $0000
| ENTER  END  ADDRESS:  $7FFF       | <- default is $7FFF
|                                   |
+-----------------------------------+
```

The only legal entries are as follows:

1. The HEX FILL BYTE must be within $00 - $FF.

2. The STARTING ADDRESS must be within $0000 - $7FFF.

3. The END ADDRESS must be within $0000 - $7FFF. The START address must also be less than or equal to (<=) the END address.

4. Hitting <RETURN> (to accept the defaults) at each prompt will yield the display above.

There are three possible errors that can be made during data entry for this operation.

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraph 5)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

```
+-----------------------------------+
|                                   |
|        FUNCTION COMPLETED         |
|                                   |
|                                   |
|HIT ANY KEY TO CONTINUE!           |
|                                   |
+-----------------------------------+
```

Pressing any key will return you to the OPERATIONS MENU.

### 7.0.1 MOVE DATA WITHIN THE BUFFER

This operation allows you to move code within the confines of the buffer area OR to gain access to the absolute memory map for the same purpose.

The first option is completely safe as far as crashing the system is concerned, i.e. it is impossible to crash the system if the move is restricted to the buffer. All addresses are checked and any address that is outside of the current buffer limits or a move that tries to overflow the buffer will be greeted with an error message. This form of the move operation is most commonly used to build up a large binary file from several small EPROMS for programming into a large EPROM or saving out to disk. Alternatively it can be used to take the code from one large EPROM and break it up into several smaller ones.

The second option, which can be quite dangerous, is provided to gain access to the absolute memory map, i.e. it turns the entire 64K memory map into the programmer buffer. This allows you, for example, to move the code in your system monitor ROM into the buffer area. Once the code is in the buffer area it can be modified and programmed into a new EPROM or saved out to disk.

**W A R N I N G**

The nature of the second form of the move operation will enable you to move code right on top of the EPROM programmer software, variable storage and stack in the $8000 - $A2FF area or the DOS area between $C000 - $DFFF. If you do this the system will crash in no uncertain terms. Therefore when you are using the move operation to access the absolute memory map be very careful of the addresses you supply.

```
* * * * * * * * * * * * * * *
*                           *
*  YOU HAVE BEEN WARNED!    *
*                           *
* * * * * * * * * * * * * * *
```

NOTE: The second form of this option is not available in the OS-9 version for obvious reasons.

## 7.0.1 MOVE DATA WITHIN THE BUFFER (continued)

When this operation is selected you will first be prompted to restrict the move to the buffer. The default answer is 'Y' so hitting <RETURN> will give you the display below. You will then be prompted for a START address, an END address and the DESTINATION address as follows:

```
+-------------------------------+
|     2764   $0000 - $1FFF      |
|     ====================      |
|                               |
|           MOVE  DATA          |
|                               |
|  RESTRICT TO BUFFER?  (Y/N) Y | <- default is 'Y'
|                               |
| BUFFER RANGE = $0000 -> $7FFF |
|                               |
| ENTER START ADDRESS:  $1000   | <- no defaults available
| ENTER  END  ADDRESS:  $1FFF   | <- no defaults available
| DESTINATION ADDRESS:  $0000   | <- no defaults available
+-------------------------------+
```

Note that the current range of the buffer is displayed and that there are no default values for any of the address prompts.

The 'START' address means the address of the first byte in the block of bytes to be moved. The 'END' address means the address of the last byte in the block of bytes to be moved. The 'DESTINATION' address means the address where you want the first byte in the block being moved to be placed; all subsequent bytes will be located above this address. The two blocks may overlap.

If you attempt to make a block move that will result in a buffer overflow the move will not be performed and you will be greeted with an error message.

There are four possible errors that can be made during data entry for this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

4. NO DEFAULTS AVAILABLE (see section 5.1, paragraph 1)

## 7.0.1 MOVE DATA WITHIN THE BUFFER (continued)

If you choose not to restrict the move to the buffer the display will look like the following:

```
+-----------------------------------+
|     2764   $0000 - $1FFF          |
|     ====================          |
|                                   |
|           MOVE  DATA              |
|                                   |
|  RESTRICT TO BUFFER? (Y/N) N      | <- default is 'Y'
|                                   |
|    ABSOLUTE ADDRESSES APPLY       |
|                                   |
|  PROG BUFFER = $0000  -> $7FFF    |
|                                   |
| ENTER START ADDRESS:  $F000       | <- no defaults available
| ENTER  END  ADDRESS:  $FFFF       | <- no defaults available
| DESTINATION ADDRESS:  $0000       | <- no defaults available
+-----------------------------------+
```

In the above example we are moving the contents of the SYSTEM MONITOR ROM between $F000 and $FFFF to the programmer buffer area between $0000 and $7FFF. The code will be moved to the buffer starting at $0000 and will end at $1FFF.

There are three possible errors that can be made during data entry for this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

3. NO DEFAULTS AVAILABLE (see section 5.1, paragraph 1)

Once the move is completed the 'FUNCTION COMPLETED' message illustrated in section 7.0.0 will appear. Hitting any key will return you to the OPERATIONS MENU.

**W A R N I N G**

THERE ARE NO LIMIT CHECKS ASSOCIATED WITH THE DESTINATION ADDRESS IN THIS OPERATION. IF YOU MAKE A MISTAKE AND OVERWRITE THE EPROM PROGRAMMER SOFTWARE OR ITS VARIABLE SPACE OR THE DOS ITSELF YOU WILL PROBABLY CRASH THE SYSTEM.

### 7.0.2 EXAMINE OR CHANGE THE BUFFER

This operation will provide a MEMORY EXAMINE and CHANGE function very similar to the one most system monitors give you. Hitting the (SPACE-BAR), (+) or (line feed) will increment the address being examined. Hitting (U), (-), or (^ ... caret/up arrow) will decrement the address being examined. The contents of the memory location being examined may be changed by typing in the desired HEX byte.

Selecting this operation will prompt you for a STARTING address. The STARTING address can be anywhere within the current BUFFER RANGE displayed. The default address, which may be selected by hitting <RETURN>, is the first address of the buffer range. The initial display might look something like this:

```
+-------------------------------+
|     2764   $0000 - $1FFF      |
|     ==================        |
|                               |
|    EXAMINE OR CHANGE BUFFER   |
|                               |
|                               |
| BUFFER RANGE = $0000 -> $7FFF |
|                               |
| ENTER START ADDRESS:  $0000   | <- default is $0000
+-------------------------------+
```

As soon as you enter the start address the display might look like this if you executed the example in section 7.0.0:

```
+ key hit                +-------------------------+
+ key hit                |0000 FF +                |
55 (HEX number)          |0001 FF +                |
AA (HEX number)          |0002 FF 55               |
- key hit                |0003 FF AA               |
- key hit                |0004 FF -                |
<BREAK>                  |0005 AA -                |
                         |0006 55                  |
                         +-------------------------+
```

The initial display will contain the address '0000' and its contents 'FF'.

We then hit the (+) key two times to increment through the buffer one memory location at a time. We could just have easily used the (SPACE-BAR) or the (line feed) key. You should use the key you find the easiest to remember.

We then enter the hex number '55'. The display will automatically step to the next memory location. There we enter the hex number 'AA' which again will cause the display to step to the next memory location.

We then hit the (-) key twice to decrement through the buffer one memory location at a time. We could just have easily used the (U) key 'UP' or the (^) (caret/up-arrow) key. Again you should use the key you find the easiest to remember.

To return to the OPERATIONS MENU we simply hit the <BREAK> key.

There are three errors that can be made during data entry for this operation.

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)
2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)
3. MEMORY ERROR (see section 5.1, paragraph 8)

### 7.0.3 FORMATTED DUMP OF THE BUFFER

The formatted dump provides a fast method of examining large blocks of the programmer buffer to pick out code segments or ASCII strings. The dump may be directed to the system console (default) or to the system printer through the standard FLEX printer drivers (at $CCC0 - $CCF7).

The basic format of the screen and printer dumps are the start address followed by the next sixteen bytes in HEX followed by their ASCII equivalents. When output is directed to the system console (default) the dump will be made in 256 byte blocks. When output is directed to the system printer the dump will be made in 1K byte blocks.

The latter format is ideally suited to 80 column by 66 line printers. When output is directed to the printer two blank lines are ejected at the bottom of each page to keep the dump printout centred on the page. The format of the printed dump can be altered to suit the users printer. Consult the advanced programmers section for further information.

The standard HEX/ASCII DUMP format is to dump the buffer codes in HEX and then display the ASCII equivalents to help you spot strings or data tables. When the dump is being directed to the screen or the printer all codes within the range of $20 to $7D will have their ASCII equivalents displayed on the screen. All codes from $00 thru $1F and codes $7E thru $FF will be displayed as a period (.). Since this range of codes includes things like clear screen, carriage return, line-feed, graphics codes, etc, it is not possible to predict how the terminal or printer will handle them.

When this operation is selected the following display will appear:

```
+-------------------------------+
|     2764   $0000 - $3FFF      |
|     ====================      |
|                               |
|          FORMATTED DUMP       |
|                               |
| BUFFER RANGE = $0000 -> $7FFF |
|                               |
| SCREEN OR PRINTER?  (S/P) S   | <- default is 'S'
|                               |
| ENTER START ADDRESS:  $0000   | <- default is $0000
+-------------------------------+
```

Here we have selected to dump to the screen by entering 'S'. Since the (S)creen is the default we could also have selected it by hitting <RETURN>.

Next you will be prompted for an address to start the dump. Note that the current buffer range is displayed to remind you what the current buffer limits are. The START address supplied must fall within this range or an error will be reported.

If you supply an address toward the end of the buffer, $7FFD for example, you will automatically be given an error message. This is because the DUMP routine will not permit a dump past the end of the buffer. If it started to DUMP the contents of the buffer at $7FFD it would exceed the buffer limits before it got to the fourth byte. In these circumstances it will refuse to do the DUMP and re-prompt you for another starting address.

## 7.0.3 FORMATTED DUMP OF THE BUFFER (continued)

Once a valid address is entered the dump will commence as illustrated below:

```
1000 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20   ................
1010 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30   !"#$%&'()*+,-./0
1020 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40   123456789:;<=>?@
1030 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
1040 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60   QRSTUVWXYZ[\]^_`
1050 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70   abcdefghijklmnop
1060 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80   qrstuvwxyz{|}...
1070 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90   ................
1080 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0   ................
1090 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0   ................
10A0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0   ................
10B0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0   ................
10C0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0   ................
10D0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0   ................
10E0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 00   ................
10F0 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10   ................ MORE?
```

As soon as the first 256 byte block has been dumped output will cease and the prompt 'MORE?' will appear. Hitting any key other than 'N' or <BREAK> will give you a second 256 byte dump. Hitting either of these keys will return you to the OPERATIONS MENU.

NOTE: This example used a buffer that was pre-initialized with an incrementing byte pattern. The actual data presented in the dump will depend on what the contents of the buffer are at the time of the dump.

Only two errors are possible with this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

## 7.0.3 FORMATTED DUMP OF THE BUFFER (continued)

If you wish to direct output to the printer the display would look something like the following:

```
+-----------------------------------+
|     2764   $0000 - $3FFF          |
|     ====================          |
|                                   |
|          FORMATTED DUMP           |
|                                   |
| BUFFER RANGE = $0000 -> $7FFF     |
|                                   |
| SCREEN OR PRINTER?  (S/P) P       | <- default is 'S'
|                                   |
| ENTER START ADDRESS:  $1000       | <- default is $0000
| ENTER  END  ADDRESS:  $10FF       | <- default is end of
|                                   |    EPROM.
|   SET PRINTER TO TOP OF FORM      |
|                                   |
|HIT ANY KEY TO CONTINUE!           |
+-----------------------------------+
```

In the above example we have elected to dump the buffer to the (P)rinter by entering 'P' when the prompt 'SCREEN OR PRINTER?' appeared. We are then prompted for a START address and an END address as well. This enables large dumps to be performed without having to keep answering the 'MORE?' prompt.

After you have supplied the END address a carriage return will be sent to the printer to ensure that it starts the dump in the left column. Next you will be prompted to set your printer to the top of form and hit any key. Hitting any key (other than <BREAK>) will result in a buffer dump in the following format being directed to the printer:

```
1000 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20   ................
1010 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30   !"#$%&'()*+,-./0
1020 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40   123456789:;<=>?@
1030 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50   ABCDEFGHIJKLMNOP
1040 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F 60   QRSTUVWXYZ[\]^_`
1050 61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70   abcdefghijklmnop
1060 71 72 73 74 75 76 77 78 79 7A 7B 7C 7D 7E 7F 80   qrstuvwxyz{|}...
1070 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F 90   ................
1080 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F A0   ................
1090 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF B0   ................
10A0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF C0   ................
10B0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF D0   ................
10C0 D1 D2 D3 D4 D5 D6 D7 D8 D9 DA DB DC DD DE DF E0   ................
10D0 E1 E2 E3 E4 E5 E6 E7 E8 E9 EA EB EC ED EE EF F0   ................
10E0 F1 F2 F3 F4 F5 F6 F7 F8 F9 FA FB FC FD FE FF 00   ................
10F0 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10   ................
```

As soon as the dump is completed the 'FUNCTION COMPLETED' message will appear on the screen. Hitting any key will return you to the OPERATIONS MENU.

If you wish to stop the dump to the printer at any time simply hit the <BREAK> key. Since most printers have incoming character buffers it may take a few seconds for the printer to stop.

7.0.3 FORMATTED DUMP OF THE BUFFER (continued)

Any one of four errors can occur during data entry for this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

4.
```
* * * * * * * * * * * * * * * * * *
*                                 *
*        W A R N I N G            *
*                                 *
*     NO PRINTER DRIVERS!         *
*                                 *
* * * * * * * * * * * * * * * * * *
```

If this message appears it means that the programmer software found an RTS instruction ($39) at $CCCO (the location of the printer initialization routine). This generally signifies that the operator has failed to load a printer driver before calling the programmer software. It is generally a good idea to load your printer driver as part of your FLEX 'STARTUP' file. See your FLEX manual for a description of the STARTUP file and the requirements for printer drivers.

W A R N I N G

If you direct output to a printer which is not connected and 'on line' or is not working properly the system may hang up. The only way to recover from this situation if you cannot 'restore' the printer is to RESET the computer and re-start the FLEX and SSB versions of the software by executing a JUMP to $8000 (COLD START) or a JUMP to $8003 (WARM START). The MDOS entry points are $3000 and $3003 respectively.

NOTE: The MDOS and SSB DOS versions do not have the printer drivers implemented as there are no standard locations for the entry points. They may be implemented by the user however. General details are provided in the advanced programmers section.

## 7.0.4 CRC CHECKSUM OF BUFFER

This operation performs a 'cyclic redundancy code check' over a specified block of the programmer buffer and produces a 16-bit checksum. This CRC checksum is a quick, and extremely accurate, method of determining if the code in the EPROM buffer has changed. The CRC checksum is also useful in identifying or verifying EPROMS or code over a block of memory as the numbers produced with the CRC technique are virtually unique.

A CRC check is significantly more accurate than a simple 16-bit checksum formed by addition and as a consequence takes approximately 10 seconds to perform over the entire 32K of the programmer buffer. The differences between the accuracy and reliability two techniques are substantial. In a CRC check every bit of every byte and its position in memory are parameters in the formation of the checksum. The primary advantage of a CRC over a simple additive checksum is that errors do not tend to cancel one another.

When you select this operation you will be prompted for a START and an END address as illustrated below:

```
+-----------------------------------+
|     2764   $0000 - $1FFF          |
|     ====================          |
|                                   |
|          CRC CHECKSUM             |
|                                   |
| BUFFER RANGE = $0000 -> $7FFF     |
|                                   |
| ENTER START ADDRESS:  $0000       | <- default is $0000
| ENTER END ADDRESS  :  $1FFF       | <- default is end of EPROM
|                                   |
|      PERFORMING CRC CHECK         |
|                                   |
|        CRC CKSM = $CD7F           |
|                                   |
|                                   |
|HIT ANY KEY TO CONTINUE!           |
+-----------------------------------+
```

The default START address is the first address in the EPROM range shown at the top of the banner. To enter the default START address simply hit <RETURN> when this prompt appears. The default END address is the second address in the EPROM range at top of the banner. To enter the default END address simply hit <RETURN>.

THE DEFAULT ADDRESSES ARE THE FIRST AND LAST ADDRESSES OF THE EPROM

Once the address range for the CRC is entered the message PERFORMING CRC CHECK will appear. As soon as the CRC has been calculated it will appear as indicated and you will then be prompted to hit any key. Hitting any key at this point will return you to the OPERATIONS MENU.

There are three possible errors that can be made during data entry for this operation:

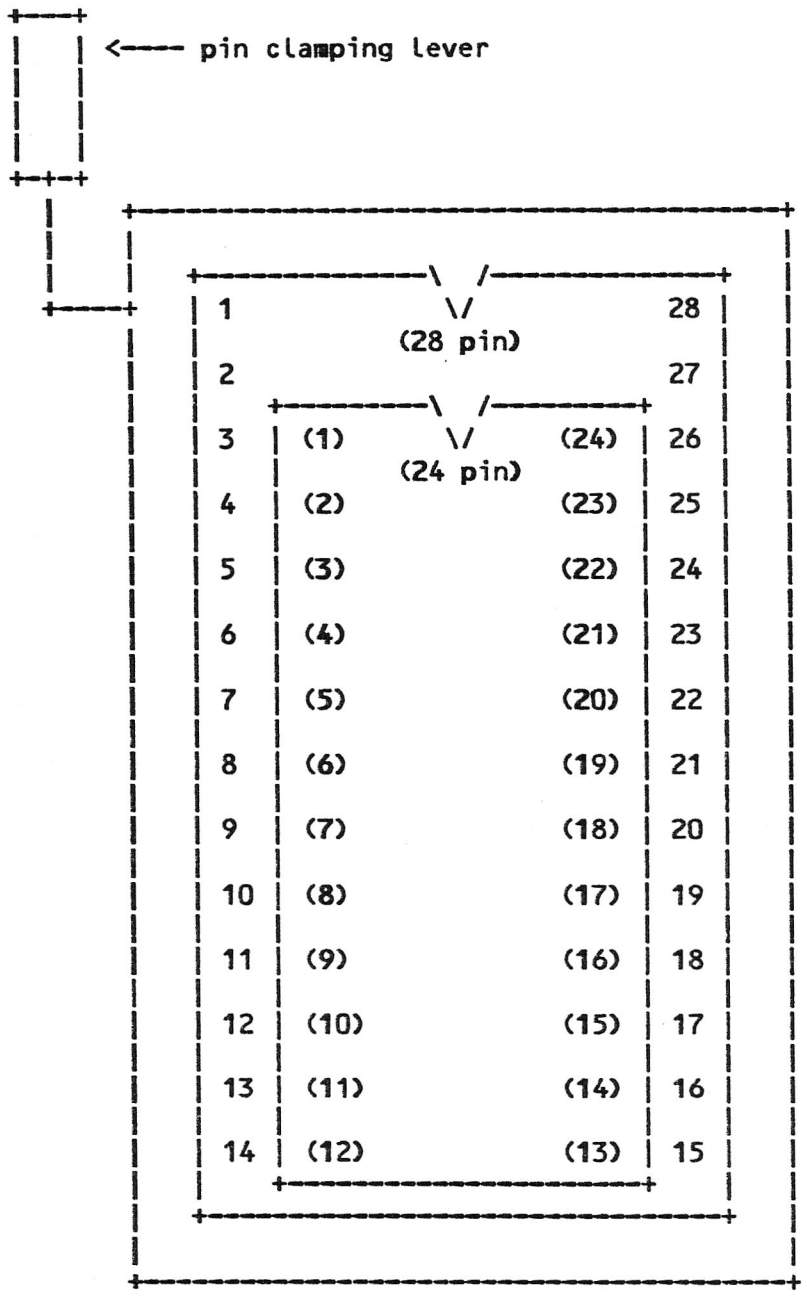
1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

## 7.0.X DEVICE ORIENTATION

We felt that this would be a good place to define the orientation of the ZIF socket as it is right in the vicinity of the three operations that will use it, viz: COPY, VERIFY and PROGRAM.




Pin numbers within brackets () are the pin assignments of the ZIF when any of the 24 pin EPROMs have been selected. The other pin numbers are the pin assignments of the ZIF socket when any of the 28 pin EPROMS have been selected.

**W A R N I N G**

**NEVER INSERT OR REMOVE AN EPROM IF EITHER LED ON THE PROGRAMMER IS ILLUMINATED!**

**NEVER INSERT AN EPROM UNLESS PROMPTED TO DO SO**

## 7.0.5 COPY AN EPROM INTO THE BUFFER

This operation enables you to copy a MASTER EPROM (or part of a MASTER EPROM) into the buffer. You may then VERIFY other EPROMS against the buffer, PROGRAM other EPROMS from the buffer, or save the buffer area on to disk.

You can also alter the buffer area by returning to the OPERATIONS MENU and using the MOVE operation, FILL BUFFER operation, or the EXAMINE & CHANGE operation after you have copied the MASTER PROM. This enables minor modifications to PROM'd code to be accomplished with ease.

If you have selected any of the 28 pin devices, i.e. EPROM select menu items (6) through (9) the following display will appear on the screen:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|            COPY EPROM             |
|                                   |
|                                   |
|       PLACE EPROM IN SOCKET       |
|                                   |
|   READY TO CONTINUE? (Y/N) Y      | <- default is 'Y'
+-----------------------------------+
```

If, however, you have selected any of the 24 pin devices, i.e. EPROM select menu items (0) through (5) a message similar to the following will appear:

```
+-----------------------------------+
|       68764  $0000 - $1FFF        |
|       ====================        |
|                                   |
|            COPY EPROM             |
|                                   |
|* * * * * * * * * * * * * * * * * *|
|*                                 *|
|*        W A R N I N G            *|
|*                                 *|
|*  INSERT THE  EPROM  AT THE      *|
|*  BOTTOM OF THE ZIF SOCKET!      *|
|* * * * * * * * * * * * * * * * * *|
|                                   |
|       PLACE EPROM IN SOCKET       |
|                                   |
|   READY TO CONTINUE? (Y/N)        | <-- default is 'Y'
+-----------------------------------+
```

Hitting any key other than 'N' at this juncture will result in the range prompt on the following page. If you hit 'N' you will be returned to the OPERATIONS MENU.

