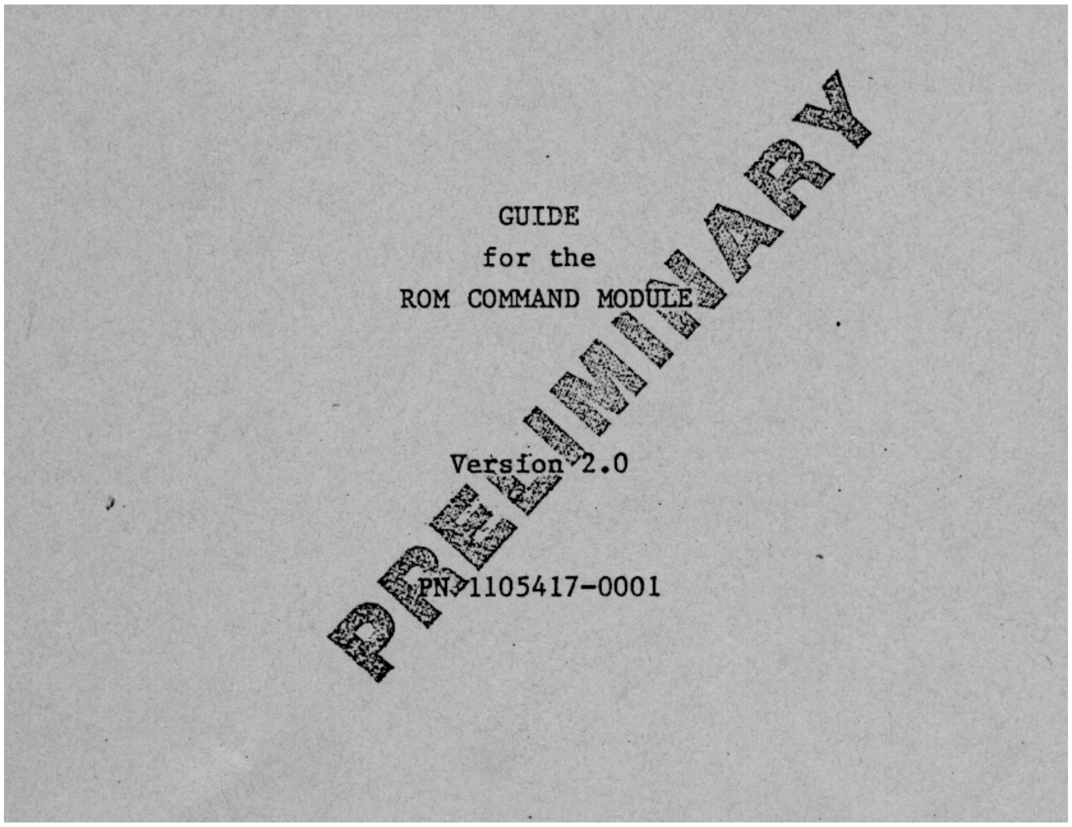


GUIDE

for the

ROM COMMAND MODULE

Version 2.0

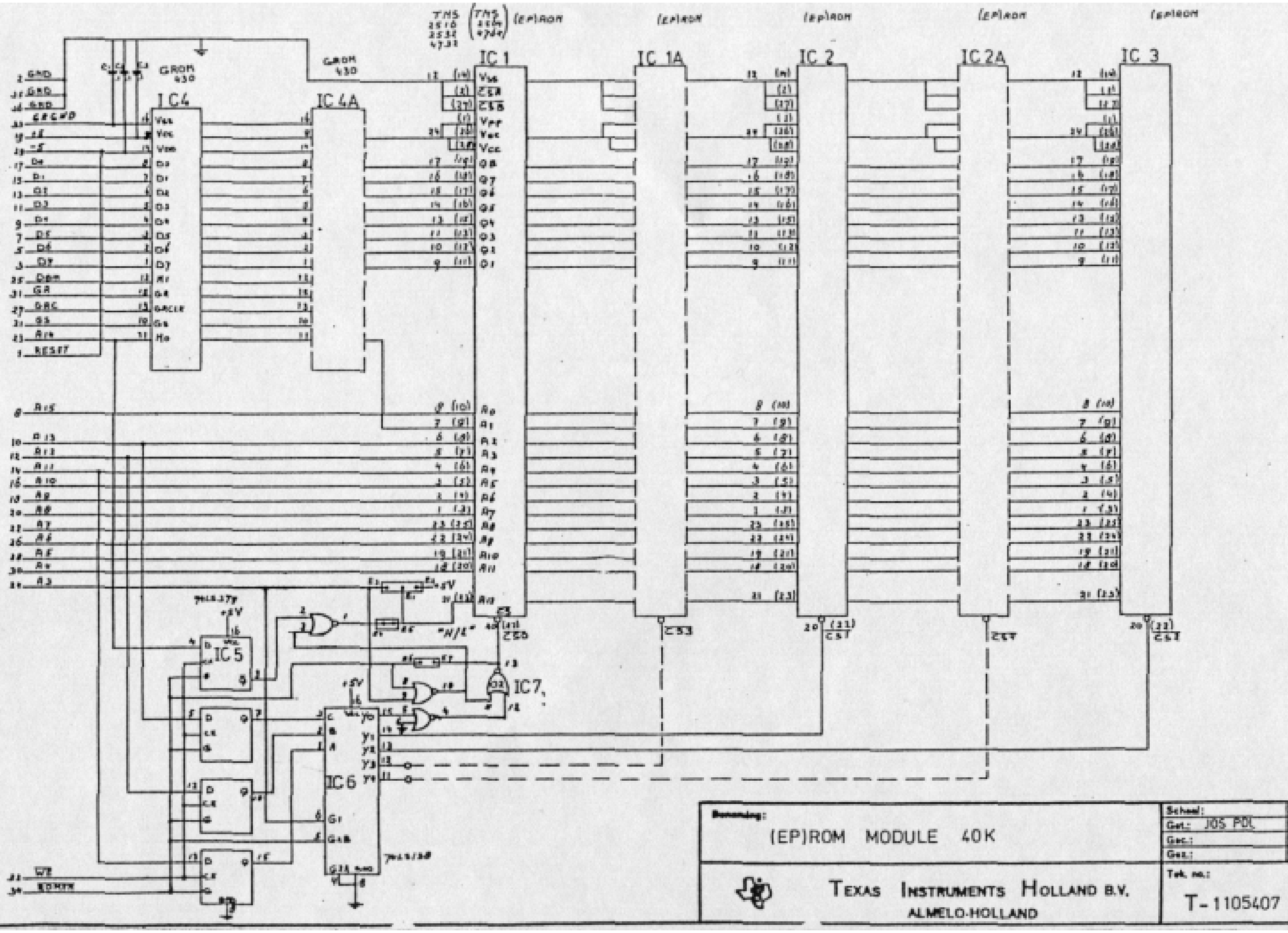
PN 1105417-0001

PRELIMINARY

European consumer division

Date: February 8, 1983

TI Almelo Holland



Benaming: (EP)ROM MODULE 40K

TEXAS INSTRUMENTS HOLLAND B.V.
ALMELO-HOLLAND

School:	
Get.:	JOS POL
Gec.:	
Goz.:	
Tek. no.:	T-1105407

SECTION 1

INTRODUCTION

1.1 PURPOSE

This manual provides a complete description of how Assembly Language User Programs need to be written so that the object code can be downloaded into (EP)ROM's which can then be used in the "(EP)ROM module", a module designed to be used with the TI 99/4A Home Computer. Such a module consists of one or more EPROM's and a GROM. The GROM is a special 6K bytes ROM device which is necessary to let the home computer access your Command Module. The modules can contain either subroutines or complete User Programs whose name will appear on the introductory menu. The modules are created using 9900 Assembly language. This manual is an addendum to the Assembly Language manual which belongs to the "TI EDITOR/ASSEMBLER" package.

1.2 AUDIENCE

This manual is written for programmers who want to make their own Command Modules in Assembly Language, containing Assembly Language subroutines or complete User Programs. This manual assumes that the programmer already understands the "TI EDITOR/ASSEMBLER" package.

SECTION 2

PROGRAM ORGANIZATION

2.1 INTRODUCTION

Making an Assembly Language Program for the "ROM MODULE" can be done in two ways; by converting an existing Assembly Language Program made with the "TI EDITOR/ASSEMBLER" or by starting programming with the special requirements of the "ROM MODULE" in mind. For both, the programmer needs to fulfill the requirements and accept the limitations of Assembly Language structures in a "ROM MODULE".

This section shows how to set up an Assembly Language program that is meant for a "ROM MODULE". When a programmer understands the material in this section, converting an existing Assembly Language program should be a simple task.

2.2 DEVELOPMENT AND PRODUCTION

To develop a ROM Command Module a programmer needs at least a system which consists of:

1. TI-99/4A console
2. Disk system
3. Memory expansion unit

either

4. "TI EDITOR/ASSEMBLER' module
5. Utilities diskette

or

4. "DEVELOPERS ROM MODULE"

5. Developers Utilities diskette

NOTE

The "DEVELOPERS ROM MODULE" has been designed specifically for the development of software for a "ROM MODULE" and contains the GROM IC from the "TI EDITOR/ASSEMBLER" module. The developer has not only all the facilities of the "TI EDITOR/ASSEMBLER", he can also download completed subroutines, program parts or text into EPROM making them easily accessible to the rest of the program under development.

There are two possible methods of testing the software

- Loading the Assembly Language programs into the Memory expansion unit together with the Utilities provided on the Utilities diskette.