W A R N I N G

NEVER INSERT OR REMOVE AN EPROM IF EITHER LED ON THE PROGRAMMER IS ILLUMINATED!

NEVER INSERT AN EPROM UNLESS PROMPTED TO DO SO

## 7.0.5 COPY AN EPROM INTO THE BUFFER (continued)

At this juncture you are expected to have inserted an EPROM into the ZIF socket. If you got here by mistake you can still hit <BREAK> and return to the OPERATIONS MENU.

```
+-----------------------------------+
|        2764   $0000 - $1FFF       |
|        ====================       |
|                                   |
|              COPY EPROM           |
|                                   |
|   ENTER START ADDRESS:  $0000     | <-- default is $0000
|   ENTER  END  ADDRESS:  $1FFF     | <-- default is end of
|                                   |     EPROM.
+-----------------------------------+
```

As with virtually all of the commands associated with this programmer you are given a range prompt. This enables you to copy only a small section of a ROM into the buffer thus enabling you to build up a second ROM from bits and pieces of other ROMs using this command and the MOVE command.

Since most operations will be to read the entire device the defaults have been so established. Hitting <RETURN> followed by <RETURN> will copy the entire device into the buffer. Alternatively you can supply hex addresses.

As soon as you have entered the last digit of the 'END' address power will be applied to the EPROM socket. This will be signified by the 'VCC' LED on the programmer illuminating briefly and then extinguishing.

N O T E

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                             *
*  THIS  OPERATION  WILL  DESTROY  THE  PREVIOUS  CONTENTS   *
*  OF THE BUFFER OVER THE SPECIFIED RANGE OF THE EPROM.       *
*                                                             *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Once the copy of the EPROM has been made the 'FUNCTION COMPLETED' message and the prompt 'HIT ANY KEY TO CONTINUE' will appear. Pressing any key will return you to the OPERATIONS MENU.

Three error messages are possible with this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

### 7.0.6 VERIFY AN EPROM AGAINST THE CONTENTS OF THE BUFFER

This operation compares the contents of an EPROM against the data in the buffer, byte-by-byte over a specified range of the device. The data in the buffer can originate from a MASTER PROM using the 'COPY' operation, can be loaded into the buffer area beforehand by using the FLEX 'LOAD' operation (explained later), may entered in the buffer by hand using the EXAMINE/CHANGE operation, or may consist of data moved into the buffer by using the MOVE operation.

When this operation is selected the following display will appear on the screen:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|           VERIFY EPROM            |
|                                   |
|                                   |
|       PLACE EPROM IN SOCKET       |
|                                   |
|    READY TO CONTINUE?  (Y/N)      | <- default is 'Y'
+-----------------------------------+
```

If you are working with a 24 pin device a warning identical to the one illustrated in the previous section will appear.

Hitting any key other than 'N' or <BREAK> will result in a range prompt thus:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|           VERIFY EPROM            |
|                                   |
|  ENTER START ADDRESS:  $0000      | <-- default is $0000
|  ENTER  END  ADDRESS:  $1FFF      | <-- default is end of
|                                   |     EPROM.
+-----------------------------------+
```

As with the COPY operation the defaults are the entire range of the device. Hitting <RETURN> twice will result in the display above. Alternatively you can supply a range of addresses.

As soon as you have entered the last digit of the 'END' address power will be applied to the EPROM socket. This will be signified by the 'VCC' LED on the programmer illuminating briefly and then extinguishing.

**W A R N I N G**

NEVER INSERT OR REMOVE AN EPROM IF EITHER LED ON THE PROGRAMMER IS ILLUMINATED!

## 7.0.6 VERIFY AN EPROM AGAINST THE CONTENTS OF THE BUFFER (continued)

Three error messages are possible during data entry for this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

If all is well the display will look like the following:

```
+-----------------------------------+
|                                   |
|      VERIFICATION COMPLETED       |
|                                   |
|     VERIFIED: $0000 -> $1FFF      |
|                                   |
|       PERFORMING CRC CHECK        |
|                                   |
|         CRC CKSM = $0000          |
|                                   |
| HIT ANY KEY TO CONTINUE!          |
+-----------------------------------+
```

Hitting any key at this point will take you back to the OPERATIONS MENU.

The only error that can occur during this operation once it has been initiated is the EPROM data failing to match the data in the buffer. In these circumstances the following error message will appear:

```
+-----------------------------------+
|                                   |
|    FAILED TO VERIFY AT: $XXXX     |
|                                   |
|     CONTINUE VERIFY? (Y/N)        | <- default is 'N'
|                                   |
+-----------------------------------+
```

If you wish to abort the VERIFY operation simply hit <RETURN> (the default is 'NO') and you will be returned to the OPERATIONS MENU. If you wish to continue the VERIFY simply hit 'Y'. The VERIFY operation will then continue until another location fails to verify, which will result in an identical error message, or until the end of the EPROM is reached, which will result in the prompt 'HIT ANY KEY TO CONTINUE' appearing. Hitting any key will then return you to the OPERATIONS MENU.

N O T E

THIS OPERATION DOES NOT ALTER THE CONTENTS OF THE BUFFER

## 7.0.7 PROGRAM AN EPROM FROM THE CONTENTS OF THE BUFFER

This operation enables you to program an erased or partially programmed EPROM from data contained in the buffer. The data in the buffer can originate from any of the sources described in the previous section.

When this operation is selected the following will appear on the screen:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|          PROGRAM  EPROM           |
|                                   |
|                                   |
|       PLACE EPROM IN SOCKET       |
|                                   |
|   READY TO CONTINUE? (Y/N) Y      | <- default is 'Y'
+-----------------------------------+
```

Note: If a you are working with a 24 pin device a warning identical to the one in section 7.0.5 will be displayed.

Hitting <RETURN> or 'Y' at this point will result in the following:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|          PROGRAM  EPROM           |
|                                   |
|   ENTER START ADDRESS:  $0000     | <- $0000 or last data
|   ENTER  END  ADDRESS:  $1FFF     | <- end of EPROM or
|                                   |    last data.
|      PERFORMING ERASE CHECK       |
|                                   |
|      NOT ERASED AT   $1000        |
|                                   |
|   CONTINUE ERASE CK? (Y/N)  N     | <- default is 'N'
|                                   |
|   PROGRAM ANYWAY?    (Y/N)  N     | <- default is 'N'
+-----------------------------------+
```

The above example illustrates the sequence of prompts that occur for this operation once the EPROM is installed in the ZIF socket. First you will be prompted for the range of the EPROM you wish to program. The first prompt is for the START address. Hitting <RETURN> will automatically enter the first address of the EPROM as indicated in the banner. If you only wish to program a selected range of the EPROM you can enter the address of the first location you wish to program. Next you will be prompted for the END address. Hitting <RETURN> will automatically enter the second address of the EPROM as indicated in the banner. If you only wish to program a selected range you should enter the address of the last location to be programmed. The START and END addresses may be identical thus allowing programming down to a single byte to be accomplished.

There are three possible errors that can be made during data entry:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)
2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)
3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

## 7.0.7 PROGRAM AN EPROM FROM THE CONTENTS OF THE BUFFER (continued)

### EPROM ERASE CHECK

After you have entered valid START and END addresses power will be applied to the EPROM. This will be signified by the 'VCC' LED on the programmer illuminating. The EPROM will then be checked over the range specified to ensure that all locations to be programmed have been erased, i.e. they contain $FF. If any location does not contain $FF you will be advised of the address and asked if you wish to continue the erase check. In the example on the preceeding page we have answered the prompt 'NO'. This results in a second prompt appearing asking us if you wish to program the device knowing that one or more locations may not be erased. This facility is provided primarily to enable single bit locations to be re-programmed and will seldom be required in normal service.

The normal sequence of events during the programming operation looks like this:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|          PROGRAM  EPROM           |
|                                   |
|   ENTER START ADDRESS:  $0000     | <- $0000 or last data
|   ENTER  END  ADDRESS:  $1FFF     | <- end of EPROM or
|                                   |    last data.
|      PERFORMING ERASE CHECK       |
|                                   |
|      ERASE CHECK  COMPLETED       |
|                                   |
|   START PROGRAMMING? (Y/N)  Y     | <- default is 'Y'
+-----------------------------------+
```

Here we have again specified that we wish to program the entire device which may be done by hitting <RETURN> two times in succession. This time the erase check is performed satisfactorily and you are given one last chance to abort the programming operation. Hitting <RETURN> at this point starts the programming operation which is signified by the 'VPP' LED on the programmer illuminating and the following appearing on the screen:

```
+-----------------------------------+
|       2764   $0000 - $1FFF        |
|       ====================        |
|                                   |
|   PROGRAMMING: $0000 -> $1FFF     |
|                                   |
|* * * * * * * * * * * * * * * * * *|
|*                                 *|
|*       INTELIGENT (TM)           *|
|*                                 *|
|*   PROGRAMMING IN PROGRESS       *|
|*                                 *|
|* * * * * * * * * * * * * * * * * *|
+-----------------------------------+
```

The above display indicates that INTELIGENT programming is in operation. If the device is not capable of being programmed in this manner or you have intentionally selected 'DUMB' programming the 'INTELIGENT (TM)' line will not be present.

## 7.0.7 PROGRAM AN EPROM FROM THE CONTENTS OF THE BUFFER (continued)

If you are using the inteligent programming algorithm and it fails to program a particular location it will abort the programming operation and issue the following prompt:

```
+-----------------------------+
|  FAILED TO PROGRAM AT: $XXXX |
|                             |
|    HIT ANY KEY TO CONTINUE.  |
+-----------------------------+
```

Once the programming cycle has been completed the 'VPP' and 'VCC' LED's on the programmer will extinguish. Approximately 2 seconds later the the 'VCC' LED illuminate again. This signifies that the post-program verify cycle has begun. After the verify cycle has ended the 'VCC' LED will extinguish. You should see the following on the screen:

```
+-----------------------------+
|      PROGRAMMING COMPLETED   |
|                             |
|        VERIFYING EPROM       |
|                             |
|     VERIFICATION COMPLETED   |
|                             |
|    VERIFIED: $0000 -> $1FFF  |
|                             |
|      PERFORMING CRC CHECK    |
|                             |
|         CRC CKSM = $0000     |
|                             |
| HIT ANY KEY TO CONTINUE!     |
+-----------------------------+
```

Once the verification of the device is completed the VCC LED will extinguish and a CRC checksum of the buffer will take place. This post programming CRC is provided to assist you in marking the device with the CRC after it is programmed. Pressing any key will return you to the OPERATIONS MENU.

### N O T E

The post-program VERIFY and CRC checks are only performed over the range of the device that was actually programmed. This is in contrast to the normal OPERATIONS MENU selections which normally perform their respective operations over the entire range of the device.

The defaults for two address range prompts are dynamic. Initially they will be set to the full range of the device. If you enter specific 'START' or 'END' addresses they will then become the defaults.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
* The WINDRUSH UNIVERSAL EPROM PROGRAMMER has been designed for use in *
* development system environments where it will be used in intermittent duty *
* service. It is NOT intended for production programming of EPROMS. Ensure *
* that the programmer is allowed to cool for at least two minutes between *
* programming operations otherwise damage to the +12/21/25 volt switching *
* regulator and its associated circuitry may result.                      *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

## 7.0.8 RETURN TO EPROM SELECT MENU

Selecting this operation will return you to the EPROM SELECT (TYPES) MENU:

```
+-------------------------------+
|  SINGLE VOLTAGE DEVICES ONLY! |
|                               |
|        EPROM TYPES MENU       |
|                               |
|     0 = 2758   (INTEL)  1K    |
|     1 = 2716   (INTEL)  2K    |
|     2 = 2532   (TEXAS)  4K    |
|     3 = 2732   (INTEL)  4K    |
|     4 = 2732A  (INTEL)  4K    |
|     5 = 68764  (M'OLA)  8K    |
|     6 = 2564   (TEXAS)  8K    |
| ->  7 = 2764   (INTEL)  8K    |
|     8 = 2764A  (INTEL)  8K    |
|     9 = 27128  (INTEL) 16K    |
|     A = 27128A (INTEL) 16K    |
|     B = 27256  (INTEL) 32K    |
|                               |
|  SELECT TYPE BY NUMBER:       |
+-------------------------------+
```

This menu was described in detail in section 6.0

### 7.0.9 MONITOR

This operation exits the EPROM programmer and passes control of the system to your system monitor. This operation passes control via the 'MONIT' vector in FLEX. The 'MONIT' vector (at $D3F3) should point to the warm start entry point of your sytstem monitor.

Once you have entered your system monitor be very careful not to corrupt any of the buffer code between $0000 and $7FFF or the programmer code between $8000 and $A1FF or any part of the program variable storage between $A200 and $A2FF.

Whenever you exit the software with this option the 'DP' and 'CCR' will be restored to the same state they were on entry. A message will be posted to tell you how to re-enter the software thus:

```
+-----------------------------------+
|                                   |
| RE-ENTER UPROM II SOFTWARE AT: $8003 |
|                                   |
+-----------------------------------+
```

You can return to the programmer software by executing a JUMP to $8000 (cold start)or by executing a JUMP to $8003 (warm start). Cold starting the software will give the same results as calling the software off disk. Warm starting the software will take you straight back into the MAIN (OPERATIONS) MENU.

NOTE: This facility is omitted from the OS-9 version for obvious reasons. The MDOS/XDOS cold start entry point is $3000 and its corresponding warm start entry point is $3003. The SSB DOS entry points are the same as FLEX.

### 7.0.A ALL DONE ... RETURN TO DOS

This should be self explanatory!

NOTE: Whenever you exit the software with this option the 'DP' and 'CCR' will be restored to the same state they were on entry.

## 7.0.F FIND A HEX BYTE STRING

This operation is provided to assist you in editing the code within the buffer. It is most often used in conjunction with the DUMP and EXAMINE and CHANGE operations. For example suppose you wished to locate the hex byte string '7E CD 03' in an EPROM. If you attempted to locate this string visually using the formatted DUMP you would probably go blind before locating it unless you had a fairly good idea of where it was. Invoking this operation results in the following display:

```
+-------------------------------+
|     2764   $0000 - $1FFF      |
|     ==================        |
|                               |
|     FIND HEX BYTE STRING      |
|                               |
| BUFFER RANGE = $0000 -> $7FFF |
|                               |
| ENTER START ADDRESS:  $0000   | <- default is $0000
| ENTER  END  ADDRESS:  $7FFF   | <- default is $7FFF
|                               |
| BYTES?                        |
+-------------------------------+
```

The defaults in this instance are the entire range of the buffer. Once the search limits have been established you will then be prompted 'BYTES?'. You should now enter the hex byte string you wish to search for. Up to 16 bytes can be specified. Entry of the 16th byte or any non-hex character, like <SPACE> or <RETURN> will initiate the search. If you enter a non-hex character in the right nibble of a byte the left nibble will be ignored.

Once the search has been initiated one of two events will take place. The first is that the prompt 'BYTES?' appears again. This means that the specified string could not be found. You may then enter another hex byte string and initiate another search over the previous limits. Alternatively you can hit any non-hex key and you will return to the OPERATIONS MENU.

If a match is found a dump very similar to the HEX/ASCII dump will take place. We emphasize similar because the hex address in the left column is the address of the first byte in the string and not the first byte of the DUMP which is adjacent to it. This is because the six bytes that preceed the string that has been located are displayed before the string itself. This facility is provided to illuminate the context in which a particular string was located.

An example is in order at this point. Use the FILL operation to fill the entire buffer with $00, then use the EXAMINE/CHANGE operation to insert $01 at location $1000, $02 at location $1001 and $03 at location $1002.

Next use FIND to initiate a byte search over the entire range of the buffer as in the sample display above and then type the following:

```
BYTES? 01 02 03 <RETURN>

1000 00 00 00 00 00 00 01 02 03 00 00 00 00 00 00 00   ................

BYTES?
```

You will get the display above.

## 7.0.F FIND A HEX BYTE STRING (continued)

These are the six bytes that preceed the byte string searched for.

```
1000 00 00 00 00 00 00 01 02 03 00 00 00 00 00 00 00   ................
 |                     |
 +-- this    address --+
     corresponds  to
     this byte
```

Now try typing:

BYTES? 00 <RETURN>

And you will get something like this:

```
0000 ?? ?? ?? ?? ?? ?? 00 00 00 00 00 00 00 00 00 00   ??????........
0001 ?? ?? ?? ?? ?? 00 00 00 00 00 00 00 00 00 00 00   ?????.........
0002 ?? ?? ?? ?? 00 00 00 00 00 00 00 00 00 00 00 00   ????..........
0003 ?? ?? ?? 00 00 00 00 00 00 00 00 00 00 00 00 00   ???...........
0004 ?? ?? 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ??............
0005 ?? 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ?.............
0006 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
0007 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
0008 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
0009 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
000A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
000B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ..............
```

The display will just keep dumping data indefinately!

This can happen when you try to locate a byte string that occurs very often in a program. For example if you tried to locate all occurances of the branch instruction ($20) in a program you could end up locating far more than you bargained for.

TO STOP A DUMP THAT RUNS ON FOREVER SIMPLY HIT THE <BREAK> KEY

You should have noticed that the above display shows several bytes as '??'. These are bytes that fall outside of the buffer area and therefore contain meaningless data. The same thing will happen at the end of the buffer if a string is located from $7FF9 to $7FFF.

There are three possible errors that can be made during data entry for this operation:

1. NOT A HEX NUMBER (see section 5.1, paragraph 2)

2. BUFFER LIMITS EXCEEDED (see section 5.1, paragraphs 5 and 6)

3. ILLEGAL ADDRESSES (see section 5.1, paragraph 3)

## 7.0./ EXECUTE A FLEX COMMAND

Use of this command will allow you to access any FLEX utility that runs in the FLEX TCA between $C100 and $C7FF. If you use this facility to gain access to a utility that uses memory outside of the TCA you run the risk of crashing the entire system. The 'COPY' utility is a good example of the latter.

The slash character (/) has been chosen purely for reasons of ergonomics. This is the same character we use in MACE, XMACE, ASM05, ED, and PL/9 for accessing FLEX.

This facility has been provided primarily for five purposes:

1. To 'GET' the printer drivers for the 'DUMP' operation should you have neglected to do so before starting up the programmer software.

2. To do a 'CAT' or 'DIR' of a disk to see what is on it.

3. To use the 'MAP' utility provided with this package to define the memory limits of a disk resident binary file.

4. To use the 'LOAD' utility provided with this package to load a binary file resident on disk into the eprom programmer buffer area.

5. To use the FLEX 'SAVE' utility to save the contents of the eprom programmer buffer out to disk.

If you decide to use it for any other purpose you do so at your own risk! Typing a '/' will yield the following display:

```
+-------------------------------+
|     EXECUTE A FLEX COMMAND    |
|                               |
| (+++)[FLEX utility command]   |
|                               |
+-------------------------------+
```

You may now type in a command just as though you were in FLEX. When the FLEX utility is completed you will be prompted to 'HIT ANY KEY'.

NOTE 1: If you have the TTYSET pause facility enabled the output will cease whenever the 'DP' lines have been reached. Hitting <ESCAPE> will resume output; hitting <RETURN> will terminate the command. You can also stop and start the screen output in the ususal manner by using the <ESCAPE> key.

NOTE 2: The traditional FLEX '+++' prompt is enclosed within '(+++)' to distinguish the result of using this this command from the result of using the 'A' command!

NOTE 3: This facility is not available in the SSB DOS and MDOS versions. A similar facility is available in the OS-9 version. The OS-9 version also has two additional commands 'R' and 'W'. See the OS-9 addendum for further details.

## 8.0 THE 'MAP' UTILITY

This utility is used to produce a memory image map of a binary file that shows you where it would fit in memory. It is consistent with the GET memory resident command, that is a text file MAY APPEAR to have binary records in it and a GET would indeed load those as though they were binary records!

The syntax for invoking MAP is:

```
MAP,<fname>
```

where <fname> is a filename which defaults to a .BIN type on the assigned WORK drive.

It only makes sense to MAP a file containing binary records, though using it on a text file will show where GET would load it!

MAP will print out each contiguous block of the <fname> and if no records are found will report the fact. In addition the TRANSFER address, or lack of it, will be printed. Two examples follow:

```
+++MAP,MONITOR
F800 - F816
F820 - FEEC
FFE0 - FFED
FFF2 - FFFF

File has no TRANSFER ADDRESS
```

The first example is that of a monitor. As such it is just straight binary and does not include a transfer address.

```
+++MAP,0.EXEC.CMD
C100 - C1E8

TRANSFER ADDRESS is at $C100
```

The second example is of the EXEC command. This comprises a single contiguous block of memory $E9 bytes long starting at $C100 with an execution address at $C100.

## 9.0 THE 'LOAD' UTILITY

The LOAD utility allows a program or binary file to be loaded into memory at an absolute address, and informs what the new start address is if a program is loaded.

The syntax of the LOAD utility is:

```
LOAD,<address>,<fname>
```

where <address> is the base address where the <fname> will be loaded in memory,

<fname> is the filename of the command or binary file to be loaded.

The defaults for <fname> are a .CMD file on the assigned system drive.

LOAD has two main uses: loading binary files into RAM memory so that an EPROM may be programmed, loading a command/utility which has been written in Position Independent Code (PIC) in readiness for it's execution at the new address. An example of each use follows:

```
+++LOAD,0000,NEW_MON.BIN.1
```

This example will load what would appear to be a new version of a system monitor into memory starting at $4000. This could be a prerequisite of an EPROM programmer which does not itself contain a binary file offset loader. The fact that there is no transfer address will be indicated.

```
+++LOAD,1000,DEBUG
```

This example will load the DEBUG utility into memory at a base address of $1000 and the transfer address, in this case $1000, will be indicated. Execution at this address will perform a cold start of the DEBUG package.

Two points should be borne in mind about LOAD. Firstly it DOES NOT ALTER ANY OF THE CODE THAT IS LOADED, it merely positions it in memory at a different address. For this reason, any command/utility which is expected to be relocated and execute at the new address ...

```
* * * * * * * * * * * * * * * *
*                               *
*  MUST BE WRITTEN IN PIC       *
*                               *
* * * * * * * * * * * * * * * *
```

## 9.0 THE 'LOAD' UTILITY (continued)

The second point is that LOAD has no way of knowing the structure of all the different files that it may have to load, it therefore places the first binary record loaded at the base address specified. All further records will load into memory with the same offset applied. This will only affect those people who write programs back-to-front! The following example will clarify this point:

```
+++MAP ODD_BALL.BIN.1
C600 - C6E7
C100 - C107
C118 - C4C9

TRANSFER ADDRESS is at $C3DF
```

Given the above program map, consider the following invocation of LOAD:

```
+++LOAD,0,ODD_BALL.BIN.1

C600 - C6E7     -->     0000 - 00E7
C100 - C107     -->     FB00 - FB07
C118 - C4C9     -->     FB18 - FEC9

TRANSFER ADDRESS is at $FDDF
```

This is obviously a mistake as the code will have attempted to overlay the monitor. The only reason for this quirk is that the code is mapped in a sort of backward way. To overcome this it should be noted that the lowest address from MAP was $C100 and further that the first address encountered was $C600. The following formula will then hold true:

LOAD ADDRESS = REQUIRED CODE BASE + FIRST ADDRESS - LOWEST ADDRESS

LOAD ADDRESS = 0 + $C600 - $C100 i.e. $0500

So using the following invocation:

```
+++LOAD,500,ODD_BALL.BIN.1
```

The addresses actually loaded into will be:

```
C600 - C6E7     -->     0500 - 05E7
C100 - C107     -->     0000 - 0007
C118 - C4C9     -->     0018 - 03C9

TRANSFER ADDRESS is at $02DF
```

Our appologises for this detailed example, but it is provided in order to prevent a lot of wasted time if anyone uses LOAD on the type of file that can put the actual code into an illegal area of memory - after all stamping over FLEX would be suicidal!

## 10.0 EPROM READ AND PROGRAMMING CHARACTERISTICS

This section of the manual is provided to assist those of you who have an EPROM with a part number which does not match any of the numbers given in the device selection table. If you have a data sheet for the device you are trying to read or program the information in this section should assist you in determining which device matches the characteristics of the device you are using.

Several pins on the EPROM (ZIF) socket are not altered from one device type to the next. The following table details the pins that have a fixed signal assignment and the pins whose signal assignment varies from one device type to the next.

| 24 PIN DEVICE | 28 PIN DEVICE | SIGNAL DESCRIPTION |
|---|---|---|
| - | 1 | VARIES |
| - | 2 | VARIES |
| 1 | 3 | A7 |
| 2 | 4 | A6 |
| 3 | 5 | A5 |
| 4 | 6 | A4 |
| 5 | 7 | A3 |
| 6 | 8 | A2 |
| 7 | 9 | A1 |
| 8 | 10 | A0 |
| 9 | 11 | D0 |
| 10 | 12 | D1 |
| 11 | 13 | D2 |
| 12 | 14 | GROUND (COMMON) |
| 13 | 15 | D3 |
| 14 | 16 | D4 |
| 15 | 17 | D5 |
| 16 | 18 | D6 |
| 17 | 19 | D7 |
| 18 | 20 | VARIES |
| 19 | 21 | A10 |
| 20 | 22 | VARIES |
| 21 | 23 | VARIES |
| 22 | 24 | A9 |
| 23 | 25 | A8 |
| 24 | 26 | VCC ON 24 PIN ... VARIES ON 28 PIN. |
| -- | 27 | VARIES |
| -- | 28 | VCC ON ALL 28 PIN |

The following sections detail the signal assignments for the pins that vary from one device to the next. The signal levels for the READ and PROGRAM cycles are also given as is a short description of the signal level changes, if any, that take place during the READ and PROGRAM cycles.

### P L E A S E N O T E

Each and every device included in the EPROM selection table has been extensively tested with REAL devices. We did not design this programmer or its software from data sheets alone! This philosophy is the only reason we have not included the AMD 27512 with this release ... we have a preliminary data sheet but cannot get a device to test.

## 10.1 2758/2508

This is a 1K x 8 device. The 2758 is the INTEL part number and 2508 is the TEXAS INSTRUMENTS part number. Neither of these devices is very popular and most manufacturers do not recommend that you design new equipment using them. Note that the TMS2708 is NOT supported as it is a TRI-VOLT device.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 18 | /PD ... /PGM | 0 VOLTS | 0-5-0 (50MS) |
| 20 | /CHIP SELECT (/CS) | 0 VOLTS | 5 VOLTS |
| 21 | VPP (25 VOLTS) | 5 VOLTS | +25 VOLTS |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting the static levels defined above, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond positive pulse to 5 volts to pin 18. At the end of the 50 millisecond pulse pin 18 returns to 0 volts and the process is repeated for the next location to be programmed.

## 10.2 2716/2516

This is a 2K x 8 device. The 2716 is the INTEL part number and 2516 is the TEXAS INSTRUMENTS part number. Both of these devices are second sourced by a variety of other manufacturers. Note that the TMS2716 is NOT supported as it is a TRI-VOLT device.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 18 | /PD ... /PGM | 0 VOLTS | 0-5-0 (50MS) |
| 20 | /CHIP SELECT (/CS) | 0 VOLTS | 5 VOLTS |
| 21 | VPP (25 VOLTS) | 5 VOLTS | +25 VOLTS |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting the static levels defined above, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond positive pulse to 5 volts to pin 18. At the end of the 50 millisecond pulse pin 18 returns to 0 volts and the process is repeated for the next location to be programmed.

## 10.3 2532

This is a 4K x 8 device manufactured by TEXAS INSTRUMENTS and second sourced by a variety of other manufacturers.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 18 | A11 | A11 | A11 |
| 20 | /CS ... /PGM | 0 VOLTS | 5-0-5 (50MS) |
| 21 | VPP (25 VOLTS) | 5 VOLTS | +25 VOLTS |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting the static levels defined above, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 20. At the end of the 50 millisecond pulse pin 20 returns to 5 volts and the process is repeated for the next location to be programmed.

## 10.4 2732 AND 2732A

This is a 4K x 8 device manufactured by INTEL and second sourced by a variety of other manufacturers. The only difference between these two devices is the level of the programming voltage VPP.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 18 | /CE ... /PGM | 0 VOLTS | 5-0-5 (50MS) |
| 20 | /OE ... VPP | 0 VOLTS | VPP (see note) |
| 21 | A11 | A11 | A11 |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting the static levels defined above, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 20. At the end of the 50 millisecond pulse pin 20 returns to 5 volts and the process is repeated for the next location to be programmed.

NOTE: The 2732 requires that VPP be set to +25 volts. The 2732A requires that VPP be set to +21 volts. If you program the 2732A as a 2732 it probably will not live through the experience.

## 10.5 MCM68764 AND MCM68766

This is an 8K x 8 device manufactured by MOTOROLA and is second sourced by by THOMPSON EFCIS. This device is pin compatible with the MOTORLA 8K x 8 mask ROMS and is the only 8K x 8 EPROM manufactured in a 24 pin package. This device is generally not found in production equipment but is often found in pre-production prototypes and engineering prototypes where the design will ultimately use a mask ROM. The fact that this device is pin compatible with the BASIC ROMS in the Tandy 'COLOR COMPUTER' and the DRAGON-32 is one of the major reasons we included it in the design of this programmer.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 18 | A11 | A11 | A11 |
| 20 | /E ... VPP | 0 VOLTS | 5-25-5 (2 MS) |
| 21 | A12 | A12 | A12 |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting pin 20 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 2 millisecond positive pulse to 25 volts to pin 20. At the end of the 2 millisecond pulse pin 20 returns to 5 volts and the process is repeated for the next location to be programmed. After each location to be programmed has had a 2 millisecond programming pulse applied to it the address counter is reset to the first location and the process is repeated for 15 passes. This allows the device to 'accumulate' 30 milliseconds of programming time per byte cell. The reason for the multi-pass approach is to reduce the heat generated in the cells during the programing cycle. Applying a single 30 millisecond pulse would destroy the device.

## 10.6 2564

This is an 8K x 8 device manufactured by TEXAS INSTRUMENTS and second sourced by a variety of other manufacturers.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 1 | VPP | 5 VOLTS | 25 VOLTS |
| 2 | /CS1 | 0 VOLTS | 0 VOLTS |
| 20 | A11 | A11 | A11 |
| 22 | /PD ... /PGM | 0 VOLTS | 5-0-5 (50MS) |
| 23 | A12 | A12 | A12 |
| 26 | VCC | 5 VOLTS | 5 VOLTS |
| 27 | /CS2 | 0 VOLTS | 0 VOLTS |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The PROGRAM cycle consists of setting the static levels defined above, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 22. At the end of the 50 millisecond pulse pin 22 returns to 5 volts and the process is repeated for the next location to be programmed.

## 10.7 2764 AND 2764A

This is an 8K x 8 device manufactured by INTEL and second sourced by a variety of other manufacturers.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 1 | VPP | 5 VOLTS | see note |
| 2 | A12 | A12 | A12 |
| 20 | /CE | 0 VOLTS | 0 VOLTS |
| 22 | /OE | 0 VOLTS | see text |
| 23 | A11 | A11 | A11 |
| 26 | NOT CONNECTED | 5 VOLTS | 5 VOLTS |
| 27 | /PGM | 5 VOLTS | 5-0-5 see text |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The DUMB PROGRAM cycle consists of setting the static levels defined above, setting VPP as defined by the note below, setting pin 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 27. At the end of the 50 millisecond pulse pin 27 returns to 5 volts and the process is repeated for the next location to be programmed.

The INTELLIGENT PROGRAM cycle consists of setting the static levels defined above, setting VPP as defined by the note below, initially setting pin 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 1 millisecond negative pulse to 0 volts to pin 27. At the end of the 1 millisecond pulse pin 27 returns to 5 volts. Pin 22 is then set to 0 volts which allows the device to be 'read' by the software. Once the software has read the data pin 22 is returned to 5 volts. The next action taken will depend on whether the data 'read' matches the data we are trying to program. Refer to section 6.4 for details of the 'inteligent programming algorithm'.

NOTE: The 2764 requires that VPP be set to +21 volts. The 2764A requires that VPP be set to +12.5 volts. If you program the 2764A as a 2764 it probably will not live through the experience.

The programming algorithm for the 2764 is identical to the algorithm for the 27128 described in section 6.4. The programming algorithm for the 2764A is identical to the algorithm for the 27256 described in section 6.5.

If intelligent programming is being used VCC (on pin 28) will be raised to 6 VOLTS during the programming cycle.

## 10.8 27128 AND 27128A

This is a 16K x 8 device manufactured by INTEL and second sourced by a variety of other manufacturers.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 1 | VPP | 5 VOLTS | see note |
| 2 | A12 | A12 | A12 |
| 20 | /CE | 0 VOLTS | 0 VOLTS |
| 22 | /OE | 0 VOLTS | see text |
| 23 | A11 | A11 | A11 |
| 26 | A13 | A13 | A13 |
| 27 | /PGM | 5 VOLTS | 5-0-5 see text |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The DUMB PROGRAM cycle consists of setting the static levels defined above, setting VPP as defined by the note below, setting pin 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 27. At the end of the 50 millisecond pulse pin 27 returns to 5 volts and the process is repeated for the next location to be programmed. The 'DUMB' option is provided for compatibility with possible future devices.

The INTELLIGENT PROGRAM cycle consists of setting the static levels defined above, setting VPP as defined by the note below, initially setting pin 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 1 millisecond negative pulse to 0 volts to pin 27. At the end of the 1 millisecond pulse pin 27 returns to 5 volts. Pin 22 is then set to 0 volts which allows the device to be 'read' by the software. Once the software has read the data pin 22 is returned to 5 volts. The next action taken will depend on whether the data 'read' matches the data we are trying to program. Refer to section 6.4 for details of the 'inteligent programming algorithm'.

NOTE: The 27128 requires that VPP be set to +21 volts. The 27128A requires that VPP be set to +12.5 volts. If you program the 27128A as a 27128 it probably will not live through the experience.

The 27128 programming algorithm is described in section 6.4. The programming algorithm for the 27128A is identical to the algorithm for the 27256 described in section 6.5.

If intelligent programming is being used VCC (on pin 28) will be raised to 6 VOLTS during the programming cycle.

## 10.9 27256

This is a 32K x 8 device manufactured by INTEL and second sourced by a variety of other manufacturers.

| PIN | SIGNAL DESCRIPTION | READ LEVEL | PROGRAM LEVEL |
|---|---|---|---|
| 1 | VPP | 5 VOLTS | 12.5 VOLTS |
| 2 | A12 | A12 | A12 |
| 20 | /CE ... /PGM | 0 VOLTS | 5-0-5 see text |
| 22 | /OE | 0 VOLTS | see text |
| 23 | A11 | A11 | A11 |
| 26 | A13 | A13 | A13 |
| 27 | A14 | A14 | A14 |

The READ cycle consists of setting the static levels defined above and then applying an address to the address lines and reading the resulting data.

The DUMB PROGRAM cycle consists of setting the static levels defined above, setting pins 20 and 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 50 millisecond negative pulse to 0 volts to pin 20. At the end of the 50 millisecond pulse pin 20 returns to 5 volts and the process is repeated for the next location to be programmed.

The INTELLIGENT PROGRAM cycle consists of setting the static levels defined above, initially setting pins 20 and 22 to 5 volts, setting the address lines with the desired address, setting the data lines with the desired data and then applying a 1 millisecond negative pulse to 0 volts to pin 20. At the end of the 1 millisecond pulse pin 20 returns to 5 volts. Pin 22 is then set to 0 volts which allows the device to be 'read' by the software. Once the software has read the data pin 22 is returned to 5 volts. The next action taken will depend on whether the data 'read' matches the data we are trying to program. Refer to section 6.4 for details of the 'inteligent programming algorithm'.

## 10.10 FUTURE DEVICES

At the time this product was developed INTEL and ADVANCED MICRO DEVICES where on the verge of releasing preliminary data on the 27512, 64K x 8, EPROMs. Devices were not available for sampling for a further 'two or three months'. Having decided to wait the proverbial two or three months (which in reality was closer to ten months) for a 27256 we have managed to delay the release of this product for nearly a year!

We are not going to get caught in the same trap twice! We have built 'hooks' into the software and hardware design of this product to cater for the preliminary data on the AMD 27512. Unfortunately the PCB design was finalized before we recieved this data sheet. The AMD 27512 has put 'A15' on pin 1 and multiplexed VPP with the output enable on pin 22. This will neccesitate some minor (honest!) hardware modifications to be made to the existing upper ('B') PCB to accomodate this device and a new software package which will enable you to program the 27512 as two blocks of 32K.

As soon as we have a sample of the INTEL and AMD devices we will finalize a new software package and a field modification notice for all existing customers. The upgrade package will cost $65.00 which will include AIR MAIL postage. Contact our office for availability in the first quarter of 1985.

We are also looking into supporting the 'inteligent identifier' mode of operation (see Intel data sheets) if manufacturers other than Intel start using it. After all if only Intel are supporting it who needs a bunch of extra hardware and software when the 'eye' and the 'brain' do the job just as well?

At the moment we have the ultimate 'intelligent identifier' ... the user. If the user can read the part number and manufacturers code that is printed on the device he has identified it!

### AN OFFER YOU CAN'T REFUSE

The first person out there who sends us a sample of the 27512 so that we can test our in-house software drivers and hardware design will get a free programmer pod and the software to drive it in return.

Contact us before taking us up on this offer just to make sure that you are the first!

## 11.0 'STANDARD' HARDWARE CONFIGURATIONS

**W A R N I N G**

The information presented in this section is for guidance only. You should use the information supplied by the manufacturer of your system in preference to any information you see here particularly if this information conflicts with your hardware documentation.

## 11.1 INTERFACING WITH A PIA

If you are considering connecting the EPROM programmer to an interface board not supplied by us ensure that the PIA addresses stack up as follows:

BASE ADDRESS + 0: 'A' SIDE DATA PORT
BASE ADDRESS + 1: 'A' SIDE CONTROL PORT
BASE ADDRESS + 2: 'B' SIDE DATA PORT
BASE ADDRESS + 3: 'B' SIDE CONTROL PORT

**W A R N I N G**

Connecting the EPROM programmer to any interface board not supplied by Windrush Micro Systems Limited will void the warranty.

## 11.2 S-30 BUS PORT ADDRESSES

S-30 BUS WITH FOUR ADDRESSES PER PORT

| BASE ADDRESS | PORT 0 | PORT 1 | PORT 2 | PORT 3 | PORT 4 | PORT 5 | PORT 6 | PORT 7 |
|---|---|---|---|---|---|---|---|---|
| $8000 | $8000 | $8004 | $8008 | $800C | $8010 | $8014 | $8018 | $801C |
| $E000 | $E000 | $E004 | $E008 | $E00C | $E010 | $E014 | $E018 | $E01C |
| $F7E0 | $F7E0 | $F7E4 | $F7E8 | $F7EC | $F7F0 | $F7F4 | $F7F8 | $F7FC |

S-30 BUS WITH SIXTEEN ADDRESSES PER PORT

| BASE ADDRESS | PORT 0 | PORT 1 | PORT 2 | PORT 3 | PORT 4 | PORT 5 | PORT 6 | PORT 7 |
|---|---|---|---|---|---|---|---|---|
| $8000 | $8000 | $8010 | $8020 | $8030 | $8040 | $8050 | $8060 | $8070 |
| $E000 | $E000 | $E010 | $E020 | $E030 | $E040 | $E050 | $E060 | $E070 |
| $F7E0 | $F7E0 | $F790 | $F7A0 | $F7B0 | $F7C0 | $F7D0 | $F7E0 | $F7F0 |

## 12.0 ADVANCED PROGRAMMERS SECTION

This section of the manual is for those of you who may wish to alter some of the characteristics of the EPROM programmer software but do not wish to buy the source package. This section applies to the FLEX, SSB and MDOS versions only.

This information is for guidance only. We will not provide any assistance in the customization of this software to suit any particular purpose.

There will be a file on the FLEX, SSB, and MDOS disks called 'UPRII-EQ.ASM', 'UPRIIQ.ASM' and 'UPRIIQ.SA' respectively. The purpose of this 'equates' file is to provide the user with an assembly language source file which he can modify, assemble and append to the main EPROM programmer software to create a modified version of the software.

The user should first modify the 'equates' file and then assemble it out to disk. The user should then append the modified binary file to the original code. The easiset way to 'append' the files is to use your DOS resident 'GET' or 'LOAD' command to load the original software, then load in the new binary file created by assembling the 'equates' file, and finally using your DOS 'SAVE' or 'ROLLOUT' command to save the new memory resident object code on to disk.

To do the latter you must know the boundaries of the existing software. The range of the software is always displayed when the software first starts up so this is one way of determining the load area. You can also use your DOS 'FIND' or 'MAP' utilities to do the same thing. When you know the boundaries of the software you simply save it out to disk specifying the original transfer address. The transfer address will be the first byte of the program load range.

An example is in order. Suppose you are using FLEX and you have just modified the file called 'UPRII-EQ.ASM' and have assembled it to a file called 'UPRII-EQ.BIN' and want to create a new programmer file. There are two approaches which can be used:

### APPROACH 1

Type the following in on the FLEX command line:

```
+++APPEND,1.UPRII.CMD,1.UPRII-EQ.BIN,1.MY-UPRII.CMD<RETURN>
```

FLEX will then create a new binary file called 'MY-UPRII.CMD' which you can treat just like any other command file.

### APPROACH 2

Type the following in on the FLEX command line:

```
+++GET,1.UPRII.CMD
+++GET,1.UPRII-EQ.BIN
+++SAVE,1.MY-UPRII.CMD,8000,A1FF,8000<RETURN>
```

NOTE: The address $A1FF is an example only. The exact address will vary.

Again FLEX will create a new binary file called 'MY-UPRII.CMD' which you can treat just like any other command file.

The SSB DOS and MDOS disk operating systems operate in a similar manner. Consult the DOS manuals for detailed information.

## 12.1 USER CONFIGURATION TABLE

PAGE 1: NOVEMBER 10 1983

```
 1 ************************************************
 2 *     UNIVERSAL EPROM PROGRAMMER II SOFTWARE   *
 3 ************************************************
 4
 5 ************************************************
 6 * THIS  SOFTWARE  IS COPYRIGHT (C) 1983 AND 1984 *
 7 * AND MAY NOT BE REPRODUCED IN WHOLE OR IN  PART *
 8 * BY  ANY  MEANS  WITHOUT  THE  EXPRESS  WRITTEN *
 9 * PERMISSION OF WINDRUSH MICRO SYSTEMS LIMITED.  *
10 ************************************************
11
12 ************************************************
13 * THIS SOFTWARE MAY BE MODIFIED BY THE  ORIGINAL *
14 * PURCHASER  FOR  THE  SOLE  USE OF THE ORIGINAL *
15 * PURCHASER BUT SUCH MODIFICATIONS  MAY  NOT  BE *
16 * PASSED  ON  TO  THIRD  PARTIES  BY  ANY  MEANS *
17 * WITHOUT  THE  EXPRESS  WRITTEN  PERMISSION  OF *
18 * WINDRUSH MICRO SYSTEMS LIMITED.                *
19 ************************************************
20
21 ************************************************
22 * WINDRUSH  MICRO  SYSTEMS  LIMITED,  WORSTEAD *
23 * LABORATORIES  NORTH WALSHAM, NORFOLK, ENGLAND. *
24 * TEL: (0692) 405189                             *
25 ************************************************
26
27 ************************************************
28 *                                              *
29 *        THIS VERSION IS FOR 6809 'FLEX'       *
30 *                                              *
31 ************************************************
32
33 ************************************************
34 * THIS  SOFTWARE  COMPRISES EIGHT SECTIONS WHICH *
35 * MUST BE ASSEMBLED CONCURRENTLY:                *
36 *                                              *
37 * 1. UPRII-QG.ASM  I/O EQUATES FILE            *
38 * 2. UPRII-1G.ASM  LOW-LEVEL CONSOLE I/O       *
39 * 3. UPRII-2G.ASM  HARDWARE DRIVERS            *
40 * 4. UPRII-3G.ASM  MENUS                       *
41 * 5. UPRII-4G.ASM  FILL, MOVE, E&C, DUMP, CRC  *
42 * 6. UPRII-5G.ASM  COPY, VERIFY, PROGRAM       *
43 * 7. UPRII-6G.ASM  TEXT MESSAGES               *
44 * 8. UPRII-7G.ASM  FLEX INTERFACE              *
45 ************************************************
46
47 ************************************************
48 * THE   BUFFER   IS  DEFINED  BY  'BUFF.LO'  AND *
49 * 'BUFF.HI' WHICH MAY BE  ALTERED  TO  SUIT  THE *
50 * MEMORY   MAP   OF   THE  USERS  SYSTEM.  THESE *
51 * DEFINITIONS ARE MADE VIA 'FDB' STATEMENTS  AND *
52 * ARE  THEREFORE CAPABLE OF BEING PATCHED AT THE *
53 * BINARY LEVEL.                                  *
54 ************************************************
55
```

12.1 USER CONFIGURATION TABLE (continued)

PAGE 2: NOVEMBER 10 1983

```
                        56 ******************
                        57 * VERSION NUMBER *
                        58 ******************
                        59 *
0058                    60 VER.1     EQU   'X
0058                    61 VER.2     EQU   'X
0058                    62 VER.3     EQU   'X
                        63
                        64
                        65 ***********************************
                        66 * PROM PROGRAMMER MEMORY ORIGINS *
                        67 ***********************************
                        68 *
0000                    69 LO.BUFF   EQU   $0000      START OF BUFFER
7FFF                    70 HI.BUFF   EQU   $7FFF      END OF BUFFER
                        71 *
8000                    72 PRMPRG    EQU   $8000      PROGRAM START ADDRESS
                        73 *
E090                    74 PIA.PORT  EQU   $E090      DEFAULT ADDRESS OF PIA
                        75
                        76
                        77 ***************
                        78 * I/O EQUATES *
                        79 ***************
                        80 *
CCC0                    81 XPINIT    EQU   $CCC0      INITIALIZE PRINTER
CCE4                    82 XPOUTCH   EQU   $CCE4      HARD COPY OUTPUT
                        83
CD15                    84 XINPUT    EQU   $CD15      GET CHAR AND ECHO IT
CD12                    85 XOUTPUT   EQU   $CD12      PUT CHAR ON SCREEN
CD4E                    86 XSTAT     EQU   $CD4E      NON-ZERO = KEY PENDING
                        87
                        88
                        89 ****************
                        90 * FLEX EQUATES *
                        91 ****************
                        92 *
CD03                    93 XDOS      EQU   $CD03      DOS WARM ENTRY POINT
D3F3                    94 XMONITR   EQU   $D3F3      FLEX 'MONIT' VECTOR
CC2B                    95 XMEMEND   EQU   $CC2B      FLEX 'MEMEND'
                        96
                        97
                        98 *********************************
                        99 * TERMINAL CONTROL CHARACTERS *
                       100 *********************************
                       101 *
0000                   102 NL        EQU   $00        USED FOR EOT
0004                   103 ET        EQU   $04        END OF TRANSMISSION
0007                   104 BL        EQU   $07        BELL
0008                   105 BS        EQU   $08        BACKSPACE
000A                   106 LF        EQU   $0A        LINE-FEED
000C                   107 HC        EQU   $0C        FORM-FEED (HOME AND CLR)
000D                   108 CR        EQU   $0D        CARRIAGE RETURN LINE-FEED
001B                   109 ES        EQU   $1B        ESCAPE
0020                   110 SP        EQU   $20        SPACE
```

12.1 USER CONFIGURATION TABLE (continued)

PAGE 3: NOVEMBER 10 1983

```
                                111 ********************
                                112 * START OF PROGRAM *
                                113 ********************
                                114 *
0000                            115           SETDP $0000     PROGRAM WILL SET THE 'DP'
                                116
8000                            117           ORG   PRMPRG    PROGRAM START ADDRESS
                                118
8000   16 0075    (8078)        119 ENTRE    >LBRA  COLDS1    COLD START ENTRY POINT
8003   16 0074    (807A)        120 ENTRE2   >LBRA  WARMS1    WARM START ENTRY POINT
                                121
                                122
                                123 ****************************
                                124 * VERSION NUMBER IN ASCII *
                                125 ****************************
                                126 *
8006 563A                       127           FCC   /V:/
8008 58                         128           FCB   VER.1      :
8009 2E                         129           FCC   /./
800A 58                         130           FCB   VER.2      : VERSION NUMBER IN ASCII
800B 58                         131           FCB   VER.3      :
                                132
                                133
                                134 **************
                                135 * JUMP TABLE *
                                136 **************
                                137 *
                                138 ************************************************
                                139 * EACH 'JUMP' IS FOLLOWED BY A 'NOP' TO ALLOW *
                                140 * THE  USER TO PATCH THIS TABLE TO POINT AT A *
                                141 * VECTOR TABLE RATHER THAN THE ENTRY POINT OF *
                                142 * THE  ACTUAL  ROUTINE  OR A JUMP TABLE. i.e. *
                                143 * THIS STRUTURE ALLOWS YOU  TO  'JMP  [PROG]' *
                                144 ************************************************
                                145 *
800C   7E CD4E                  146 STAT      JMP   XSTAT      K/B STATUS IN 'CCR'
800F   12                       147           NOP
                                148
8010   7E CD15                  149 INPUT     JMP   XINPUT     GET CHAR FROM THE K/B
8013   12                       150           NOP
                                151
8014   7E CD12                  152 OUTPUT    JMP   XOUTPUT    SEND CHAR TO THE CONSOLE
8017   12                       153           NOP
                                154
8018   7E CCC0                  155 PINIT     JMP   XPINIT     INITIALIZE PRINTER PORT
801B   12                       156           NOP
                                157
801C   7E CCE4                  158 POUT      JMP   XPOUTCH    SEND CHAR TO LINE PRINTER
801F   12                       159           NOP
                                160
8020   7E CD03                  161 SYS.DOS   JMP   XDOS       RETURN TO DOS WARM START
8023   12                       162           NOP
                                163
8024   6E 9F D3F3               164 SYS.MON   JMP   [XMONITR] RETURN TO SYSTEM MONITOR
                                165
```

## 12.1 USER CONFIGURATION TABLE (continued)

PAGE 4: NOVEMBER 10 1983

```
                        166 ****************************************
                        167 * SYSTEM SOFTWARE HARDWARE PATCH TABLE *
                        168 ****************************************
                        169 *
8028 0000               170 BUFF.LO   FDB   LO.BUFF     BASE OF BUFFER
802A 7FFF               171 BUFF.HI   FDB   HI.BUFF     TOP OF BUFFER
                        172 *
802C E090               173 PIAPORT   FDB   PIA.PORT    DEFAULT ADDRESS OF PIA
802E 00                 174 PRMT.FLG  FCB   0           = 0 ... NO PROMPT
                        175 *                          <>0 ... PROMPT FOR PORT
                        176 *
802F 05                 177 NUL.CNT   FCB   5           NULLS AFTER CR-LF, ETC.
                        178 *
8030 FF                 179 STATMODE  FCB   $FF         $F0 = EQ,   $FF = NE
                        180 *                          $00 = CS,   $0F = CC
8031 37                 181 DEF.EPRM  FCB   '7          DEFAULT EPROM (ASCII)
                        182 *
8032 32                 183 NORM.PUL  FCB   50          DUMB PROGRAM PULSE WIDTH
8033 02                 184 SPCL.PUL  FCB   2           68764 PROGRAM PULSE WIDTH
8034 01                 185 INTL.PUL  FCB   1           INITIAL PROGRAM PULSE
                        186 *
8035 0F                 187 PASSES    FCB   15          PASSES FOR MCM68764
                        188 *
8036 0F                 189 LMT.1     FCB   15          INITIAL LIMIT FOR 2764/128
8037 04                 190 OVP.1     FCB   4           MULTIPLIER FOR 2764/128
                        191 *
8038 19                 192 LMT.2     FCB   25          LIMIT FOR 2764A/128A/256
8039 03                 193 OVP.2     FCB   3           MULT FOR 2764A/128A/256
                        194 *
803A 00                 195           FCB   0           RESERVED FOR INTEL 27512
803B 00                 196           FCB   0           RESERVED FOR INTEL 27512
                        197 *
803C 00                 198           FCB   0           RESERVED FOR AMD 27512
803D 00                 199           FCB   0           RESERVED FOR AMD 27512
```

## 12.1 USER CONFIGURATION TABLE (continued)

PAGE 5: NOVEMBER 10 1983

```
803E 0010          201 COLS.S    FDB   16          SCREEN COLUMNS
8040 10            202 ROWS.S    FCB   16          SCREEN ROWS
                   203 *
8041 0010          204 COLS.P    FDB   16          PRINTER COLUMNS
8043 40            205 ROWS.P    FCB   64          PRINTER ROWS
8044 02            206 ROWS.B    FCB   2           PRINTER BLANK ROWS
                   207 *
8045 20            208 LO.ASCII  FCB   $20         LOWEST DISPLAYABLE CHAR
8046 7D            209 HI.ASCII  FCB   $7D         HIGHEST DISPLAYABLE CHAR
8047 7D            210 PR.ASCII  FCB   $7D         HIGHEST PRINTABLE CHAR
                   211 *
8048 0D            212 DEF.CODE  FCB   $0D         ACCEPT DEFAULT VALUES
8049 03            213 ABT.CODE  FCB   $03         ABORT AND RETURN TO MENU
                   214 *
804A 55            215 PREV1     FCB   'U          PREVIOUS #1
804B 5E            216 PREV2     FCB   '^          PREVIOUS #2
804C 2D            217 PREV3     FCB   '-          PREVIOUS #3
                   218 *
804D 20            219 NEXT1     FCB   $20         NEXT #1 (SPACE-BAR)
804E 0A            220 NEXT2     FCB   $0A         NEXT #2 (LINE-FEED)
804F 2B            221 NEXT3     FCB   '+          NEXT #3
                   222
804F               223 END.TBLE  EQU   *-1
```

12.1 USER CONFIGURATION TABLE (continued)

PAGE 6: NOVEMBER 10 1983

```
                        225 ************************************************
                        226 * THE FOLLOWING TABLES MAY  BE  USED  TO  INVOKE *
                        227 * VIDEO  ATTRIBUTES  THAT  MAY BE AVAILABLE WITH *
                        228 * YOUR TERMINAL. THE SOFTWARE HAS BEEN  DESIGNED *
                        229 * TO  PROVIDE HIGHLIGHTED BANNERS WITH TERMINALS *
                        230 * THAT SUPPORT A REVERSE VIDEO ATTRIBUTE.        *
                        231 *                                                *
                        232 * IF YOU INSTALL A CODE SEQUENCE IN EACH OF  THE *
                        233 * THREE  TABLES  THE SOFTWARE WILL AUTOMATICALLY *
                        234 * USE IT IN PREFERENCE TO  THE  'DUMB'  TERMINAL *
                        235 * DRIVERS  THAT DEFAULT IF THE FIRST BYTE IN THE *
                        236 * TABLE IS $04. SOME TERMINALS  DO  NOT  HAVE  A *
                        237 * REVERSE  VIDEO ATTRIBUTE BUT HAVE A REDUCED OR *
                        238 * AN INTENSIFIED ATTRIBUTE. WE'LL  LEAVE  IT  TO *
                        239 * YOU  TO  DECIDE IF THE DISPLAY THAT RESULTS IS *
                        240 * ACCEPTABLE TO YOU. IF IT IS NOT THEN DON'T USE *
                        241 * THE VIDEO ATTRIBUTES.                          *
                        242 ************************************************
                        243
8060                    244           ORG   PRMPRG+$60 LEAVE ROOM FOR TABLE
                        245
8060 04                 246 HOME.CLR  FCB   ET
8061 04                 247           FCB   ET
8062 04                 248           FCB   ET
8063 04                 249           FCB   ET
8064 04                 250           FCB   ET
8065 04                 251           FCB   ET
8066 04                 252           FCB   ET
8067 04                 253           FCB   ET          DON'T ALTER!
                        254 *
8068 04                 255 ATTR.ON   FCB   ET
8069 04                 256           FCB   ET
806A 04                 257           FCB   ET
806B 04                 258           FCB   ET
806C 04                 259           FCB   ET
806D 04                 260           FCB   ET
806E 04                 261           FCB   ET
806F 04                 262           FCB   ET          DON'T ALTER!
                        263 *
8070 04                 264 ATTR.OFF  FCB   ET
8071 04                 265           FCB   ET
8072 04                 266           FCB   ET
8073 04                 267           FCB   ET
8074 04                 268           FCB   ET
8075 04                 269           FCB   ET
8076 04                 270           FCB   ET
8077 04                 271           FCB   ET          DON'T ALTER!
                        272
                        273 ******************
                        274 * ENTRY BRANCHES *
                        275 ******************
                        276 *
8078                    277 COLDS1    EQU   *
807A                    278 WARMS1    EQU   *+2
8000                    279           END   PRMPRG
```

## 12.1 USER CONFIGURATION TABLE (continued)

PAGE 7: NOVEMBER 10 1983

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ABT.CODE | 8049 | ATTR.OFF | 8070 | ATTR.ON | 8068 | BL | 0007 | BS | 0008 |
| BUFF.HI | 802A | BUFF.LO | 8028 | COLDS1 | 8078 | COLS.P | 8041 | COLS.S | 803E |
| CR | 000D | DEF.CODE | 8048 | DEF.EPRM | 8031 | END.TBLE | 804F | ENTRE | 8000 |
| ENTRE2 | 8003 | ES | 001B | ET | 0004 | HC | 000C | HI.ASCII | 8046 |
| HI.BUFF | 7FFF | HOME.CLR | 8060 | INPUT | 8010 | INTL.PUL | 8034 | LF | 000A |
| LMT.1 | 8036 | LMT.2 | 8038 | LO.ASCII | 8045 | LO.BUFF | 0000 | NEXT1 | 804D |
| NEXT2 | 804E | NEXT3 | 804F | NL | 0000 | NORM.PUL | 8032 | NUL.CNT | 802F |
| OUTPUT | 8014 | OVP.1 | 8037 | OVP.2 | 8039 | PASSES | 8035 | PIA.PORT | E090 |
| PIAPORT | 802C | PINIT | 8018 | POUT | 801C | PR.ASCII | 8047 | PREV1 | 804A |
| PREV2 | 804B | PREV3 | 804C | PRMPRG | 8000 | PRMT.FLG | 802E | ROWS.B | 8044 |
| ROWS.P | 8043 | ROWS.S | 8040 | SP | 0020 | SPCL.PUL | 8033 | STAT | 800C |
| STATMODE | 8030 | SYS.DOS | 8020 | SYS.MON | 8024 | VER.1 | 0058 | VER.2 | 0058 |
| VER.3 | 0058 | WARMS1 | 807A | XDOS | CD03 | XINPUT | CD15 | XMEMEND | CC2B |
| XMONITR | D3F3 | XOUTPUT | CD18 | XPINIT | CCC0 | XPOUTCH | CCE4 | XSTAT | CD4E |

## 12.2 GET STATUS OF SYSTEM CONSOLE KEYBOARD

LINE #146. Default is the FLEX 'STAT' routine at $CD4E. This routine is called to obtain the current status of the keyboard. The routine must never lock-up and is expected to return the status of the keyboard in the 'CCR', any other registers may be modified. The 'STATMODE' flag, explained in a moment, must be set according to the condition code register flag used by the users routine to signify that a key is waiting.

## 12.3 KEYBOARD STATUS MODE BYTE

LINE #179. Default is $FF which corresponds to the FLEX 'STAT' routine returning a 'not-equal' condition in the CCR when a key is waiting. The STATMODE flag definitions are as follows:

$F0 = EQUAL (Z=1) when key is waiting.
$FF = NOT EQUAL (Z=0) when key is waiting.
$00 = CARRY SET (C=1) when key is waiting.
$0F = CARRY CLEAR (C=0) when key is waiting.

## 12.4 INPUT A CHARACTER FROM THE SYSTEM CONSOLE KEYBOARD

LINE #149. Default is the FLEX 'GETCHR' routine at $CD15. This routine is called to obtain a key from the keyboard AND automatically echo it to the screen. If your keyboard input routine does not echo the character to the screen simply change the 'JMP' instruction on LINE #149 to a 'JSR' instruction. Since the output character routine JUMP follows the input character JUMP this change will provide the auto-echo facilities required. The routine is expected to lock-up until the user presses a key. The keyboard code is to be returned in the 'A' accumulator, any other registers may be modified.

## 12.5 OUTPUT CHARACTER TO THE SYSTEM CONSOLE

LINE #152. Default is is the FLEX 'OUTCH2' routine at $CD12. This routine is called to output a character to the system console. This routine is called with the character to be output in the 'A' accumulator, any other registers may be modified.

## 12.6 INITIALIZE THE SYSTEM PRINTER

LINE #155. Default is the FLEX 'PINIT' routine at $CCC0. This routine is called to initialize the system printer and/or its associated drivers. Any registers may be modified.

## 12.7 OUTPUT A CHARACTER TO THE SYSTEM PRINTER

LINE #158. Default is the FLEX 'POUT' routine at $CCE4. This routine is called to output a character to the system printer. This routine is called with the character to be output in the 'A' accumulator, any other registers may be modified.

## 12.8 DOS WARM START

LINE #161. Default is the FLEX 'WARMS' entry point at $CD03. This jump is called to return control of the system to the disk operating system. The 'DP' and 'CCR' will be restored to the values they were on entry to the programmer software before execution is passed to this point.

## 12.9 RE-ENTER SYSTEM MONITOR

LINE #164. Default is the FLEX 'MONIT' vector at $D3F3. This jump is called to hand control of the system to the users system monitor. The 'DP' and 'CCR' will be restored to the values they were on entry to the programmer software before execution is passed to this point. Note that an indirect jump (JMP [VECTOR]) in this case.

## 12.10 BUFF.LO/BUFF.HI

LINE #170/171. Default is $0000 and $7FFF respectively. These addresses may be alterered to change the physical position of the buffer in the system memory map. Since the progrmmer code is position independent the user can create his own programmer module by altering these two locations and then moving the software to a new address and saving it out to disk.

## 12.11 PIAPORT/PRMT.FLG

LINE #173/174. This location contains the address of the default pia port address. If the 'PRMT.FLG' is $00 (as supplied) the software will attempt to use the 'PIAPORT' as the default device. If 'PRMT.FLG' is $FF the software will always prompt for the PIA port address when it is cold-started. The address of the default PIA varies with the interface board supplied. The standard defaults are as follows:

```
EXORciser: $EC20
S-30:      $E070 (GIMIX PORT 7)
3U:        $E090
6U:        $E090
```

## 12.12 NUL.CNT

LINE #177. This location contains a decimal count value that represents the number of NULLS ($00) that will be sent to the terminal after any CR-LF, LF, or special code sequence. This is provided as many terminals require null padding if they are used at very high baud rates. The default value is 5. If the software produces a dis-jointed appearance on the screen this is an indication that this value needs to be increased. The upper limit is 256 nulls ($FF).

## 12.13 DEF.EPRM

LINE #181. This location contains the ASCII value of the EPROM in the EPROM TYPES MENU (see section 6.0) that you wish to use as the default device. We supply the software with this set to item '7' ($37) which is a 2764 type device. If you are using the 2716 more often than not changing this location to '1' ($31) whould make the 2716 the default device. The '->' prompt would then point to the 2716 and hitting <RETURN> in response to the device type will do the same thing as typing '1'.

## 12.14 HARDWARE EQUATES

LINE #183 through #199. The comments in the listing should adequately explain what these variables represent. Generally speaking there should not be any need to alter them unless you are using a device which has an intelligent programming algorithm slightly different from the Intel devices. If this is the case we would appreciate it if you could forward us a copy of the manufacturers data sheet. We will then advise you what changes are required.

## 12.15 SCREEN AND PRINTER FORMATTING

LINE #201 THROUGH LINE #210. These variables establish the format used during the HEX/ASCII dump. The values as supplied assume an 80 column x 24 line system console and an 80 column x 66 line system printer. If your system console is different you can re-format the screen dump by altering 'COLS.S' and 'ROWS.S'. If your system printer is different you can re-format the printer dump by altering 'COLS.P', 'ROWS.P' and ROWS.B'. The latter variable determines the number of blank lines ejected at the bottom of each page to ensure that the dump remains within the paper limits.

The range of the displayable and printable ASCII codes is determined by the values at 'LO.ASCII', 'HI.ASCII' and 'PR.ASCII'. The first is common to the screen and printer routines. The second applies to the screen routine only and the latter applies to the printer routine only.

## 12.16 KEYBOARD DEFAULTS

LINE #212/213. These variables determine which keys are going to be used for 'ACCEPT DEFAULTS' and 'BREAK' (ABORT ACTIVITY) respectively. DON'T use the ESCAPE KEY '$1B' for 'ABT.CODE' as FLEX 'GETCHR' does funny things when it sees this code.

## 12.17 EXAMINE AND CHANGE

LINE #215 THROUGH LINE #221. These ASCII variables allow you to alter the keys used by the EXAMINE and CHANGE operation to move forward and backward in the buffer. There are three possible codes for PREVIOUS and three possible codes for NEXT. If you want the software to recognize only one key for each of these operations simply put the same ASCII code in all three positions.

## 12.18 HOME CURSOR AND CLEAR SCREEN

LINE #246 THROUGH LINE #253. If the first position in this table is $04 (EOT) the software will send 24 line-feeds to the system console whenever it wants to clear the screen. If your terminal supports home and clear you should put the code sequence necessary to invoke it in this table.

## 12.19 VIDEO ATTRIBUTES

LINE #255 THROUGH LINE #271. This software package has been designed to produce very bold banners if your terminal supports a reverse-video attribute. In addition to highlighted banners the default answers to prompts are also displayed in reverse video. If your terminal supports reverse video put the code sequence to turn on reverse video in the 'ATTR.ON' table and the code sequence to restore normal video in the 'ATTR.OFF' table. If your terminal supports reduced or intensified video attributes you might try putting in the code sequences just to see if the resulting display is pleasing to you. If it is not don't use the video attributes.

## A.1 INTRODUCTION

This section of the manual is provided to document the differences between the MDOS and XDOS versions of the accompanying software and the FLEX version as described in the body of this document. There is no difference between the MDOS and XDOS versions of this software other than the former is supplied on 8" disk and the latter is supplied on 5" disk.

## A.2 WHAT YOU SHOULD GET

The floppy disk you receive with this package should have the following files on it:

UPRII .CM ... the executable object file
UPRMMQ.SA ... the software tables which may be modified by the user
README.TX ... tells you what is on the disk

## A.3 MAJOR DIFFERENCES

The initial memory range checks performed by the FLEX version are not implemented. The software and buffer reside in a different areas of memory than the FLEX version. The command option '/' (EXECUTE A FLEX COMMAND) is not available. The HEX/ASCII dump routine printer drivers are not implemented but they may be installed by the user by a patch to the software tables (see section 12). The ability to abort a HEX/ASCII dump or a FIND operation that locates too many occurances is not implemented as there is no 'KEYBOARD STATUS' facility in MDOS. If the user can provide a STAT routine these facilities will become available if he patches the software tables (see section 12). The default address of the PIA is $EC20.

## A.4 MEMORY MAP

The MDOS software memory map is as follows:

$0000 - $2FFF USER/MDOS RAM
$3000 - $52FF EPROM PROGRAMMER SOFTWARE
$6000 - $DFFF EPROM PROGRAMMER BUFFER

Thus you require 56K of contiguous RAM memory from $0000 to $DFFF for proper operation. The references to $8XXX in section 12 of the manual should be be adjusted to $5XXX.

## A.5 COLD AND WARM START ENTRY

The MDOS entry points are as follows:

COLD START ... $3000 WARM START ... $3003

## A.6 LOADING BINARY FILES INTO THE PROGRAMMER BUFFER

Disk memory image binary files produced by an assembler, compiler, etc. or RAM saved onto disk using the ROLLOUT command will often be used a the source of code to program into an EPROM.

In order to program this code into an EPROM it must be loaded into the programmer buffer area which resides from $6000 to $DFFF. We have provided a combined binary file 'MAP' utility with a binary file 'OFFSET LOAD' utility. This utility is called 'ULOAD' and is documented in the following section.

Before you invoke the 'ULOAD' command to load the buffer we suggest that you use the UPROM software to pre-load the buffer area with $FF (using the FILL operation) as this will make the program code easier to locate and will also prevent random data in the buffer from being programmed into the EPROM.

### A.6.1 LOAD9H

Like the main programmer software 'UPROM' this is also a menu driven software package. LOAD9H is invoked by typing in 'LOAD9H' followed by a carriage return.

LOAD9H will then display the default buffer area $6000 - $DFFF and will prompt the user for the name of the binary file to be loaded. An incorrect response will return an MDOS error code and return the user to MDOS. If the response is the name of a memory image binary file LOAD9H will 'map' the file, i.e. will inform the user of its STARTING ADDRESS, ENDING ADDRESS and the NUMBER OF BYTES in the file.

Just under the file map will be the location in the UPROM buffer area the file will be loaded if the default values are accepted. If the user wishes to load the file into a different area of the buffer, as when concatonating two 1K disk files into a 2K EPROM, he/she should answer the 'ACCEPT DEFAULT VALUES' prompt with an 'N' (NO). If you answer the prompt with a 'Y' (YES) LOAD9H will proceed to load the file at the addresses indicated under the file map. If no errors occur LOAD9H will then return control to MDOS. If an error is detected the error code will be displayed before returning to MDOS.

If the user answers the prompt 'ACCEPT DEFAULT VALUES' with an 'N' he/she will be prompted for further information as follows:

A FILE OFFSET — A value from $0000 to the end of the file which indicates the position in the file where the loader is to start reading data.

A BUFFER OFFSET — A value from $0000 to the buffer limit ($7FFF) which indicates the position in the buffer that the loader is to start loading the data read from the file.

THE TRANSFER SIZE — A value from $0001 to the file size which indicates how many bytes are to be 'READ' from the file and 'LOADED' into the buffer.

If the values given exceed allowed limits or are non-hex an error message will be displayed and you will be re-prompted for the information.

If the values are acceptable the loader will proceed to load the file as specified. If any errors occur whilst loading the file the error code will be posted before control is returned to MDOS.

A.6.1 LOAD9H (continued)

The 'LOAD9H' utility is supplied to match the standard EPROM programmer buffer area of $6000 - $DFFF. If you have to alter the position of the buffer area for any reason you should also alter the following two locations in the 'LOAD9H' utility:

location $300F contains the lower limit of the buffer ($6000)
location $3011 contains the upper limit of the buffer+1 ($E000)

To patch these locations first use 'LOAD9H' to map itself so you know the last address used. You will require this information in a moment. Next use the MDOS 'LOAD' utility to load the file called 'LOAD9H.CM' into memory. Now enter 'XBUG' and make the necessary changes to the above locations using the memory examine and change facilities. RE-ENTER/BOOT MDOS and use the 'ROLLOUT' command to save the contents of memory from $3000 to the address noted previously (somewhere around $35FF) to disk as 'MYLOAD.CM' or some other name.

## A.7 THE INTERFACE BOARD

The 'EXORFACE' board occupies 4 bytes and may be addressed to any portion of the 64K memory map. The board also supports the 'VUA', 'VXA' and 'PAGE' modes of operation as used in some systems. Consult the schematic diagram of the board for details.

If you are in the single map mode of the EXORciser/EXORset (as are most people) no connections need be made to the 'VUA', 'VXA' or 'PAGE' selection facilities. If you are using these modes of operation connect them as you would for any other Motorola board.

### A.7.1 ADDRESSING THE INTERFACE BOARD

To establish the base address of the board set the address switches 'ON' or 'OFF' for a '0' or '1' respectively that matches the base address you require. For example suppose you wish to set the board to appear at $EC20 (just above the parallel printer port location in some EXORcisers) the switches should be set as follows:

| A15 | A14 | A13 | A12 | A11 | A10 | A9 | A8 | A7 | A6 | A5 | A4 | A3 | A2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OFF | OFF | OFF | ON | OFF | OFF | ON | ON | ON | ON | OFF | ON | ON | ON |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 'E' | | | | 'C' | | | | '2' | | | | '0' | |

## A.8 INTERRUPTS

The MDOS and XDOS versions of this software do not require interrupts. If you are using these operating systems set the interrupt switches as follows:

IRQ = OFF
NMI = OFF

The OS9 version of this software DOES use interrupts. If you are using the OS9 version of the software set the interrupt switches as follows:

IRQ = ON
NMI = OFF

## A.9 MDOS/XDOS USER CONFIGURATION TABLE

PAGE 1: NOVEMBER 10 1983

```
 1 ************************************************
 2 *     UNIVERSAL EPROM PROGRAMMER II SOFTWARE     *
 3 ************************************************
 4
 5 ************************************************
 6 * THIS  SOFTWARE  IS COPYRIGHT (C) 1983 AND 1984 *
 7 * AND MAY NOT BE REPRODUCED IN WHOLE OR IN  PART *
 8 * BY  ANY  MEANS  WITHOUT  THE  EXPRESS  WRITTEN *
 9 * PERMISSION OF WINDRUSH MICRO SYSTEMS LIMITED.  *
10 ************************************************
11
12 ************************************************
13 * THIS SOFTWARE MAY BE MODIFIED BY THE  ORIGINAL *
14 * PURCHASER  FOR  THE  SOLE  USE OF THE ORIGINAL *
15 * PURCHASER BUT SUCH MODIFICATIONS  MAY  NOT  BE *
16 * PASSED  ON  TO  THIRD  PARTIES  BY  ANY  MEANS *
17 * WITHOUT  THE  EXPRESS  WRITTEN  PERMISSION  OF *
18 * WINDRUSH MICRO SYSTEMS LIMITED.                *
19 ************************************************
20
21 ************************************************
22 * WINDRUSH  MICRO  SYSTEMS  LIMITED,  WORSTEAD *
23 * LABORATORIES  NORTH WALSHAM, NORFOLK, ENGLAND. *
24 * TEL: (0692) 405189                             *
25 ************************************************
26
27 ************************************************
28 *                                              *
29 *     THIS VERSION IS FOR 6809 'MDOS/XDOS'     *
30 *                                              *
31 ************************************************
32
33 ************************************************
34 * THIS  SOFTWARE  COMPRISES EIGHT SECTIONS WHICH *
35 * MUST BE ASSEMBLED CONCURRENTLY:                *
36 *                                                *
37 * 1. UPRMQG.SA  I/O EQUATES FILE                 *
38 * 2. UPRM1G.SA  LOW-LEVEL CONSOLE I/O            *
39 * 3. UPRM2G.SA  HARDWARE DRIVERS                 *
40 * 4. UPRM3G.SA  MENUS                            *
41 * 5. UPRM4G.SA  FILL, MOVE, EXAMINE & CHANGE,    *
42 *                  DUMP, CRC, FIND               *
43 * 6. UPRM5G.SA  COPY, VERIFY, PROGRAM            *
44 * 7. UPRM6G.SA  TEXT MESSAGES                    *
45 * 8. UPRM7G.SA  DOS INTERFACE (NOT USED)         *
46 ************************************************
47
48 ************************************************
49 * THE  BUFFER  IS  DEFINED  BY  'BUFF.LO'  AND *
50 * 'BUFF.HI' WHICH MAY BE  ALTERED  TO  SUIT  THE *
51 * MEMORY  MAP  OF  THE  USERS  SYSTEM.  THESE *
52 * DEFINITIONS ARE MADE VIA 'FDB' STATEMENTS  AND *
53 * ARE  THEREFORE CAPABLE OF BEING PATCHED AT THE *
54 * BINARY LEVEL.                                  *
55 ************************************************
```

A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 2: NOVEMBER 10 1983

```
                        56 ******************
                        57 * VERSION NUMBER *
                        58 ******************
                        59 *
0058                    60 VER.1     EQU   'X
0058                    61 VER.2     EQU   'X
0058                    62 VER.3     EQU   'X
                        63
                        64
                        65 ***********************************
                        66 * PROM PROGRAMMER MEMORY ORIGINS *
                        67 ***********************************
                        68 *
6000                    69 LO.BUFF   EQU   $6000      START OF BUFFER
DFFF                    70 HI.BUFF   EQU   $DFFF      END OF BUFFER
                        71 *
3000                    72 PRMPRG    EQU   $3000      PROGRAM START ADDRESS
                        73 *
EC20                    74 PIA.PORT  EQU   $EC20      'EXORFACE' DEFAULT
                        75
                        76
                        77 ***************
                        78 * I/O EQUATES *
                        79 ***************
                        80 *
0000                    81 XSTAT     EQU   $0000      NOT AVAILABLE
F015                    82 XINPUT    EQU   $F015      GET CHAR AND ECHO IT
F018                    83 XOUTPUT   EQU   $F018      PUT CHAR ON SCREEN
                        84
                        85
                        86 ********************
                        87 * EXTERNAL EQUATES *
                        88 ********************
                        89 *
F000                    90 XMONITR   EQU   $F000      RESET ENTRY POINT
                        91
                        92
                        93 ********************************
                        94 * TERMINAL CONTROL CHARACTERS *
                        95 ********************************
                        96 *
0000                    97 NL        EQU   $00        USED FOR EOT
0004                    98 ET        EQU   $04        END OF TRANSMISSION
0007                    99 BL        EQU   $07        BELL
0008                   100 BS        EQU   $08        BACKSPACE
000A                   101 LF        EQU   $0A        LINE-FEED
000C                   102 HC        EQU   $0C        FORM-FEED (HOME AND CLR)
000D                   103 CR        EQU   $0D        CARRIAGE RETURN LINE-FEED
001B                   104 ES        EQU   $1B        ESCAPE
0020                   105 SP        EQU   $20        SPACE
```

A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 3: NOVEMBER 10 1983

```
                                107 ********************
                                108 * START OF PROGRAM *
                                109 ********************
                                110 *
0000                            111          SETDP $0000     PROGRAM WILL SET THE 'DP'
                                112
3000                            113          ORG   PRMPRG    PROGRAM START ADDRESS
                                114
3000   16 0075    (3078)        115 ENTRE   >LBRA  COLDS1    COLD START ENTRY POINT
3003   16 0074    (307A)        116 ENTRE2  >LBRA  WARMS1    WARM START ENTRY POINT
                                117
                                118
                                119 ****************************
                                120 * VERSION NUMBER IN ASCII *
                                121 ****************************
                                122 *
3006 563A                       123          FCC   /V:/
3008 58                         124          FCB   VER.1      :
3009 2E                         125          FCC   /./
300A 58                         126          FCB   VER.2      : VERSION NUMBER IN ASCII
300B 58                         127          FCB   VER.3      :
                                128
```

A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 4: NOVEMBER 10 1983

```
                        130 **************
                        131 * JUMP TABLE *
                        132 **************
                        133 *
                        134 ***********************************************
                        135 * EACH 'JUMP' IS FOLLOWED BY A 'NOP' TO ALLOW *
                        136 * THE  USER TO PATCH THIS TABLE TO POINT AT A *
                        137 * VECTOR TABLE RATHER THAN THE ENTRY POINT OF *
                        138 * THE  ACTUAL  ROUTINE  OR A JUMP TABLE. i.e. *
                        139 * THIS STRUCTURE ALLOWS YOU  TO 'JMP  [PROG]' *
                        140 ***********************************************
                        141 *
300C   1C FE            142 STAT     ANDCC #$FE      (CLEAR CARRY)
300E   39               143          RTS             RETURN WITH 'NO KEY PENDING'
300F   12               144          NOP
                        145
3010   7E F015          146 INPUT    JMP   XINPUT    GET CHAR FROM THE K/B
3013   12               147          NOP
                        148
3014   7E F018          149 OUTPUT   JMP   XOUTPUT   SEND CHAR TO THE CONSOLE
3017   12               150          NOP
                        151
3018   39               152 PINIT    RTS             INITIALIZE PRINTER DRIVERS
3019   12               153          NOP
301A   12               154          NOP
301B   12               155          NOP
                        156
301C   39               157 POUT     RTS             SEND CHAR TO LINE PRINTER
301D   12               158          NOP
301E   12               159          NOP
301F   12               160          NOP
                        161
3020   3F               162 SYS.DOS  SWI             SYSTEM CALL
3021 1A                 163          FCB   $1A        '.MDENT' OFFSET (BACK TO DOS)
3022   12               164          NOP
3023   12               165          NOP
                        166
3024   7E F000          167 SYS.MON  JMP   XMONITR   RETURN TO SYSTEM MONITOR
3027   12               168          NOP
```

## A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 5: NOVEMBER 10 1983