- Write the Assembly Language programs using references to the Utilities pre-programmed with the "DEVELOPERS ROM MODULE" and the completed subroutines, program parts or text already downloaded into the EPROM's of the "ROM MODULE", loading the Assembly Language program part under test into the Memory expansion unit.

A "ROM MODULE" consists always of one or more EPROM's and one GROM. Such a GROM is necessary to let the home computer access the programs or routines in this module. A GROM consists of 6K bytes which must be used by the programmer for the 'ROM header' and 'PROGRAM headers' (See section 4). The rest of the 6K GROM memory can be used for data such as: text, character definitions and branch tables. A GROM can not be simulated by the programmer and should be designed with the help of the 'Texas Instruments third party assistance team'.

For eventual production of the "ROM MODULE", the software must be split up in a part for the GROM (headers and data) and for the EPROM's (Assembly Language Code and data). The EPROM part must be reassembled and downloaded into the EPROM's, PROM's or ROM's, together with the object code for the Utilities referenced by the programs.

The basic system, to run the "ROM MODULE" is:

- TI-99/4A console

- “ROM MODULE”

2.3 ROM MODULE LIMITATIONS

Some special attention should be paid to the structure of the Assembly Language programs meant for a "ROM MODULE". The most important limitations of the code for a module with respect to "TI EDITOR/ASSEMBLER" are:

* Only one 4K ROM is fixed, the other part of the Assembly Language program most be structured as pages of 4K each.

* When using the "ROM MODULE", the Memory Expansion unit need not be connected. This has the following results:

  - When no Memory Expansion unit is connected, the CPU RAM memory available is that resident in the console i.e. the directly accessible CPU RAM is 256 bytes. (NOTE: The VDP RAM available is, as usual, 16K bytes).

  - Due to lack of directly accessible RAM, BLWP statements should be avoided.

  - If a programmer needs more than 256 bytes of directly accessible RAM, it is possible to store part of the CPU RAM temporarily in the VDP RAM, and then restore it when required.

  - Assembly Language programs running with the "TI EDITOR/ASSEMBLER" often use Utilities which are loaded into the Memory expansion unit from a Utilities diskette. Some of these utilities have been adapted for inclusion in the "ROM MODULE". For further information, see section 5.

* A special header block is required for all "ROM MODULES" (see section 4).

2.4 DEVELOPMENT STRATEGY

An Assembly Language program which is larger than 8K must be programmed in pages. When paging is necessary, only one page of 4K is fixed and the others can be selected one at a time.

Development should begin with the preparation of a programming specification including the definition of the screens, the total amount of ROM needed, the Utilities needed, RAM usage, and so on. Then a ROM memory map should be created using the following procedure.