```
                    170 ****************************************
                    171 * SYSTEM SOFTWARE HARDWARE PATCH TABLE *
                    172 ****************************************
                    173 *
3028 6000           174 BUFF.LO  FDB   LO.BUFF     BASE OF BUFFER
302A DFFF           175 BUFF.HI  FDB   HI.BUFF     TOP OF BUFFER
                    176 *
302C EC20           177 PIAPORT  FDB   PIA.PORT    DEFAULT ADDRESS OF PIA
302E 00             178 PRMT.FLG FCB   0           = 0 ... NO PROMPT
                    179 *                          <>0 ... PROMPT FOR PORT
                    180 *
302F 05             181 NUL.CNT  FCB   5           NULLS AFTER CR-LF, ETC.
                    182 *
3030 00             183 STATMODE FCB   $00         $F0 = EQ,   $FF = NE
                    184 *                          $00 = CS,   $0F = CC
                    185
3031 37             186 DEF.EPRM FCB   '7          DEFAULT EPROM (ASCII)
                    187 *
3032 32             188 NORM.PUL FCB   50          DUMB PROGRAM PULSE WIDTH
3033 02             189 SPCL.PUL FCB   2           68764 PROGRAM PULSE WIDTH
3034 01             190 INTL.PUL FCB   1           INITIAL PROGRAM PULSE
                    191 *
3035 0F             192 PASSES   FCB   15          PASSES FOR MCM68764
                    193 *
3036 0F             194 LMT.1    FCB   15          INITIAL LIMIT FOR 2764/128
3037 04             195 OVP.1    FCB   4           MULTIPLIER FOR 2764/128
                    196 *
3038 19             197 LMT.2    FCB   25          LIMIT FOR 2764A/128A/256
3039 03             198 OVP.2    FCB   3           MULT FOR 2764A/128A/256
                    199 *
303A 00             200          FCB   0           RESERVED FOR INTEL 27512
303B 00             201          FCB   0           RESERVED FOR INTEL 27512
                    202 *
303C 00             203          FCB   0           RESERVED FOR AMD 27512
303D 00             204          FCB   0           RESERVED FOR AMD 27512
```

## A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 6: NOVEMBER 10 1983

```
303E 0010              206 COLS.S    FDB   16        SCREEN COLUMNS
3040 10                207 ROWS.S    FCB   16        SCREEN ROWS
                       208 *
3041 0010              209 COLS.P    FDB   16        PRINTER COLUMNS
3043 40                210 ROWS.P    FCB   64        PRINTER ROWS
3044 02                211 ROWS.B    FCB   2         PRINTER BLANK ROWS
                       212 *
3045 20                213 LO.ASCII  FCB   $20       LOWEST DISPLAYABLE CHAR
3046 7D                214 HI.ASCII  FCB   $7D       HIGHEST DISPLAYABLE CHAR
3047 7D                215 PR.ASCII  FCB   $7D       HIGHEST PRINTABLE CHAR
                       216 *
3048 0D                217 DEF.CODE  FCB   $0D       ACCEPT DEFAULT VALUES
3049 03                218 ABT.CODE  FCB   $03       ABORT AND RETURN TO MENU
                       219 *
304A 55                220 PREV1     FCB   'U        PREVIOUS #1
304B 5E                221 PREV2     FCB   '^        PREVIOUS #2
304C 2D                222 PREV3     FCB   '-        PREVIOUS #3
                       223 *
304D 20                224 NEXT1     FCB   $20       NEXT #1 (SPACE-BAR)
304E 0A                225 NEXT2     FCB   $0A       NEXT #2 (LINE-FEED)
304F 2B                226 NEXT3     FCB   '+        NEXT #3
                       227 *
3050 7F                228 PARITY    FCB   $7F       'AND' INCOMING CHAR
                       229
3050                   230 END.TBLE  EQU   *-1
```

A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 7: NOVEMBER 10 1983

```
                        232 ************************************************
                        233 * THE FOLLOWING TABLES MAY  BE  USED  TO  INVOKE *
                        234 * VIDEO  ATTRIBUTES  THAT  MAY BE AVAILABLE WITH *
                        235 * YOUR TERMINAL. THE SOFTWARE HAS BEEN  DESIGNED *
                        236 * TO  PROVIDE HIGHLIGHTED BANNERS WITH TERMINALS *
                        237 * THAT SUPPORT A REVERSE VIDEO ATTRIBUTE.        *
                        238 *                                                *
                        239 * IF YOU INSTALL A CODE SEQUENCE IN EACH OF  THE *
                        240 * THREE  TABLES  THE SOFTWARE WILL AUTOMATICALLY *
                        241 * USE IT IN PREFERENCE TO  THE  'DUMB'  TERMINAL *
                        242 * DRIVERS  THAT DEFAULT IF THE FIRST BYTE IN THE *
                        243 * TABLE IS $04. SOME TERMINALS  DO  NOT  HAVE  A *
                        244 * REVERSE  VIDEO ATTRIBUTE BUT HAVE A REDUCED OR *
                        245 * AN INTENSIFIED ATTRIBUTE. WE'LL  LEAVE  IT  TO *
                        246 * YOU  TO  DECIDE IF THE DISPLAY THAT RESULTS IS *
                        247 * ACCEPTABLE TO YOU. IF IT IS NOT THEN DON'T USE *
                        248 * THE VIDEO ATTRIBUTES.                          *
                        249 ************************************************
                        250
3060                    251           ORG   PRMPRG+$60 LEAVE ROOM FOR TABLE
                        252
3060 04                 253 HOME.CLR  FCB   ET
3061 04                 254           FCB   ET
3062 04                 255           FCB   ET
3063 04                 256           FCB   ET
3064 04                 257           FCB   ET
3065 04                 258           FCB   ET
3066 04                 259           FCB   ET
3067 04                 260           FCB   ET          DON'T ALTER!
                        261 *
3068 04                 262 ATTR.ON   FCB   ET
3069 04                 263           FCB   ET
306A 04                 264           FCB   ET
306B 04                 265           FCB   ET
306C 04                 266           FCB   ET
306D 04                 267           FCB   ET
306E 04                 268           FCB   ET
306F 04                 269           FCB   ET          DON'T ALTER!
                        270 *
3070 04                 271 ATTR.OFF  FCB   ET
3071 04                 272           FCB   ET
3072 04                 273           FCB   ET
3073 04                 274           FCB   ET
3074 04                 275           FCB   ET
3075 04                 276           FCB   ET
3076 04                 277           FCB   ET
3077 04                 278           FCB   ET          DON'T ALTER!
                        279
                        280 ******************
                        281 * ENTRY BRANCHES *
                        282 ******************
                        283 *
3078                    284 COLDS1    EQU   *
307A                    285 WARMS1    EQU   *+2
3000                    286           END   PRMPRG
```

## A.9 MDOS/XDOS USER CONFIGURATION TABLE (continued)

PAGE 8: NOVEMBER 10 1983

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ABT.CODE | 3049 | ATTR.OFF | 3070 | ATTR.ON | 3068 | BL | 0007 | BS | 0008 |
| BUFF.HI | 302A | BUFF.LO | 3028 | COLDS1 | 3078 | COLS.P | 3041 | COLS.S | 303E |
| CR | 000D | DEF.CODE | 3048 | DEF.EPRM | 3031 | END.TBLE | 3050 | ENTRE | 3000 |
| ENTRE2 | 3003 | ES | 001B | ET | 0004 | HC | 000C | HI.ASCII | 3046 |
| HI.BUFF | DFFF | HOME.CLR | 3060 | INPUT | 3010 | INTL.PUL | 3034 | LF | 000A |
| LMT.1 | 3036 | LMT.2 | 3038 | LO.ASCII | 3045 | LO.BUFF | 6000 | NEXT1 | 304D |
| NEXT2 | 304E | NEXT3 | 304F | NL | 0000 | NORM.PUL | 3032 | NUL.CNT | 302F |
| OUTPUT | 3014 | OVP.1 | 3037 | OVP.2 | 3039 | PARITY | 3050 | PASSES | 3035 |
| PIA.PORT | EC20 | PIAPORT | 302C | PINIT | 3018 | POUT | 301C | PR.ASCII | 3047 |
| PREV1 | 304A | PREV2 | 304B | PREV3 | 304C | PRMPRG | 3000 | PRMT.FLG | 302E |
| ROWS.B | 3044 | ROWS.P | 3043 | ROWS.S | 3040 | SP | 0020 | SPCL.PUL | 3033 |
| STAT | 300C | STATMODE | 3030 | SYS.DOS | 3020 | SYS.MON | 3024 | VER.1 | 0058 |
| VER.2 | 0058 | VER.3 | 0058 | WARMS1 | 307A | XINPUT | F015 | XMONITR | F000 |
| XOUTPUT | F018 | XSTAT | 0000 | | | | | | |

## B.1 INTRODUCTION

This section of the manual is provided to document the differences between the OS9 version of the accompanying software and the FLEX version as described in the body of this document.

## B.2 WHAT YOU SHOULD GET

The floppy disk you receive with this product should contain, as a minimum, the following files:

UPRII ... the main executable program (only one version ... object only)
UPRIIDEV ... the device descriptor (several versions ... source included)
UPRIIDVR ... the device driver (several versions ... source included)
READ_ME ... tells you what the various files on the disk are for

## B.3 MAJOR DIFFERENCES

The command option '/' (EXECUTE A FLEX COMMAND) is replaced by a command that allows complete access to a new OS9 'shell'. Typing <ESCAPE> will terminate the 'shell' and return to the parent process (the EPROM programmer software). Binary file 'READ' and 'WRITE' utilities are added to the main operations menu. The software no longer 'hoggs' the system as did the first release. This version schedules itself just like any other task and will hence work properly in a multi-user environment. You should note however that due to the system task switching overheads associated with multi-user operation (multi-user operation in OS9 level two in particular) the software will run SIGNIFICANTLY slower than it does under FLEX.

## B.4 GETTING IT UP

To get the OS9 version of the EPROM programmer software up and running you must LOAD one of the device descriptors (prefixed 'UPRIIDEV') that matches the hardware memory map of your system (see section b.5). The device descriptors main job is to tell the device driver where it can find the PIA on the interface board. The next task is to LOAD the device driver (prefixed 'UPRIIDVR') that matches the version of OS9 you are using. The device drivers main job is drive the EPROM programmer pod hardware. The last task is to invoke the main program 'UPRII'. The main program contains the operator interface, buffer manipulation utilities and controls the programmer pod via the device driver. If you wish to use the ability of the software to produce a formatted dump on the printer you must redirect the output WHEN YOU INVOKE UPRII thus: 'UPRII >P<CR>'.

The following is the sequence required to use the software with a GIMIX OS9, level 2, version 1.2 system (assuming that all files reside in the commands directory on your system disk):

```
OS9: load UPRIIDEV_GMX<RETURN>          (loads the device descriptor)
OS9: load UPRIIDVR_L2V1.2<RETURN>       (loads the device driver)
OS9: load UPRII<RETURN>                 (loads the main program)
OS9: UPRII >P<RETURN>                   (directs 'DUMP' to parallel printer)
```

## B.4 GETTING IT UP (continued)

Once you have established which descriptor and driver you wish to use we suggest that you use the OS9 'MERGE' command to create a new file that consists of the three required elements as defined above. For example ....

OS9: MERGE UPRIIDEV_GMX UPRIIDVR_L2V1.2 UPRII >MY_UPRII<RETURN>

Don't forget to use 'ATTR' to set the exececute flags (e pe) on the file created. For example ....

OS9: ATTR MY_UPRII e pe<RETURN>

## B.5 LOADING BINARY FILES INTO THE PROGRAMMER BUFFER

Disk memory image binary files produced by an assembler, compiler, etc. will often be used a the source of code to program into an EPROM. In order to program this code into an EPROM it must be loaded into the programmer buffer.

Before you load the buffer we suggest that you use the UPROM software to pre-load the buffer area with $FF (using the FILL operation) as this will make the program code easier to locate and will also prevent random data in the buffer from being programmed into the EPROM.

The OS9 version of this software has two additional items in the main operations menu. The 'READ' operation can be selected by typing the letter 'R'. The 'WRITE' operation can be selected by typing the letter 'W'. The two operations work substantially the same. The following text will describe the 'READ' operation with the differences noted in brackets thus (...).

### 1. ENTER NAME OF FILE: -/D0/CMDS/UPRII<CR>

Type in the usual path followed by the name of the desired file. This example specifies a file named UPRII in the CMDS directory of device D0.

### 2. ENTER FILE OFFSET: $0000<CR> (no default)

Enter, in HEX, the number of bytes you wish to go 'into' the file before you start reading (writing) data. This example starts loading (writing) at the beginning of the file.

### 3. ENTER START ADDRESS: $0800<CR> (default is base of buffer)

Enter, in HEX, the position in the buffer you wish to have the binary file begin writing (reading) data. This example starts at $0800 in the buffer.

### 4. ENTER END ADDRESS: $1FFF<CR> (default is end of buffer)

Enter, in HEX, the position in the buffer you wish to end writing (reading data). This example ends at $1FFF in the buffer.

If any OS9 system errors occur they will be reported and control will return to the OPERATIONS MENU.

## B.6 INTERFACE BOARDS AND OS9 DEVICE DESCRIPTORS

The main purpose of the device descriptor is to define the base address of the PIA on the interface board. The OS9 device descriptor supplied with this package have been configured for several 'standard' locations as follows:

| | | | |
|---|---|---|---|
| UPRIIDEV_SSB | SSB | (S30 BUS) | $F7FC (port 7) |
| UPRIIDEV_GMX | GIMIX/WMS | (S30 BUS) | $E070 (port 7) |
| UPRIIDEV_WMS | WINDRUSH | (S50 BUS) | $E090 (GEN-IO board) |
| URPIIDEV_EXOR | MOTOROLA | (EXORBUS) | $EC20 |

If none of the supplied drivers match the memory map of your system you will have to edit the supplied source file and reassemble it. See the 'READ-ME' file on the disk for further information.

We supply several different types of interface boards. The S-30 boards are simply plugged into an appropriate bus slot and the motherboard provides all of the decoding. The Windrush S-50 board is pre-decoded to place the EPROM programmer interface at $E090. $E090 is also the standard address we use in our 3U and 6U eurocard systems.

The Exorciser interface board is described in the MDOS addendum. The Windrush EURO-BUS boards are described in their own manuals called '3U UPRM' and '6U UPRM' which are supplied separately.

General information on all three boards is provided in the form of schematic diagrams at the end of this document.

## B.7 OS9 DEVICE DRIVERS

The main purpose of the device driver is to control the EPROM programmer pod hardware whilst OS9 is in 'system' state. As there are various versions of OS9 around we have supplied drivers for four of the most recent versions:

| | | |
|---|---|---|
| UPRIIDVR_L1V1.1 | ... | LEVEL 1, VERSION 1.1 (tested on SSB SYSTEM) |
| UPRIIDVR_L1V1.2 | ... | LEVEL 1, VERSION 1.2 (tested on GIMIX SYSTEM) |
| UPRIIDVR_L2V1.0 | ... | LEVEL 2, VERSION 1.0 (tested on SSB SYSTEM) |
| UPRIIDVR_L2V1.2 | ... | LEVEL 2, VERSION 1.2 (tested on GIMIX SYSTEM) |

If the device drivers we supply do not work with your system you will have to assemble the source file of the most appropriate one with the OS9 'DEFS' files that came with your system. If your version of OS9 has been implemented correctly this is all the action that should be required. See the 'READ_ME' file on the disk for further information.

As we only have access to the equipment described above we can only guarantee that our software will work in the above hardware configurations with the above versions of OS9. If you have any problems we would like to hear about them and details of your hardware configuration.

## B.8 INTERRUPTS

The OS9 version of this software DOES use interrupts. If you are using the OS9 version of the software set the interrupt switches as follows:

IRQ = ON ('A' AND 'B' PORTS)
NMI = OFF (if present)

## B.9 OS9 USER CONFIGURATION INFORMATION

If you wish to modify any of the characteristics of the main software we recommend that you purchase the source package from us. For those of you who wish to 'have a go' at patching the user software with the OS9 'DEBUG' the information in the file entitled 'UPRII_USER_EQUATES' may be useful.

The aforementioned file is a partial listing of the assembly of the files that comprise the main program 'UPRII' and hence contains information about the relative positions of the user modifiable variables described in section 12. The variables have the same relative effect as the variables the FLEX version.

## C.1 INTRODUCTION

This section of the manual is provided to document the differences between the SSB DOS version of the accompanying software and the FLEX version as described in the body of this document.

## C.2 WHAT YOU SHOULD GET

The floppy disk you receive with this package should have the following files on it:

| | | |
|---|---|---|
| UPRII .$ | ... | the executable object file |
| UPRMEQ.ASM | ... | the software tables which may be modified by the user |
| README.TXT | ... | tells you what is on the disk |

## C.3 MAJOR DIFFERENCES

The initial memory range checks performed by the FLEX version are not implemented. The command option '/' (EXECUTE A FLEX COMMAND) is not available. The HEX/ASCII dump routine printer drivers are not implemented but they may be installed by the user by a patch to the software tables (see section 12). The default address of the PIA is $F7FC (PORT #7).

## C.4 MEMORY MAP

The SSB DOS software memory map is identical to that of FLEX (see section 3.2).

## C.5 COLD AND WARM START ENTRY

The SSB DOS software entry points are identical to those of FLEX.

## C.6 LOADING BINARY FILES INTO THE PROGRAMMER BUFFER

Disk memory image binary files produced by an assembler, compiler, etc. or RAM saved onto disk using the SAVE command will often be used a the source of code to program into an EPROM.

In order to program this code into an EPROM it must be loaded into the programmer buffer area which resides from $0000 to $7FFF. SSB DOS provides two utilities to assist this operation: 'FIND' and 'GET'.

Before you load the buffer we suggest that you use the UPROM software to pre-load the buffer area with $FF (using the FILL operation) as this will make the program code easier to locate and will also prevent random data in the buffer from being programmed into the EPROM.

## C.6 LOADING BINARY FILES INTO THE PROGRAMMER BUFFER (continued)

Providing that the binary file loads into RAM within the confines of the buffer area simply typing:

```
GET,1:FILE.NAM<CR>
```

will load the file into memory. This example assumes that the file is called 'FILE.NAM' and resides on drive #1. It does not matter if the code starts loading at an address other than $0000 (as long as it does not load past $7FFF) as the code can be moved around in the buffer area by the MOVE command available in the programmer software.

If the object file loads outside of the buffer area, i.e. it loads between $8000 and $FFFF you must use an offset when invoking the 'GET' utility. To calculate the offset proceed as follows:

Use the DOS 'FIND' utility to acertain where the file starts loading, e.g.:

```
FIND,1:FILE.NAM<CR>
```

Subtract the starting address from $FFF and then add $0001 to the result. For example suppose the file normally started loading at $4000, the calculations would be as follows:

$FFFF - $4000 = $BFFF . . . . . $BFFF + $0001 = $C000 which is the offset.

To load the file you would then enter:

```
GET,1:FILE.NAM,C000<CR>
```

This will load the file into the buffer starting at $0000. Refer to the section on the FLEX 'LOAD' utility for a discussion on the calculations required for files that have an address in the middle of a file that is lower than the first address of the file. The same rules apply to SSB DOS.

To save code out to disk that is present in the programmer buffer area you must return to DOS and use the 'SAVE' command.

## C.7 INTERRUPTS

The SSB DOS version of this software does not require interrupts.

## C.8 SSB DOS USER CONFIGURATION TABLE

PAGE 1: NOVEMBER 10 1983

```
 1 ************************************************
 2 *     UNIVERSAL EPROM PROGRAMMER II SOFTWARE   *
 3 ************************************************
 4
 5 ************************************************
 6 * THIS  SOFTWARE  IS COPYRIGHT (C) 1983 AND 1984 *
 7 * AND MAY NOT BE REPRODUCED IN WHOLE OR IN  PART *
 8 * BY  ANY  MEANS  WITHOUT  THE  EXPRESS  WRITTEN *
 9 * PERMISSION OF WINDRUSH MICRO SYSTEMS LIMITED.  *
10 ************************************************
11
12 ************************************************
13 * THIS SOFTWARE MAY BE MODIFIED BY THE  ORIGINAL *
14 * PURCHASER  FOR  THE  SOLE  USE OF THE ORIGINAL *
15 * PURCHASER BUT SUCH MODIFICATIONS  MAY  NOT  BE *
16 * PASSED  ON  TO  THIRD  PARTIES  BY  ANY  MEANS *
17 * WITHOUT  THE  EXPRESS  WRITTEN  PERMISSION  OF *
18 * WINDRUSH MICRO SYSTEMS LIMITED.                *
19 ************************************************
20
21 ************************************************
22 * WINDRUSH  MICRO  SYSTEMS  LIMITED,  WORSTEAD *
23 * LABORATORIES  NORTH WALSHAM, NORFOLK, ENGLAND. *
24 * TEL: (0692) 405189                             *
25 ************************************************
26
27 ************************************************
28 *                                              *
29 *      THIS VERSION IS FOR 6809 'SSB DOS'      *
30 *                                              *
31 ************************************************
32
33 ************************************************
34 * THIS  SOFTWARE  COMPRISES EIGHT SECTIONS WHICH *
35 * MUST BE ASSEMBLED CONCURRENTLY:                *
36 *                                                *
37 * 1. UPRMQG.ASM  I/O EQUATES FILE                *
38 * 2. UPRM1G.ASM  LOW-LEVEL CONSOLE I/O           *
39 * 3. UPRM2G.ASM  HARDWARE DRIVERS                *
40 * 4. UPRM3G.ASM  MENUS                           *
41 * 5. UPRM4G.ASM  FILL, MOVE, EXAMINE & CHANGE,   *
42 *                DUMP, CRC, FIND                 *
43 * 6. UPRM5G.ASM  COPY, VERIFY, PROGRAM           *
44 * 7. UPRM6G.ASM  TEXT MESSAGES                   *
45 * 8. UPRM7G.ASM  DOS INTERFACE (NOT USED)        *
46 ************************************************
47
48 ************************************************
49 * THE   BUFFER   IS  DEFINED  BY  'BUFF.LO'  AND *
50 * 'BUFF.HI' WHICH MAY BE  ALTERED  TO  SUIT  THE *
51 * MEMORY   MAP   OF   THE  USERS  SYSTEM.  THESE *
52 * DEFINITIONS ARE MADE VIA 'FDB' STATEMENTS  AND *
53 * ARE  THEREFORE CAPABLE OF BEING PATCHED AT THE *
54 * BINARY LEVEL.                                  *
55 ************************************************
```

C.8 SSB DOS USER CONFIGURATION TABLE (continued)

PAGE 2: NOVEMBER 10 1983