* Firstly decide which routines must be placed in the fixed 4K ROM.

  - The header and main program must be in the fixed ROM (see section 4).

  - Any subroutines placed in the fixed 4K ROM can always be accessed from any other page without changing the page select. Therefore, wherever possible, all subroutines which are used in more then one page should be implemented in the fixed ROM.

  - For the same reason, the Utilities should also be implemented (See section 5) in the fixed ROM.

  - When a routine in the fixed 4K is accessed, it is possible to return to the calling page without any page selecting, unless the routine in the fixed 4K has used a subroutine on another page than the calling one. In this case, the calling page needs to be reselected before returning from the routine. Calling subroutines on non-fixed pages from the fixed ROM page should be avoided wherever possible.

- It is a good habit to put all text used in the "ROM MODULE", together in the fixed 4K ROM. In this case, translating the program into another language will only need a change of one (EP)ROM.

* Secondly, allocate the remaining routines to the other pages.

When the programming specification is complete, the programming of the first routines can be started. Assembly Language routines can be written and debugged using the Memory Expansion Unit and either the "DEVELOPERS ROM MODULE" or the "TI EDITOR/ASSEMBLER" module. Using the "DEVELOPERS ROM MODULE", each time a subroutine is completed it can be put into EPROM. This eliminates the need to load these tested routines into the Memory Expansion Unit.

When all subroutines are finalized and tested, the main program and ROM header should be written and put in the fixed page to complete the "ROM MODULE" development (see section 4).

2.5 EPROM PROGRAMMING

After a subroutine or program part is finalized it can be put in EPROM. Several types of EPROM's that can be used are listed in the APPENDIX of this manual. Other EPROM's can be used, but before doing so, it is essential to check very carefully how the jumpers should be placed and whether they are fully compatible with the device.

To program the EPROM's any EPROM programmer can be used. A small BASIC program needs to be written to transmit the object file from the diskette to the EPROM programmer by means of a RS232 peripheral (sold separately). Object files can not be linked, therefore the object file, which has to be put in EPROM, may not contain any REF statement.

The BASIC program also must format the tagged object code so that it can be used for the particular EPROM programmer. Of course BASIC is not the only language which can perform this transfer. The routines can also be written in Assembly Language or PASCAL.

When 64K EPROM's are used, special attention should be paid to the fact that two pages will be programmed in one EPROM. The first page starts at EPROM address >0000 the other at >1000. The memory addresses in the "ROM MODULE" are for the fixed page >6000 through >6FFF and, even when there are two pages in one 64K EPROM, for all other pages, >7000 through >7FFF (See section 3).

SECTION 3

MEMORY ORGANIZATION

3.1 INTRODUCTION

To understand memory organization, some knowledge of the basic terms and how they apply to the TI Home Computer is necessary. They are all described in the appendices of the "TI EDITOR/ASSEMBLER" manual.

The Home Computer has three different types of memory organization. These are CPU RAM or ROM, VDP RAM and GROM. The "ROM MODULE" consists of only CPU ROM so the other types of memory are not discussed in this section.

3.2 DIRECTLY ADDRESSABLE MEMORY

When all possible devices are connected, 64K bytes (65,536) of memory are directly addressable.

Addresses >0000 through >1FFF are built into the console. They contain 8K bytes of ROM that contain the TI BASIC language and other information necessary to the functioning of the computer.

Addresses >2000 through >3FFF are the 8K bytes of RAM that make up the low memory of the Memory Expansion Unit. They can only be used when the Memory Expansion Unit is connected.

Addresses >4000 through >5FFF are built into various peripherals. They contain up to 8K bytes of ROM for the Device Service Routine (DSR) used to run peripheral devices, such as disk drives or a printer. These ROM's are selected by CRU operations (see "TI EDITOR/ASSEMBLER" manual), so several ROM's can be at the same address.

Addresses >6000 through >7FFF are available on the Command Module port. Some Command Modules, for example "TI EXTENDED BASIC", have ROM in this space. This area is also available for the "ROM MODULE" and will be more extensively discussed in the next paragraph.

Addresses >8000 through >9FFF are built into the console. They contain RAM from addresses >8300 through >83FF and all of the memory-mapped device locations.

Addresses >A000 through >FFFF are the 24K bytes of RAM that make up the high memory of the Memory Expansion Unit. They can only be used when the Memory Expansion Unit is connected.

The following memory map summarizes the above information.

CPU Memory Use
General Case

>0000	(Console ROM) Two 4K ROM Chips
>2000	Low Memory Expansion
>4000	Peripheral ROM’s (mapped) For Device Service Routines
>6000	Application ROM’s in Command Module Also the “ROM MODULE”
>8000	Memory-mapped devices for VDP, GROM, Sound and Speech And on-board RAM at >8300->83FF
>A000	High memory Expansion

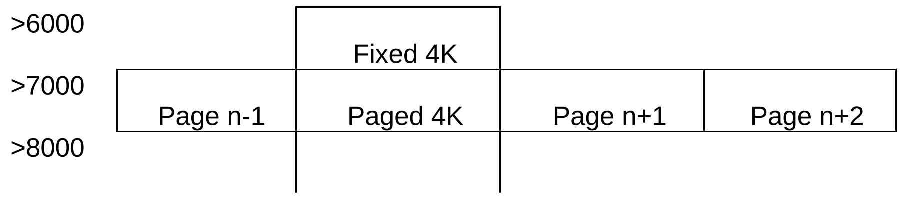
3.3 ROM MODULE MEMORY

As described above, there are >2000 bytes available for ROM in a Command Module. By mans of 'paging', however it is possible to increase the directly accessible ROM available in this area.

The memory map of the "ROM MODULE" is as follows.

- >6000 - >6FFF Fixed 4K (EP)ROM area (not pageable).
- >7000 - >7FFF Pageable 4K (EP)ROM. (Depending on the size of the (EP)ROM it can have one or two pages per device).

ROM MODULE use
Memory map of the Command module area



3.4 PAGE SELECT

The "ROM MODULE" has a capacity of 4K which is fixed and a maximum of 9 pages of 4K which are selectable. The pages must be selected by making a copy of its own data at the following addresses.

Page No.	Address	EPROM No.	Remarks
0		IC1	Fixed, needs not to be selected
1	>7002	IC1	Only available with 64K (EP)ROM
2	>7004	IC1A	Stacked on IC1
3	>7006	IC1A	Stacked on IC1 Only available with 64K (EP)ROM
4	>7008	IC2	
5	>700A	IC2	Only available with 64K (EP)ROM
6	>700C	IC2B	Stacked on IC2
7	>700E	IC2B	Stacked on IC2 Only available with 64K (EP)ROM
8	>7010	IC3	
9	>7012	IC3	Only available with 64K (EP)ROM

When no stacking is wanted, the pages 0, 1, 4, 5, 9, and 9 can be used. If 32K (EP)ROM's are used and no stacking is wanted, the pages 0, 4 and 8 are available.

A proper way of changing from one page to another is displayed by the following assembler instruction sequence.

Example to select page 8:

```
MOVB    @>7010,@>7010     Select page 8
```

or:

```
LI      R14,>7010           Get page select address
MOVB    *R14,*R14           Select page 8
```

or:

```
MOVB    @>7010,R1           Get info from the page select address
MOVB    R1,@>7010           Select page 8
```

In principle all these instructions have the same result. They select a page by making a copy of the data at the page select address to itself. As long as this rule is fulfilled any sequence is permitted, if it is not, the Home Computer can go into an undefined state and can even damage the hardware of the Home Computer or the Command Module.

SECTION 4

GROM HEADER AND ROM ACCESS

4.1 INTRODUCTION

When the "ROM MODULE" contains programs that may be accessed by other programs (not in the module) it needs a 'GROM header'. This 'GROM header' provides information that enables the Home Computer monitor to find programs when the "ROM MODULE" is plugged in and the system is initialized. The 'GROM header' has a "ROM MODULE" identification byte which distinguishes a valid one from an empty slot. The 'GROM header' also provides information enabling the monitor to search for a program of a particular name and type. There are several types of programs possible in the Home Computer system such as: User Application, Device Service, Subroutine Links and Interrupt Service programs, but only the User Application programs and power-up routine for the "ROM MODULE" will be described in this manual. For every type of program there is a chained list of 'PROGRAM headers'. The first 'PROGRAM header' of each type is pointed to by an entry in the header. The 'GROM header' must be located at GROM address >6000. Figure 4.1 shows a 'GROM header'. A 'PROGRAM header' must always start on a word boundary (This is important to look at when a previous 'PROGRAM header' has an odd number of bytes. Figure 4.2 shows a program header.

Besides a ‘GROM header' this GROM must have special statements, which perform the access to the Assembly Language code in the EPROM's.

Address	Size	Contents
>6000	byte	>AA, valid GROM identification
>6001	byte	>00 = version number
>6002	byte	>00 = number of program
>6003	byte	>00 = reserved
>6004	word	address of power-up routine header
>6006	word	address of first User Program header
>6008	word	>0000, reserved
>600A	word	>0000, reserved
>600C	word	>0000, reserved
>600E	word	>0000, reserved

Figure 4-1 GROM header

NOTE

The address of any program types should be >0000 in the 'GROM header' if there are no programs of that type. The, number of programs and version number are not currently being used but should be used for future expansion.

Size	Contents
word	pointer of next program header of the same program type (>0000 if end of list)
word	entry address of program
byte	number of characters in program (N)
N. byte	ASCII character representation of program name

Figure 4-2 PROGRAM header

4.2 USER APPLICATION PROGRAM

A User Application program is a main program which resides in the "ROM MODULE" and whose name is displayed in the menu which appears immediately after the power-up screen. User Application programs may call Device Service Routines, Subroutine Links and GROM programs.

NOTE: The access to GROM programs is not yet implemented.

4.3 ROM ACCESS

A special instruction must be defined in the GROM, which performs as a ROM access. In fact this is the only instruction in GROM to which the program entry pointer in the program header is pointing. The code sequence is:

BYTE >05,XY

To link to a routine in ROM, the instruction which is equal to >05 is followed by a byte that specifies to a table and an entry. The first nybble of this byte is from 0 through >F, indicating the wanted table (See table 4.1).

Table 4-1 ROM access nybble table

Table #	Function	Address
0	Floating point routines	“FLTTAB”
1	Conversion and TI BASIC routines	“XTAB”
2	Memory Expansion unit	>2000
3	TI BASIC enhancement	>3FC0
4	TI BASIC enhancement	>3FE0
5	Peripheral ROM	>4010
6	Peripheral ROM	>4030
7	ROM in Command Module	>6010
8	ROM in Command Module	>6030
9	ROM in Command Module	>7000
>A	Memory Expansion unit	>8000
>B	Memory Expansion unit	>A000
>C	Memory Expansion unit	>B000
>D	Memory Expansion unit	>C000
>E	Memory Expansion unit	>D000
>F	Scratch pad RAM	>8300

NOTE

The nybbles from 7 through 9 are of a special interest, because they refer to the table entry address, which will be used for the "ROM MODULE".

The second nybble of the byte is from 0 through >F. When doubled it indicates the offset from the beginning of the table, from >00 through >1E. As an example of this instruction: >05,>24 causes a branch to the address contained in the fifth entry of table 2.

4.4 INITIAL STATE

Upon entry to a ROM routine, the workspace pointer is set to >83E0 and the status register is set to an unknown value. The interrupt is disabled and must be turned on when interrupts are allowed, e.g., VDP interrupt for sound and sprites.

The monitor will start every User Application program with all CPU RAM in the TI-99/4A in a defined state. CPU RAM will be zeroed except for >8370 through >8381. The word on the location >8370 contains the highest user definable address in VDP RAM. Location >8372 will contain >9F indicating that >8300 + >9F is the data stack pointer. Location >8373 will contain >7E indicating that >8300 + >7E is the subroutine stack pointer. Location >8374 is zero. The other locations (>8375 to >8381) have undefined values.

VDP RAM will have the 7 x 8 large case character set loaded. The VDP registers will be Filled with its default values (see table 4.1). The screen will be blanked and the color table will contain all >17. All the rest of the VDP RAM will be zeroes.