```
                    56 ******************
                    57 * VERSION NUMBER *
                    58 ******************
                    59 *
0034                60 VER.1     EQU   '4
0032                61 VER.2     EQU   '2
0031                62 VER.3     EQU   '1
                    63
                    64
                    65 ***********************************
                    66 * PROM PROGRAMMER MEMORY ORIGINS *
                    67 ***********************************
                    68 *
0000                69 LO.BUFF   EQU   $0000     START OF BUFFER
7FFF                70 HI.BUFF   EQU   $7FFF     END OF BUFFER
                    71 *
8000                72 PRMPRG    EQU   $8000     PROGRAM START ADDRESS
                    73 *
F7FC                74 PIA.PORT  EQU   $F7FC     S-30 PORT #7
                    75
                    76
                    77 ***************
                    78 * I/O EQUATES *
                    79 ***************
                    80 *
D2C7                81 XSTAT     EQU   $D2C7     CARRY SET FOR KEY PENDING
D289                82 XINPUT    EQU   $D289     GET CHAR AND ECHO IT
D286                83 XOUTPUT   EQU   $D286     PUT CHAR ON SCREEN
                    84
                    85
                    86 *********************
                    87 * EXTERNAL EQUATES *
                    88 *********************
                    89 *
D28C                90 XMONITR   EQU   $D28C     RE-ENTER SYSTEM MONITOR
D283                91 XDOS      EQU   $D283     RE-ENTER DOS
                    92
                    93
                    94 ********************************
                    95 * TERMINAL CONTROL CHARACTERS *
                    96 ********************************
                    97 *
0000                98 NL        EQU   $00       USED FOR EOT
0004                99 ET        EQU   $04       END OF TRANSMISSION
0007               100 BL        EQU   $07       BELL
0008               101 BS        EQU   $08       BACKSPACE
000A               102 LF        EQU   $0A       LINE-FEED
000C               103 HC        EQU   $0C       FORM-FEED (HOME AND CLR)
000D               104 CR        EQU   $0D       CARRIAGE RETURN LINE-FEED
001B               105 ES        EQU   $1B       ESCAPE
0020               106 SP        EQU   $20       SPACE
```

## C.8 SSB DOS USER CONFIGURATION TABLE (continued)

PAGE 3: NOVEMBER 10 1983

```
                                 108 ********************
                                 109 * START OF PROGRAM *
                                 110 ********************
                                 111 *
0000                             112           SETDP $0000     PROGRAM WILL SET THE 'DP'
                                 113
8000                             114           ORG   PRMPRG    PROGRAM START ADDRESS
                                 115
8000   16 0075    (8078)         116 ENTRE    >LBRA  COLDS1    COLD START ENTRY POINT
8003   16 0074    (807A)         117 ENTRE2   >LBRA  WARMS1    WARM START ENTRY POINT
                                 118
                                 119
                                 120 ****************************
                                 121 * VERSION NUMBER IN ASCII *
                                 122 ****************************
                                 123 *
8006 563A                        124           FCC   /V:/
8008 34                          125           FCB   VER.1      :
8009 2E                          126           FCC   /./
800A 32                          127           FCB   VER.2      : VERSION NUMBER IN ASCII
800B 31                          128           FCB   VER.3      :
                                 129
                                 130
```

C.8 SSB DOS USER CONFIGURATION TABLE (continued)

PAGE 4: NOVEMBER 10 1983

```
                        132 **************
                        133 * JUMP TABLE *
                        134 **************
                        135 *
                        136 ***********************************************
                        137 * EACH 'JUMP' IS FOLLOWED BY A 'NOP' TO ALLOW *
                        138 * THE  USER TO PATCH THIS TABLE TO POINT AT A *
                        139 * VECTOR TABLE RATHER THAN THE ENTRY POINT OF *
                        140 * THE  ACTUAL  ROUTINE  OR A JUMP TABLE. i.e. *
                        141 * THIS STRUCTURE ALLOWS YOU  TO 'JMP  [PROG]' *
                        142 ***********************************************
                        143 *
800C  7E D2C7           144 STAT     JMP   XSTAT     CARRY SET = KEY PENDING
800F  12                145          NOP
                        146
8010  BD D289           147 INPUT    JSR   XINPUT    GET CHAR FROM THE K/B
8013  12                148          NOP             DROP THROUGH & ECHO IT
                        149
8014  7E D286           150 OUTPUT   JMP   XOUTPUT   SEND CHAR TO THE CONSOLE
8017  12                151          NOP
                        152
8018  39                153 PINIT    RTS             INITIALIZE PRINTER DRIVERS
8019  12                154          NOP
801A  12                155          NOP
801B  12                156          NOP
                        157
801C  39                158 POUT     RTS             SEND CHAR TO LINE PRINTER
801D  12                159          NOP
801E  12                160          NOP
801F  12                161          NOP
                        162
8020  7E D283           163 SYS.DOS  JMP   XDOS       RETURN TO DOS
8023  12                164          NOP
                        165
8024  7E D28C           166 SYS.MON  JMP   XMONITR    RETURN TO SYSTEM MONITOR
8027  12                167          NOP
```

C.8 SSB DOS USER CONFIGURATION TABLE (continued)

PAGE 5: NOVEMBER 10 1983

```
                        169 ****************************************
                        170 * SYSTEM SOFTWARE HARDWARE PATCH TABLE *
                        171 ****************************************
                        172 *
8028 0000               173 BUFF.LO   FDB   LO.BUFF     BASE OF BUFFER
802A 7FFF               174 BUFF.HI   FDB   HI.BUFF     TOP OF BUFFER
                        175 *
802C F7FC               176 PIAPORT   FDB   PIA.PORT    DEFAULT ADDRESS OF PIA
802E 00                 177 PRMT.FLG  FCB   0           = 0 ... NO PROMPT
                        178 *                           <>0 ... PROMPT FOR PORT
                        179 *
802F 05                 180 NUL.CNT   FCB   5           NULLS AFTER CR-LF, ETC.
                        181 *
8030 00                 182 STATMODE  FCB   $00         $F0 = EQ,   $FF = NE
                        183 *                           $00 = CS,   $0F = CC
                        184
8031 37                 185 DEF.EPRM  FCB   '7          DEFAULT EPROM (ASCII)
                        186 *
8032 32                 187 NORM.PUL  FCB   50          DUMB PROGRAM PULSE WIDTH
8033 02                 188 SPCL.PUL  FCB   2           68764 PROGRAM PULSE WIDTH
8034 01                 189 INTL.PUL  FCB   1           INITIAL PROGRAM PULSE
                        190 *
8035 0F                 191 PASSES    FCB   15          PASSES FOR MCM68764
                        192 *
8036 0F                 193 LMT.1     FCB   15          INITIAL LIMIT FOR 2764/128
8037 04                 194 OVP.1     FCB   4           MULTIPLIER FOR 2764/128
                        195 *
8038 19                 196 LMT.2     FCB   25          LIMIT FOR 2764A/128A/256
8039 03                 197 OVP.2     FCB   3           MULT FOR 2764A/128A/256
                        198 *
803A 00                 199           FCB   0           RESERVED FOR INTEL 27512
803B 00                 200           FCB   0           RESERVED FOR INTEL 27512
                        201 *
803C 00                 202           FCB   0           RESERVED FOR AMD 27512
803D 00                 203           FCB   0           RESERVED FOR AMD 27512
```

## C.8 SSB DOS USER CONFIGURATION TABLE (continued)

PAGE 6: NOVEMBER 10 1983

```
803E 0010          205 COLS.S    FDB   16        SCREEN COLUMNS
8040 10            206 ROWS.S    FCB   16        SCREEN ROWS
                   207 *
8041 0010          208 COLS.P    FDB   16        PRINTER COLUMNS
8043 40            209 ROWS.P    FCB   64        PRINTER ROWS
8044 02            210 ROWS.B    FCB   2         PRINTER BLANK ROWS
                   211 *
8045 20            212 LO.ASCII  FCB   $20       LOWEST DISPLAYABLE CHAR
8046 7D            213 HI.ASCII  FCB   $7D       HIGHEST DISPLAYABLE CHAR
8047 7D            214 PR.ASCII  FCB   $7D       HIGHEST PRINTABLE CHAR
                   215 *
8048 0D            216 DEF.CODE  FCB   $0D       ACCEPT DEFAULT VALUES
8049 03            217 ABT.CODE  FCB   $03       ABORT AND RETURN TO MENU
                   218 *
804A 55            219 PREV1     FCB   'U        PREVIOUS #1
804B 5E            220 PREV2     FCB   '^        PREVIOUS #2
804C 2D            221 PREV3     FCB   '-        PREVIOUS #3
                   222 *
804D 20            223 NEXT1     FCB   $20       NEXT #1 (SPACE-BAR)
804E 0A            224 NEXT2     FCB   $0A       NEXT #2 (LINE-FEED)
804F 2B            225 NEXT3     FCB   '+        NEXT #3
                   226 *
8050 7F            227 PARITY    FCB   $7F       'AND' INCOMING CHAR
                   228
8050               229 END.TBLE  EQU   *-1
```

NOTE: The variable called 'PARITY' is logically 'ANDed' with the incoming keyboard character to strip out the the MSB (parity) bit as the SSB DOS routine does not perform this function.

C.8 SSB DOS USER CONFIGURATION TABLE (continued)

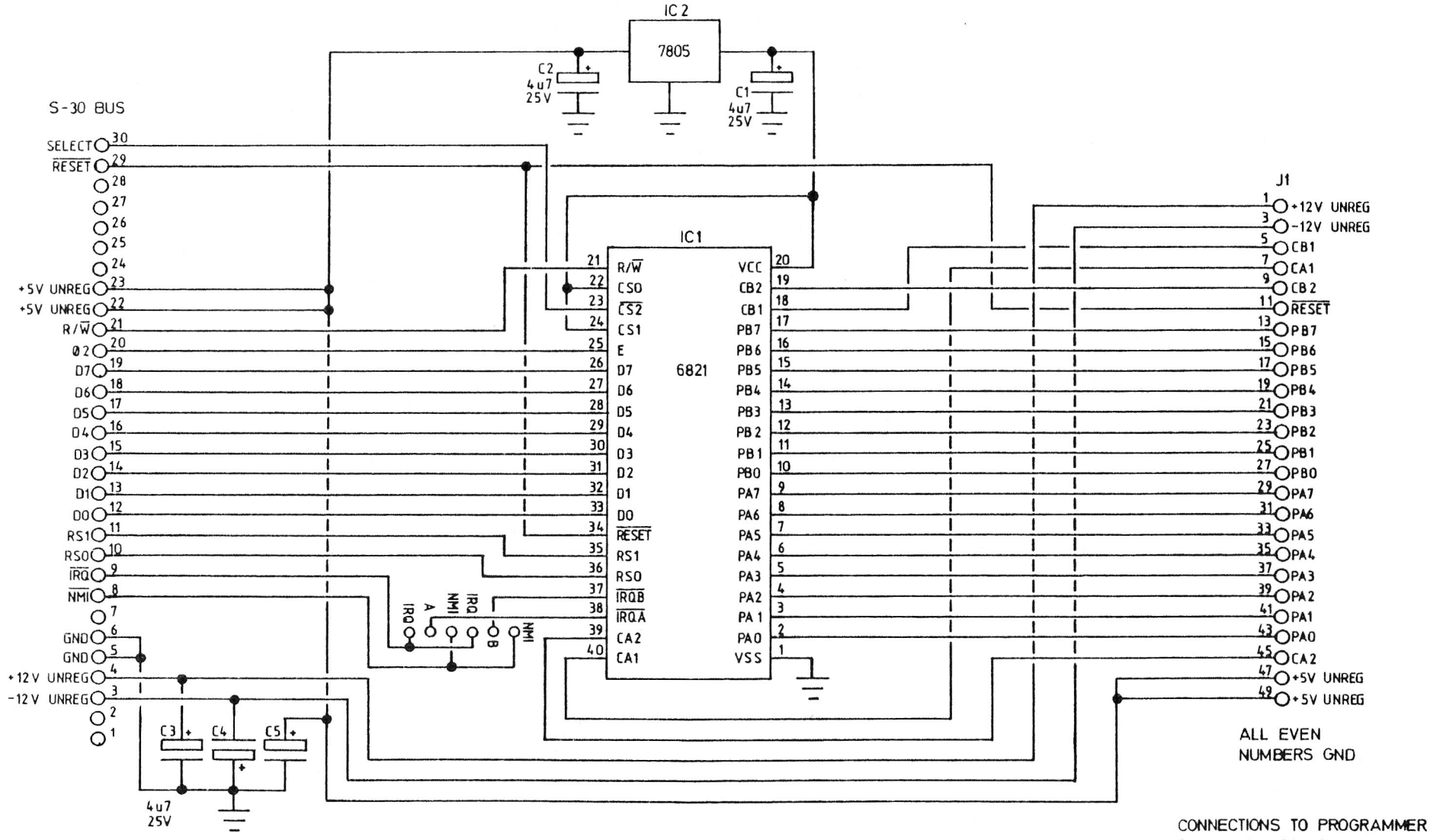
PAGE 7: NOVEMBER 10 1983