Table 4-2 Default values for VDP registers

Reg number	Value	Meaning
0	>00	No bit map mode nor external video
1	>E0	Set VDP for 16K memory, turns screen on, enables VDP interrupt, puts VDP in normal pattern mode with size 0, and unmagnified sprites
2	>00	Screen Image Table base address = >0000
3	>0E	Color Table base address = >0380
4	>01	Pattern Descriptor Table base address = >0800
5	>06	Sprite Attribute List base address = >0300
6	>00	Sprite Descriptor Table base address = >0400
7	>F5	Makes text color white, backdrop light blue

4.5 POWER-UP ROUTINES

The monitor initializes the system by calling power-up routines. Power-up routines are executed whenever the system is reset by either hardware or software. The console power-up routine executes first. This routine puts up the initial screen and menu and calls the selected program. Next, the monitor searches peripheral ROM and then GROM headers for power-up routine addresses and executes them as it finds them. After each power-up routine is executed, a search is made for the next one. When there are no more power-up routines found, the selected program is started with the system initialized as described in the last paragraph.

GROM power-up routines-can only call one level of ROM programs as a DSR or subroutine link. When power-up routines are executed, they should return to the monitor by executing a GROM Programming Language instruction 'RTN' (equals >00).

Power-up routines can use CPU RAM >04 to >71 for whatever is necessary. The complete VDP RAM may be used. The data or subroutine stack pointers may never been changed.

SECTION 5

UTILITIES AND PREDEFINED SYMBOLS

5.1 AVAILABILITY

Several Utilities are provided in the "TI EDITOR/ASSEMBLER" to give the programmer access to many of the resources of the TI Home Computer. Also predefined symbols are available in this package. The Utilities and predefined symbols are loaded at the same time as the Loader. The programmer can make them available by mentioning them in a REF statement at the beginning of the Assembly Language program.

Using the "ROM MODULE" these Utilities and predefined symbols are not available unless they are defined by the programmer himself when he puts the program in a "ROM MODULE”. For software developers the source files of adapted Utilities are included on the Developers Utility diskette (except "GPLLNK" and "LOADER"). Some special attention should be taken when defining the Utilities, because as they were defined in "TI EDITOR/ASSEMBLER" using BLWP constructions, they needed their own workspace area, which of course needs directly accessible memory space. This so called CPU RAM is very limited in the TI Home Computer used without the Memory Expansion, therefore BLWP constructions should be avoided. For the adapted Utilities, the BLWP statement is changed into BL. This means that the user must, unless mentioned otherwise, use the Utility routines with its workspace pointer equal to the GPL or DSR workspace pointer (= >83E0). Also the following registers of this workspace area are reserved for special purposes and should not be changed without a special reason.

- R12 CRU base address of a peripheral or main console
- R13 Reserved for GROM read data pointer
- R14 Register with system information
- R15 VDP write address pointer (= >8C02)

NOTE

The programmer must always keep in mind, that when a routine is accessed by means of a "BLWP" statement, the registers R13, R14 and R15 are destroyed. This means also, that when these registers are changed a "RTWP" return is not allowed.

During the development of the software, Utilities can be accessed either by including the Utility source files in the source file of the program, or by loading the Utility object code into an EPROM and including references to them in the program source file.

5.2 VDP UTILITIES

All the VDP Utilities provide access to Video Display Processor RAM. All parameters are passed through the Workspace Registers. These Utilities are accessible with a BL statement. The registers used are the same as the corresponding routines in the "TI EDITOR/ASSEMBLER"; but the names of the Utilities have been changed from the original as follows:

- VSBWRM VDP RAM Single Byte Write, ROM MODULE

- VMBWRM	VDP RAM Multiple Byte Write, ROM MODULE
- VSBRRM	VDP RAM Single Byte Read, ROM MODULE
- VMBRRM	VDP RAM Multiple Byte Read, ROM MODULE
- VWTRRM	VDP RAM Write Register, ROM MODULE

Example: To write one byte to the VDP RAM:

```
        REF   VSBWRM, . . .
VADDR   EQU   >0380
        . . . . .
        . . . . .
        LI    R0,VADDR          Load VDP RAM address
        LI    R1,>3500          Load byte to be written
        BL    @VSBWRM           Call routine
        . . . . .
        . . . . .
```

This loads the value >35 into the VDP RAM at address >038

Workspace: It is not necessary to access the VDP Utilities with the workspace pointer equal to the GPL workspace pointer (= >83E0). Nor need registers R13, R14 or R15 been filled with their specific values.

The source file is available on the Developers Utility diskette and can be accessed with the name "DSKx.VDPRM/SRC" (x is the number of the drive with the Development Utility diskette, 1...3).

NOTE

The total 16K bytes of VDP RAM is not always available, e.g., a disk system reserves some VDP memory. Overwriting of data in the VDP RAM can be avoided by restricting the use of VDP RAM to addresses below the top of memory address value which is stored in >8370. Reserved VDP RAM is always at addresses higher than the top of memory given in >8370, unless, of course, the programmer himself has altered the value in >8370.

5.3 EXTENDED UTILITIES

The "TI EDITOR/ASSEMBLER" had five Utilities, called Extended Utilities. They allowed the programmer to access the routines built into the Home Computer. Two of these routines "XMLLNK" and "DSRLNK" are also in an adapted version available for the "ROM MODULE". "GPLLNK" and "LOADER" are not implemented and "KSCAN" must be substituted by a BL statement to address >000E. All these Extended Utilities will be discussed in the following paragraphs.

5.3.1 KSCAN.

The KSCAN routine allowed the user to access the keyboard scanning routine preprogrammed in the Home Computer. This routine is not available for the "ROM MODULE". A substitute procedure follows which accesses the key scan routine directly with a BL statement to address >000E. However, before using this address the programmer must supply the GPL workspace pointer.

To implement a keyboard scanning routine the following locations should be loaded with:

- The proper keyboard number in >8374
- The GROM Read Data address (normally >8900) in register 13 of the calling workspace
- The VDP Write Address (>8C02) in register 15 of the calling workspace
- CPU RAM >8373 with a one-byte pointer into CPU RAM, i.e., the least significant byte of a CPU RAM address. This pointer is the GPL subroutine stack pointer. The scan routine pushes the current GROM address (2 bytes) on this stack, then pops it off after the scan. The stack is a pre-incrementing one.

- CPU RAM >83C6 through >83CA with the information stored there by the previous keyboard scan. This is keyboard state and debounce information and must be maintained.

- CPU RAM >83D4 with the current value of VDP register 1

Execution of the scan routine modifies the following CPU RAM locations:

- The word located two bytes higher than the address indicated by the pointer in CPU RAM >8373

- >8374 Keyboard number; reset from 3, 4, or 5 to 0

- >8375 Returned keycode

- >8376 Y joystick parameter; modified when keyboard number 1 or 2 is scanned

- >8377 X joystick parameter; modified when keyboard number 1 or 2 is scanned

- >837C Key status; cleared for old keys; >20 for new keys.

- >83C6 through >83CA - Debounce and internal flags; these values must be maintained between scans for the routine to function properly

- >83D6 and >83D7 - Screen timeout; cleared after each new key

- >83D8 and >83D9 - Save return address during scan routine
- All registers of the GPL workspace (pointer is >83E0) are used

An example of key board scan routine is shown and explained in Utility section of the "TI EDITOR/ASSEMBLER" manual.

5.3.2 GPLLNK. "GPLLNK" allows the programmer to use routines written in Graphics Programming Language. "GPLLNK" for the "ROM MODULE" has not been implemented.

5.3.3 XMLLNK.

The "XMLLNK" Utility allows the programmer to link an Assembly Language program to a routine in ROM or to branch to a routine in the Memory Expansion unit. The ROM routines perform such tasks as floating point arithmetic, stack arithmetic, string to number conversions and so on.

The "XMLLNK" routine is also adapted and is now accessible by a BL statement. The access name is changed to "XMLRM". The amended routine uses the same workspace and can destroy its registers. Of course it also destroys the memory words which are needed by an actual XML-routine which is called.

The additional data that needs to be provided to get a link to a specific routine is the same as for “XMLLNK.” Not all ROM routines are accessible with "XMLRM" (see "CIFRM" below)

Example: To call the routine to convert a floating point value into an integer.

```
REF     XMLRM
. . . . .
. . . . .
BL      @XMLRM              Call routine
DATA    >1200               Convert floating to integer
. . . . .
. . . . .
```

This converts the floating point value stored in >834A, ... to an integer value, to be found at >834A

The source file is available on the Developers Utility diskette and can be accessed with the name "DSKx.XMLRM/SRC" (x is the number of the drive with the Utility diskette, 1 ... 3).

5.3.3.1. "CIFRM" Convert Integer to Floating Point.

The "Convert Integer to Floating Point" (CIF) routine can not be accessed with the "XMLRM" Extended Utility. Therefore an adapted version of this routine "CIFRM" is provided on the Developers Utility diskette which can be linked to your own Assembly Language program with a "BL" statement.

Example: using "CIFRM":

```
Input
        FAC    >834A             contains the one word integer value
                                 to be converted
Output
        FAC    >834A             contains the floating point result
```

Note The "CIFRM" routine uses the registers R0 through R6 of the workspace of the calling routine.

```
        REF    CIFRM, . . .
VALUE   DATA   433
        . . . . .
        . . . . .
        MOV   @VALUE,@FAC        Store integer 433 in FAC
        BL    @CIFRM             Call routine
                                 @FAC >834A now contains the
                                 values >41,>04,>21,>00, . . . .
```

The source file is available on the Developers Utility diskette and can be accessed with the name "DSKx.CIFRM/SRC" (x is the number of the drive with the Utility diskette, 1 . . . 3).

5.3.4 DSRLNK. "DSRLNK" links an Assembly Language program to any Device Service Routine (DSR) or subprogram in ROM. This routine is also available in an adapted form for the "ROM MODULE". It can be accessed by a BL statement. The access name is changed to "DSRLRM". Special attention should be given to the CPU RAM which is used by the "DSRLRM" routine.

CPU RAM which is used and may be destroyed are:

- >834A through >836D 36 bytes, standard scratch area
- >8373 Subroutine stack pointer
- >83C0 through >83DF Interrupt workspace
- >83E0 through >83FF The GPL/DSR workspace

For a correct operation the following rules should be obeyed:

- A proper Peripheral Access Block (PAB) needs to be set up in VDP RAM. The pointer to the length of the file descriptor, which starts in byte 10, should be stored in CPU RAM >8356. This word may be changed by the DSR Utility.
- The routine uses the single byte VDP read routine so this routine should be defined somewhere with its label "VSBRRM".
- Workspace pointer must always be >83E0
- Registers R13 through R15 may never be changed by the user
- In CPU RAM >8373 must be a one-byte pointer into CPU RAM, i.e., the least significant byte of a CPU RAM address. The "DSRLRM" routine pushes a temporary address (2 bytes) on this stack, then pops if off after the routine is completed. This stack is a pre-incrementing one.

When no DSR is found with the defined name this Utility will return to the calling routine with the equal bit set. But other errors will not be indicated with the equal bit, therefore the programmer has to check the three error bits in the flag byte of the Peripheral Access Block (PAB).

For example suppose that a DSR access is necessary and that two bytes TSAVE+2 and TSAVE+3 are available for temporary storage

```
        REF   DSRLRM,VSBRRM, . . .

TSAVE   EQU   >7E00           Save area is at >837E+2 and +3
GPLWS   EQU   >83E0           GPL workspace area
MYWS    EQU   >8300           Programmer's workspace area
PAB     EQU   >1000           PAB entry in VDP RAM
        . . . . .
        . . . . .             Set up a PAB in VDP
        . . . . .
        LI    R1,TSAVE        Load temporary save address minus 2
        MOVB  @R1LB,@STKPNT Move address to stack pointer
        LWPI  GPLWS           Load GPL workspace pointer
        BL    @DSRLRM         Call routine
        DATA  >0008           Provide type of program
        JEQ   DSRER0          Is such a DSR found ?
        LI    R0,PAB+1        Point to status byte
        BL    @VSBRRM         Use Utility to get byte
        SRL   R1,13           Remove all not essential bits
        JNE   DSRERX          If not 0 then error !
        LWPI  MYWS            Restore user workspace pointer

        . . .Continue here when no DSR errors are detected
```

```
DSRER0    EQU    $                   Error >00 exit
. . . . .
. . . . .
DSRERX    EQU    $                   Error >01. . . >07 exit
. . . . .
. . . . .
```

After this part of a program is executed a DSR is accessed and a certain DSR operation is performed. If no DSR is found with such a name the routine escapes the program via "DSRER0". If another DSR error occurs the escape will go via "DSRERX", else the user workspace will be reloaded and the normal execution can proceed.