```
                        231 ************************************************
                        232 * THE FOLLOWING TABLES MAY  BE  USED  TO  INVOKE *
                        233 * VIDEO  ATTRIBUTES  THAT  MAY BE AVAILABLE WITH *
                        234 * YOUR TERMINAL. THE SOFTWARE HAS BEEN  DESIGNED *
                        235 * TO  PROVIDE HIGHLIGHTED BANNERS WITH TERMINALS *
                        236 * THAT SUPPORT A REVERSE VIDEO ATTRIBUTE.        *
                        237 *                                                *
                        238 * IF YOU INSTALL A CODE SEQUENCE IN EACH OF  THE *
                        239 * THREE  TABLES  THE SOFTWARE WILL AUTOMATICALLY *
                        240 * USE IT IN PREFERENCE TO  THE  'DUMB'  TERMINAL *
                        241 * DRIVERS  THAT DEFAULT IF THE FIRST BYTE IN THE *
                        242 * TABLE IS $04. SOME TERMINALS  DO  NOT  HAVE  A *
                        243 * REVERSE  VIDEO ATTRIBUTE BUT HAVE A REDUCED OR *
                        244 * AN INTENSIFIED ATTRIBUTE. WE'LL  LEAVE  IT  TO *
                        245 * YOU  TO  DECIDE IF THE DISPLAY THAT RESULTS IS *
                        246 * ACCEPTABLE TO YOU. IF IT IS NOT THEN DON'T USE *
                        247 * THE VIDEO ATTRIBUTES.                          *
                        248 ************************************************
                        249
8060                    250          ORG   PRMPRG+$60 LEAVE ROOM FOR TABLE
                        251
8060 04                 252 HOME.CLR FCB   ET
8061 04                 253          FCB   ET
8062 04                 254          FCB   ET
8063 04                 255          FCB   ET
8064 04                 256          FCB   ET
8065 04                 257          FCB   ET
8066 04                 258          FCB   ET
8067 04                 259          FCB   ET          DON'T ALTER!
                        260 *
8068 04                 261 ATTR.ON  FCB   ET
8069 04                 262          FCB   ET
806A 04                 263          FCB   ET
806B 04                 264          FCB   ET
806C 04                 265          FCB   ET
806D 04                 266          FCB   ET
806E 04                 267          FCB   ET
806F 04                 268          FCB   ET          DON'T ALTER!
                        269 *
8070 04                 270 ATTR.OFF FCB   ET
8071 04                 271          FCB   ET
8072 04                 272          FCB   ET
8073 04                 273          FCB   ET
8074 04                 274          FCB   ET
8075 04                 275          FCB   ET
8076 04                 276          FCB   ET
8077 04                 277          FCB   ET          DON'T ALTER!
                        278
                        279 ******************
                        280 * ENTRY BRANCHES *
                        281 ******************
                        282 *
8078                    283 COLDS1   EQU   *
807A                    284 WARMS1   EQU   *+2
```

## C.8 SSB DOS USER CONFIGURATION TABLE (continued)

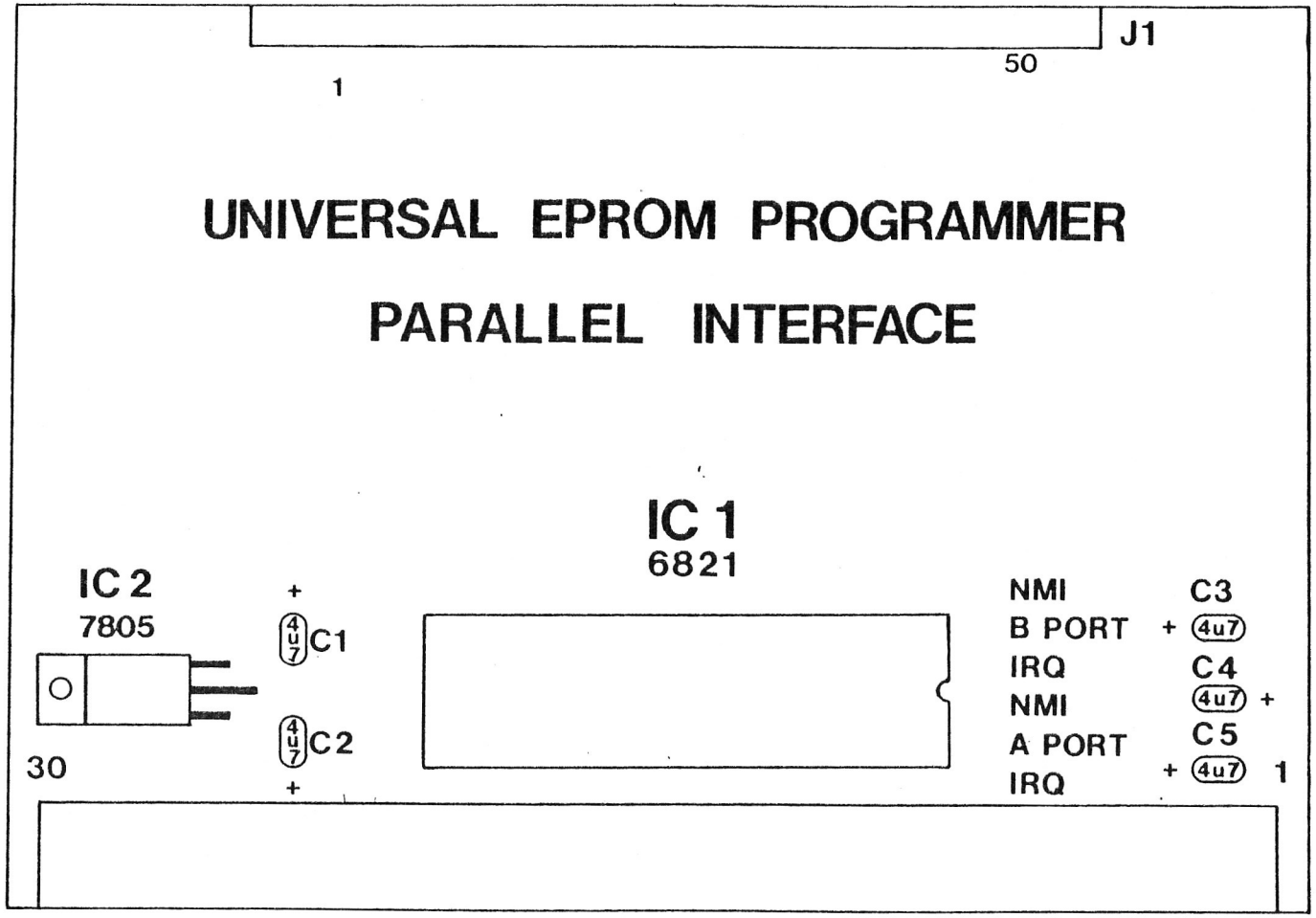
PAGE 8: NOVEMBER 10 1983

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ABT.CODE | 8049 | ATTR.OFF | 8070 | ATTR.ON | 8068 | BL | 0007 | BS | 0008 |
| BUFF.HI | 802A | BUFF.LO | 8028 | COLDS1 | 8078 | COLS.P | 8041 | COLS.S | 803E |
| CR | 000D | DEF.CODE | 8048 | DEF.EPRM | 8031 | END.TBLE | 8050 | ENTRE | 8000 |
| ENTRE2 | 8003 | ES | 001B | ET | 0004 | HC | 000C | HI.ASCII | 8046 |
| HI.BUFF | 7FFF | HOME.CLR | 8060 | INPUT | 8010 | INTL.PUL | 8034 | LF | 000A |
| LMT.1 | 8036 | LMT.2 | 8038 | LO.ASCII | 8045 | LO.BUFF | 0000 | NEXT1 | 804D |
| NEXT2 | 804E | NEXT3 | 804F | NL | 0000 | NORM.PUL | 8032 | NUL.CNT | 802F |
| OUTPUT | 8014 | OVP.1 | 8037 | OVP.2 | 8039 | PARITY | 8050 | PASSES | 8035 |
| PIA.PORT | F7FC | PIAPORT | 802C | PINIT | 8018 | POUT | 801C | PR.ASCII | 8047 |
| PREV1 | 804A | PREV2 | 804B | PREV3 | 804C | PRMPRG | 8000 | PRMT.FLG | 802E |
| ROWS.B | 8044 | ROWS.P | 8043 | ROWS.S | 8040 | SP | 0020 | SPCL.PUL | 8033 |
| STAT | 800C | STATMODE | 8030 | SYS.DOS | 8020 | SYS.MON | 8024 | VER.1 | 0034 |
| VER.2 | 0032 | VER.3 | 0031 | WARMS1 | 807A | XDOS | D283 | XINPUT | D289 |
| XMONITR | D28C | XOUTPUT | D286 | XSTAT | D2C7 | | | | |

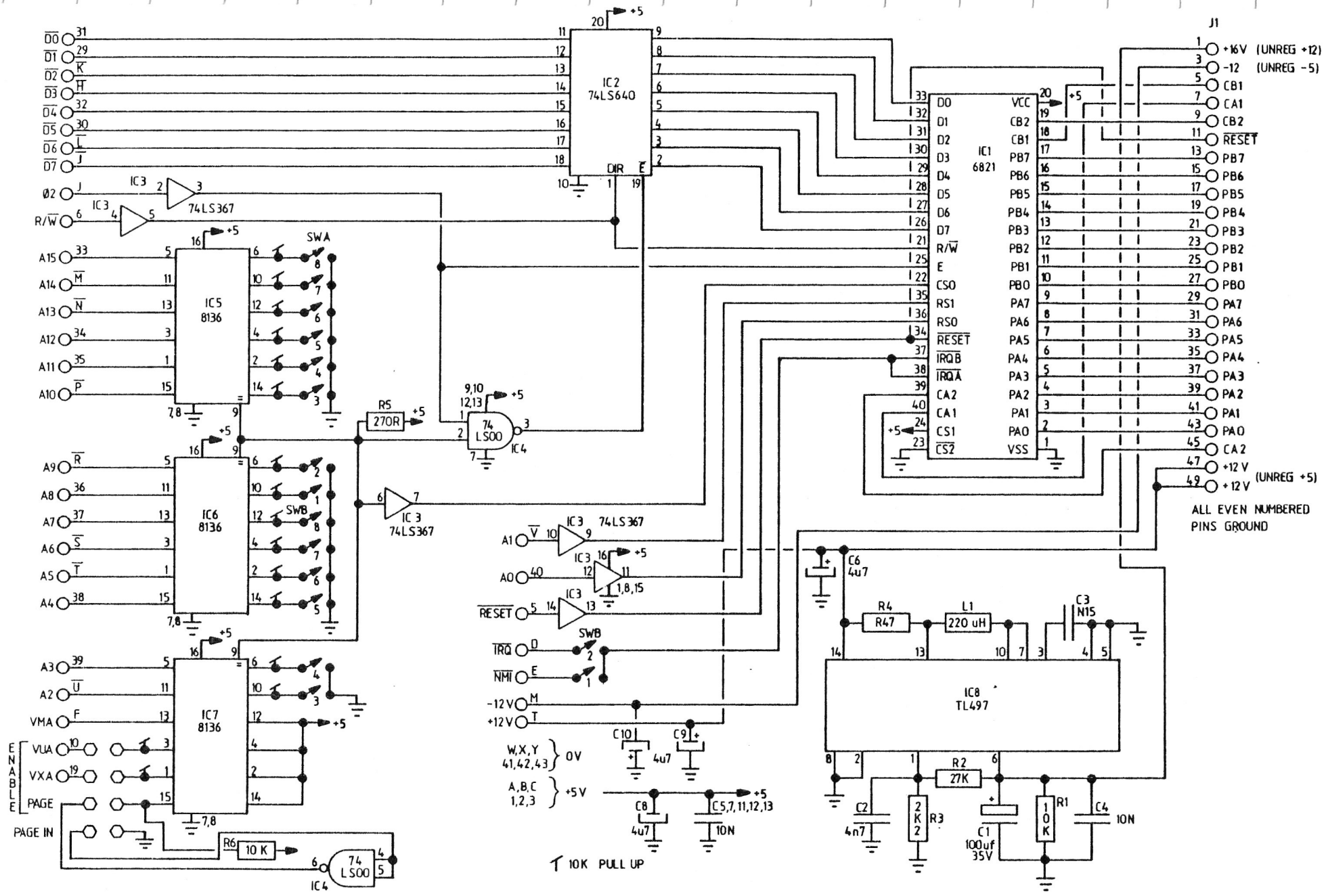



CONNECTIONS TO PROGRAMMER
BOARD VIA 50 WAY
TWIST-N-FLAT PLANAR CABLE

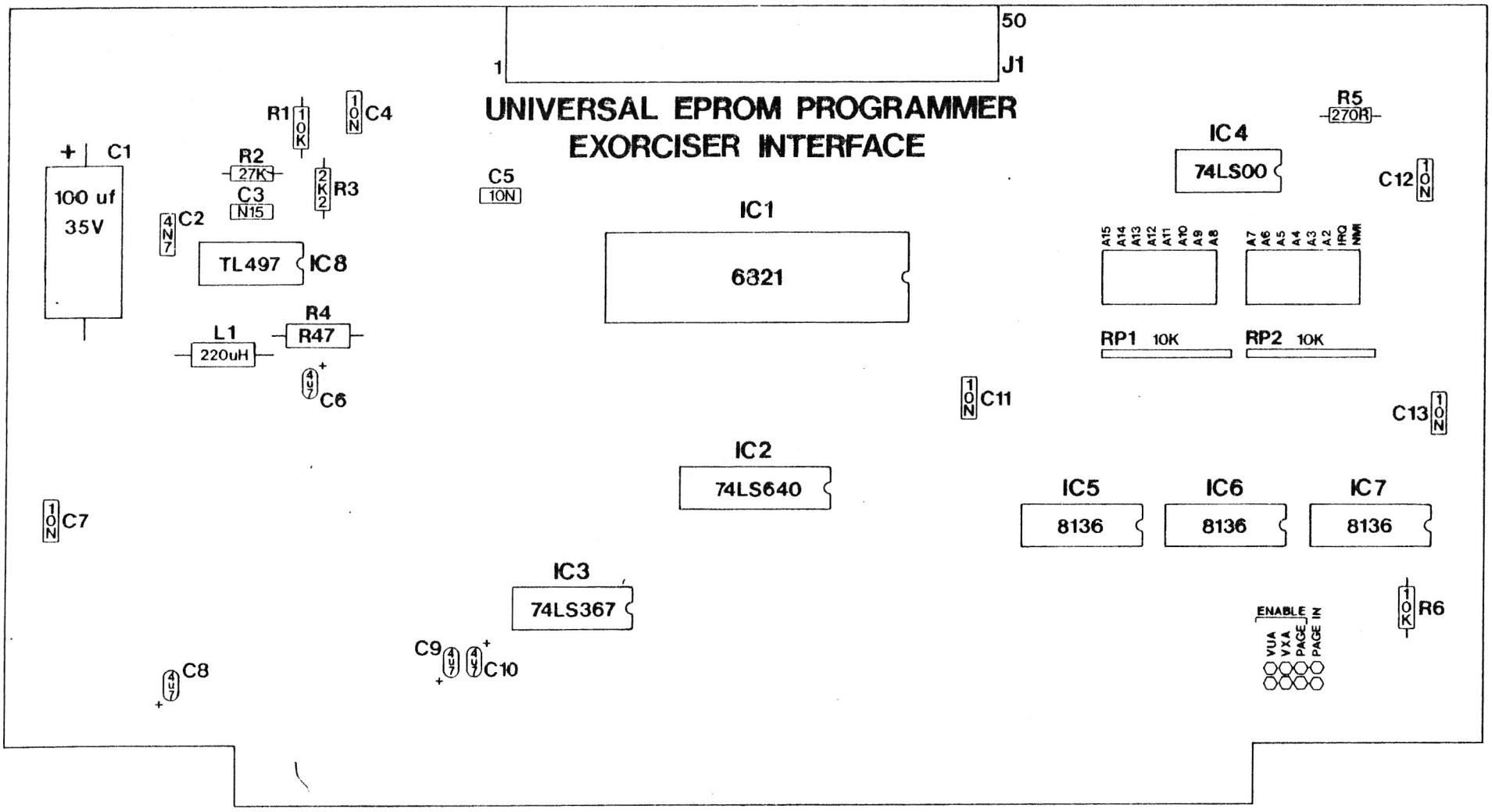
| REV | DATE | REV | DATE | DWN | | | |
|---|---|---|---|---|---|---|---|
| A | 13 OCT 81 | | | HE | UNIVERSAL EPROM PROGRAMMER, PARALLEL INTERFACE | WINDRUSH MICRO DESIGNS GAYMERS WAY, NORTH WALSHAM, NORFOLK. | EPROM PROGRAMMER SHEET 1 OF 2 |

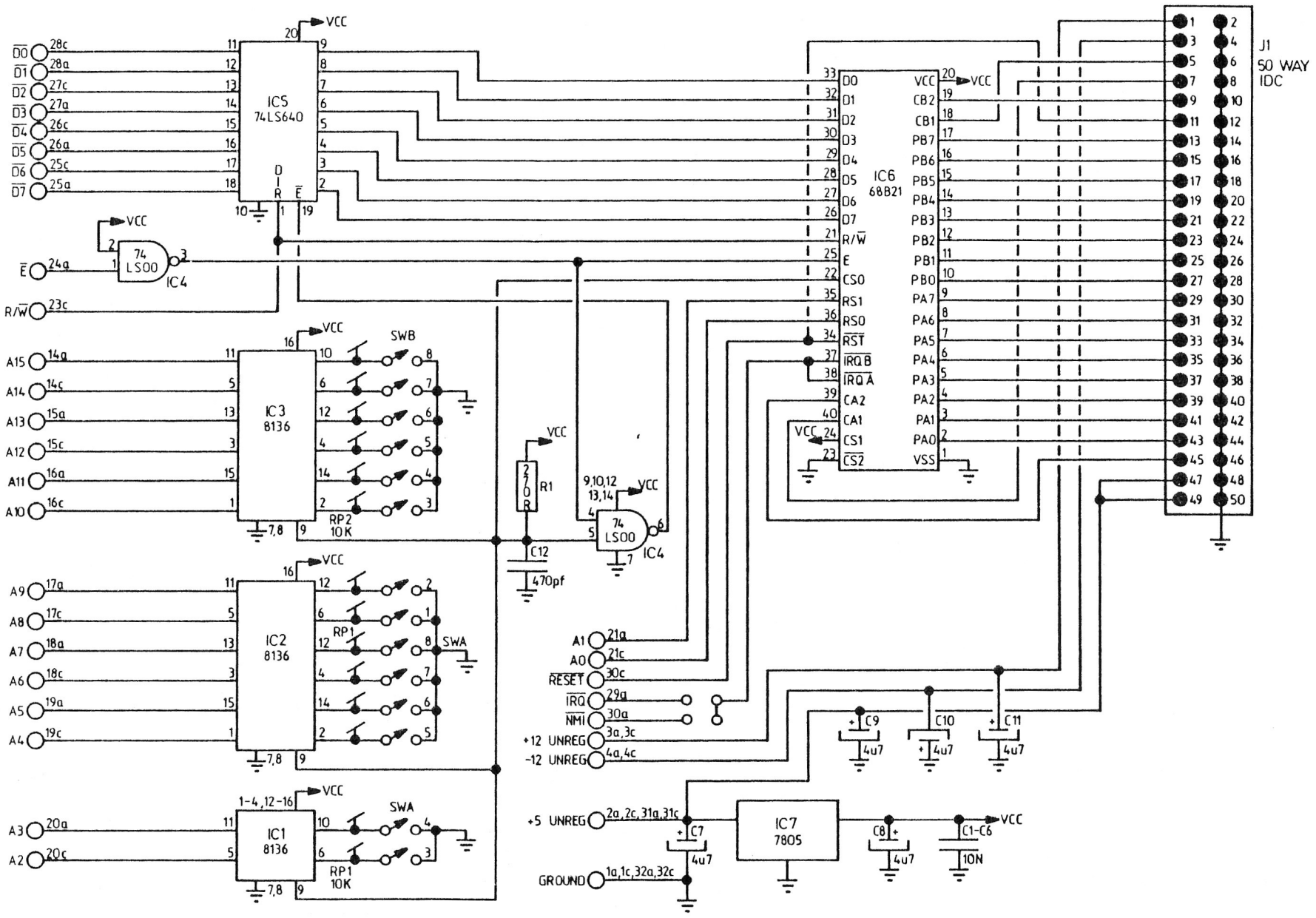



| | REV | DATE | REV | DATE | DWN | | | |
|---|---|---|---|---|---|---|---|---|
| | A | 7 OCT 81 | | | HE | COMPONENT OVERLAY FOR UNIVERSAL EPROM PROGRAMMER PARALLEL INTERFACE | WINDRUSH MICRO DESIGNS GAYMERS WAY, NORTH WALSHAM, NORFOLK. | EPROM PROGRAMMER SHEET 2 OF 2 |

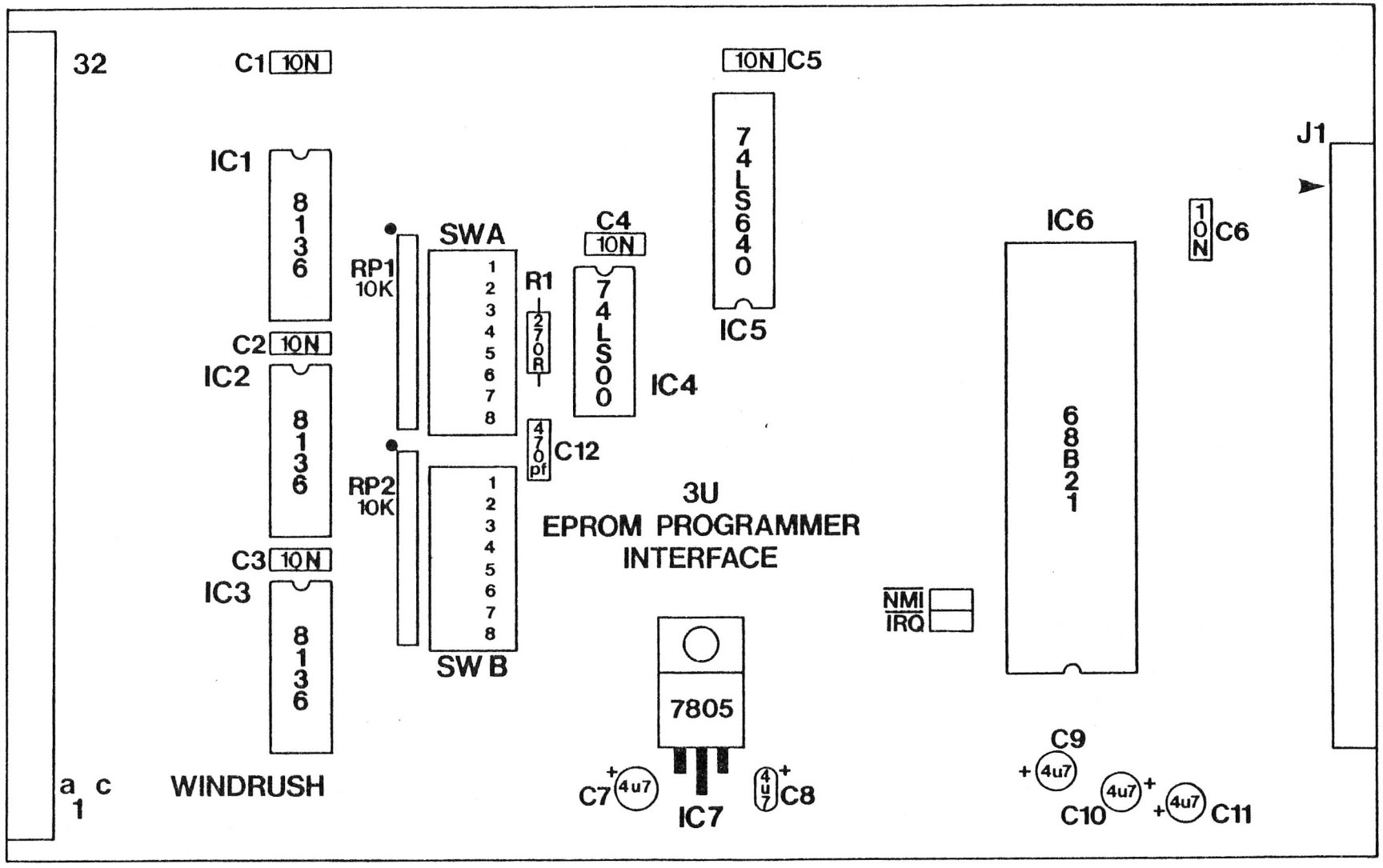



| REV | DATE | REV | DATE | DWN | UNIVERSAL EPROM PROGRAMMER EXORCISER INTERFACE BOARD | WINDRUSH MICRO SYSTEMS WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK. | EXORFACE SHEET 1 OF 2 |
|---|---|---|---|---|---|---|---|
| A | 23 JUNE 82 | | | HE | | | |


UNIVERSAL EPROM PROGRAMMER
EXORCISER INTERFACE
1
50
J1
+
C1
100 uf
35V
R1
10K
10N
C4
R2
27K
C3
N15
2K2
R3
4N7
C2
TL497
IC8
L1
220uH
R4
R47
4u7
C6
C5
10N
IC1
6821
R5
270R
IC4
74LS00
C12
10N
A15 A14 A13 A12 A11 A10 A9 A8
A7 A6 A5 A4 A3 A2 IRQ NMI
RP1 10K
RP2 10K
10N
C11
C13
10N
IC2
74LS640
IC5
8136
IC6
8136
IC7
8136
10N
C7
IC3
74LS367
C9
4u7
4u7
C10
4u7
C8
10K
R6
ENABLE
VUA
VXA
PAGE
PAGE IN




| REV | DATE | | | DWN | 3U UNIVERSAL EPROM PROGRAMMER INTERFACE | WINDRUSH MICRO SYSTEMS<br>WORSTEAD LABORATORIES,<br>NORTH WALSHAM, NORFOLK. | 3U UPRM<br><br>SHEET 1 OF 2 |
|---|---|---|---|---|---|---|---|
| A | 12 DEC 83 | | | HE | | | |
| | | | | | | | |
| | | | | | | | |




| | REV | DATE | | | DWN | COMPONENT OVERLAY | WINDRUSH MICRO SYSTEMS<br>WORSTEAD LABORATORIES,<br>NORTH WALSHAM, NORFOLK. | 3U UPRM<br><br>SHEET 2 OF 2 |
|---|---|---|---|---|---|---|---|---|
| | A | 30 NOV 83 | | | HE | | | |
| | | | | | | | | |
| | | | | | | | | |

SCHEMATIC DIAGRAMS FOR THE MAIN PROGRAMMER POD ARE NOT AVAILABLE AT THIS TIME

We are currently processing a 'REGISTERED DESIGN' application for the hardware design of the POD. Until this application is registered we will not release detailed design information.