Unlike "DSRLNK" in "TI EDITOR/ASSEMBLER" the routine will not pass back information to the Assembly Language program via the "UTLTAB" area.

The source file is available on the Developers Utility diskette and can be accessed with the name "DSKx.DSRLRM/SRC" (x is the number of the drive with the Utility diskette, 1. . . 3).

5.3.5 LOADER.

"LOADER” loads TMS9900 tagged object code such as the Assembler produces. This Utility has not been implemented in the "ROM MODULE".

5.4 PREDEFINED SYMBOLS

Unlike the "TI EDITOR/ASSEMBLER" there are no predefined symbols available which could be used in place of. addresses. The programmer must define these symbols in his Assembly Language program as soon as he needs them by means of "EQU" statements.

SECTION 6

DEVELOPERS MODULE

6.1 COMMAND MODULE

The Developers Command Module consists of a PCB with the following parts:

1. Select logic and jumpers
2. Two 28 pin EPROM sockets
3. Two EPROM's containing software
4. "TI EDITOR/ASSEMBLER" GROM IC

The GROM contains the "TI EDITOR/ASSEMBLER" and this name will appear on the introductory menu.

The two EPROM's contain the following software, as their names appear on the introductory menu:

3. TOMBSTONE CITY - (F)
4. TOMBSTONE CITY - (D)
5. TOMBSTONE CITY - (I)
6. TOMBSTONE CITY - (NL)
7. LINES AND CIRCLES
8. CIF/XML/VDP DEMO
9. DSR/VDP DEMO

6.2 DEMOPROGRAMS

To give the programmer some assistance for the coding of his program short demonstration programs are available. These demonstration programs are implemented in the "Development kit command module" and are accessible as User Programs of which the names will appear on the introductory menu. The names of the programs on this menu are:

8. CIF/XML/VDP DEMO

9. DSR/VDP DEMO

To give a good impression of the paging technique the subroutines in Assembly Language code are divided among 4 pages. The 4 source files corresponding to the pages are available on the Developers Utility diskette and can be accessed with the name "DSKx.PAGEy/SRC" (x is the number of the drive with the Utility diskette, 1. . .3 and y corresponds to the page number, 0. . .3).

During the execution of these demonstration programs the following text will be shown in the right top corner of the screen.

```
PAGE:

0123E
```

An arrow will always be displayed above one of the numbers of this text to indicate in which page the currently accessed Assembly-Language code is programmed.

"E" means a routine is accessed which is external to this Command Module.

Both demonstration programs will be discussed very briefly in the following paragraphs.

6.2.1 CIF/XML/VDP DEMO.

This program generates a screen on which it asks for a hexadecimal value. This Integer value is converted to a Floating Point value by means of the "CIF"-Utility. The "XML"-Utilities are used to perform a Floating Point multiplication of this value with "PI/2". The result of this operation is converted back to Integer by means of a "XML"-Utility and displayed on the screen.

Besides the paging technique this program is written to demonstrate how to access the following Utility routines:

CIF	Convert Integer value into a Floating Point value.
XML	EXecute Machine Language routines. This routine is accessed for two purposes. To multiply two Floating Point values and to convert a Floating Point value back into an Integer one.
VDP	VDP utilities are used to store and recall data from the VDP RAM.

6.2.2 DSR/VDP DEMO.

This program generates a screen on which it asks for a program file name. The size of this file is measured and displayed on the screen. The program file is accessed by means of the "DSR"-Utility.

Besides the paging technique this program is written to demonstrate how to access the following Utility routines:

DSR	Utility to access the "Device Service Routines".

VDP	VDP utilities are used to store and recall data from the VDP RAM.

SECTION 7

APPENDIX

7.1 COMPATIBILITY OF EPROMS AND ROMS WITH ROM MODULE

The 40K (EP)ROM module is designed to allow as many (EP)ROM types as possible; the following list although not exhaustive, covers those parts which are definitely compatible.

1. TMS 2516 16,384-bit single 5 volt EPROM
2. TMS 2532 32,768-bit single 5 volt EPROM
3. TMS 2564 65,536-bit single 5 volt EPROM
4. TMS 4732 32,768-bit single 5 volt ROM
5. TMS 4764 65,536-bit single 5 volt ROM

All these EPROM's are compatible, but have still some minor differences which should be taken care of. For this reason a table is constructed to compare all pins of the above devices. Because the TMS 2564 EPROM is compatible, but has 4 pins more, the numbering is slightly different. In the table 6.1 the pin numbers for the TMS 2564 are noted between two parentheses. An apostrophe before a signal name means that the signal needs to be logically inverted (NOT-function).

Table 7-1 ROM compatibility

Pin No.	2516	2532	2564		4732	4764	Module Signal
(1)			Vpp				+5 Volt
(2)			‘CSa				GND
1 (3)	A7	A7	A7		A7	A7	A8
2 (4)	A6	A6	A6		A6	A6	A9
3 (5)	A5	A5	A5		A5	A5	A10
4 (6)	A4	A4	A4		A4	A4	A11
5 (7)	A3	A3	A3		A3	A3	A12
6 (8)	A2	A2	A2		A2	A2	A13
7 (9)	A1	A1	A1		A1	A1	A14
8 (10)	A0	A0	A0		A0	A0	A15
9 (11)	Q1	Q1	Q1		Q1	Q1	D7
10 (12)	Q2	Q2	Q2		Q2	Q2	D6
11 (13)	Q3	Q3	Q3		Q3	Q3	D5
12 (14)	Vss	Vss	Vss		Vss	Vss	GND
13 (15)	Q4	Q4	Q4		Q4	Q4	D4
14 (16)	Q5	Q5	Q5		Q5	Q5	D3
15 (17)	Q6	Q6	Q6		Q6	Q6	D2
16 (18)	Q7	Q7	Q7		Q7	Q7	D1
17 (19)	Q8	Q8	Q8		Q8	Q8	D0
18 (20)	PD/’PGM	A11	A11		A11	A11	A4
19 (21)	A10	A10	A10		A10	A10	A5
20 (22)	‘CS	PD/’PGM	PD/’PGM		‘Csa	‘CS	A3, ‘CS0 . . .’CS4
21 (23)	Vpp	Vpp	A12		‘CSb	A12	+5.0 Volt/A3
22 (24)	A9	A9	A9		A9	A9	A6
23 (25)	A8	A8	A8		A8	A8	A7
24 (26)	Vcc	Vcc	Vcc		Vcc	Vcc	+5.0 Volt
(27)			‘CSb				GND
(28)			Vcc				+5.0 Volt

From table 7-1 can be derived that all these ROM's and EPROM's are compatible. There is only one pin which divides these devices into two types.

Type A	EPROM's TMS 2516 and TMS 2532 use pin 21 for Vpp, this means 5 volt.
Type B	EPROM TMS 2564 and the ROM's TMS 4732 and TMS 4764 use pin 21 for A3 or 'CSb. Note, that the number for this pin for the TMS 2564 is 23.

NOTE

A TMS 2564 has 4 pins more the the other devices but is compatible with the other IC's.

Supply voltage:

Vcc	5.0 Volt nominal
Vpp	5.0 Volt nominal in normal mode
Vpp	25.0 Volt nominal in program mode
Vss	0.0 Volt nominal

7.2 JUMPER USAGE

To let the user select between the features of this board, some jumpers are installed. Because there are two types of (EP)ROM's they are called type A and type B. (See paragraph 7.1). All jumpers are placed into their default positions during production. This means the board allows more then one (EP)ROM of type B. When a change of jumper is needed the solder bridge between the two solder joints can be removed and the necessary connection can be made. The following jumpers are used:

Table 7-2 Jumper usage

One/more (EP)ROM	Type A or B	Jumper(s) to be removed	Jumper(s) to be installed	Extra actions or comment
One	A	E1 – E3	E1 – E2 E6 – E7 E4 – E5	Remove IC5, 6, 7 or Jumper E6 – E7 and E4 – E5
One or more	A	E4 – E5 E1 – E3 E6 – E7	E1 – E2	Install IC5, 6, 7
One	B	E1 – E2	E1 – E3 E6 – E7 E4 – E5	Remove IC5, 6, 7 or Jumper E6 – E7 and E4 – E5
One or more	B	E1 – E2 E6 – E7 E1 – E3	E4 – E5	Install IC5, 6, 7. This is the default option.

NOTE

All jumpers have a spacing from 0.1 inch. The 3-pin jumper has its holes inline.

7.3 STACKING GROM'S AND (EP)ROM'S

There is only one position reserved for a GROM IC. Due to the fact that GROM's have internal chip select logic, which allows parallel connection, there is height enough to stack one extra GROM. Next to the (EP)ROM's IC1 and IC2 is a hole which represents the extra select to allow stacking of such a device. The select pin for the upper device may not be connected to the lower one. It should be bent apart and connected by means of an isolated wire to one of the select holes.

NOTE

When stacking of (EP)ROM's is necessary, be careful of the pin compatibility of the upper and lower device.

7.4 TI EDITOR/ASSEMBLER MANUAL ERRORS

Until now the following errors are found in the "TI EDITOR/ASSEMBLER" manual with part number 1035984-0001.

PAGE	SECTION	DESCRIPTION
42	3.1.3.1	In the first paragraph, last sentence: change "least" into "most".
83	6.2	Line 5 should read: "plus >18 (the value in memory byte 2123)".
92	6.10	In the second line of the example: change "value of addr." into "value in addr."

103	6.14.2	In the example: change "MOV *11,1" into "MOV *11+,1".
127	7.20.1	In next to last line: change ">2220" into ">C220".
142	8.2	In the example-table at the bottom of the page: the columns under Source and Destination must be swapped. (Compare with the table on page 140).
168	10.5	In the example: change ">2A41" into "@>2A41", and change "Register 3" into "Register 2".
214		This page is an exact copy of page 212; the real page is missing, containing for example the chapter concerning PSEG.
230	14.4.3.1	Last paragraph: change "source listing" into "object file".
231	14.4.3.1	Replace the second paragraph with the following text: To run the program, select the LOAD AND RUN option on the TI Editor/Assembler. The file name is DSK1.TOMB, the program name is START. Alternatively, you may run the game from TI EXTENDED BASIC. However, the object file TOMB is in truncated format, so the file TOMBS must be reassembled. Hereafter the following program may be run: 100 CALL INIT 110 CALL LOAD("<object.name>") 120 CALL LINK
262	16.2.4	Add the following note: NOTE: Some devices modify the GROM Read address. RS232 and

		TP are known offenders. if your program accesses these devices, it should save the current GROM address before the I/O operation (See Section 16.5.2), and restore it afterwards (See Section 16.5.1). Otherwise the program will not be able to return to the EDITOR/ASSEMBLER or to BASIC, or perform a "BLWP @GPLLNK" properly.
289	15.2.1	Change line 130 in the BASIC program into: CALL LOAD("DSK1.BSCUP","DSK2.STRINGO"). This assumes that the source file on the next page has been entered using the EDITOR, and saved as "DSK2.STRING", and that the ASSEMBLER has been run, using "DSK2.STRING" as source file and producing "DSK2.STRINGO" as object file.
328	21.1	The default for Register 7 is >07 in TI BASIC and EXTENDED BASIC.
335	21.5	In the second paragraph: change ">00 or >04" into ">03 or >07". In the next to last paragraph included: change ">00 or >04" into ">7F or >FF".
344	21.7.1	Change line 2 into "AI R0,->20".
415	24.4.8	The second instance of "GRMRD" must be changed into "GRMWA EQU >9802".
416	24.4.8	Change the second line into: "NUMREF EQU >200C".