ATARI A1200

Operating System Manual
(Supplement to the A400/800 Manual)

Robert A. Peck

First Draft
11/11/82

(Excludes: Section 5 part 1
and Appendices A and B)

Second Draft (complete document)
11/30/82

Adds: Section 5 part 1,
Appendices A and B, and
Appendices C and D

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT'D)

1.0 INTRODUCTION

This manual is designed to serve as a supplement to the ATARI400[TM] and ATARI800[TM] OPERATING SYSTEM MANUAL.

The A1200, as shown in sections 3-5, is a technical upgrade of the A800. The operating system for the A1200 has been written to maintain, as much as possible, compatibility with application programs which have already been developed for the A400/800.

Since the basic hardware which controls the user interface and the display is, for the most part, compatible with the earlier designs, the operating system, except for the enhancements or changes described here, has remained largely the same. Therefore the data contained in the OS manual for the A400/800 is still valid.

This manual has been written to provide the user with data regarding usage of the added features of the A1200 operating system, with some details about the characteristics of the peripheral devices with which it will operate. Programmers or peripheral developers who require a greater level of detail regarding the handling of peripheral devices should refer to the documents referenced in item 2 of section 2 below.

2.0 APPLICABLE DOCUMENTS

1. ATARI Home Computer Operating Systems Manual. Describes the OS for the A400 and A800, which is the basis for the enhancements described in this manual.

2. SERIAL INPUT/OUTPUT INTERFACE USER'S HANDBOOK PART 1.

   This document provides a detailed explanation of the interface requirements and the timing relationships for the serial communications capabilities of all of the units (A400/800/1200).

3. ATARI Home Computer Hardware Manual and A1200 Supplement.

   The Hardware Manual covers the hardware registers which control the various functions of the A400 and A800. The supplement to the hardware manual covers the added features for control of the A1200. Such details which are appropriate to the OS handling of such hardware registers ARE contained in this OS manual. The user who has need for other hardware-related data should refer to the hardware manual for more information.

4. DE RE ATARI

   This document provides the user with an introduction to the effective use of the ATARI Home Computer hardware. Although written to cover the A400/800, the data contained therein is valid for the A1200 as well.

3.0 HOW THE A1200 COMPARES TO THE A400/800

The following is a list of the features and functions which will be discussed in this chapter. Each will be explained in a separate section.

In this chapter, you will learn about:

1. The HELP Key
2. The Function Keys
3. How key codes are redefined and which ones cannot be redefined
4. How to alter the key repeat rate
5. The action of the Caps/Lowr Key
6. How the OS initializes the LED's on the keyboard
7. What happens when a cartridge is installed or removed
8. What happens during power-on self-test.
9. What the option jumper assignements mean
10. What new screen modes the A1200 can use
11. How to enable fine scrolling of the text screen
12. How the disk handler has been changed for improved operation
13. What kind of display is now produced at power-up
14. What features have been deleted as compared to the A400 or A800

Each of the items enumerated above corresponds to the paragraph number in this section which follows. For example, item 1 above is covered in paragraph 3.1, item 2 in paragraph 3.2 and so forth.

3.1 The HELP Key

The operating system, while watching the keyboard, will recognize the pressing of the HELP key as a request to set a flag in the OS database. This flag can be read by whichever application program is in control at the time and react accordingly.

No ATASCII keycode is created for passing to the applications program. Only the database variable is affected. Therefore if your program is expecting the Help key to be pressed, you must not only watch the keyboard for incoming ATASCII codes other than Help, but also occasionally check ("poll") the contents of the HELPFG (help flag) database variable to see if Help was requested.

The location of this variable is $02DC. The conditions to which it responds are listed below, along with the codes which will be stored in HELPFG:

Hex value		Condition represented
00	0	The Help flag is cleared. This flag is cleared at initial power-up reset and subsequently, if set, must be cleared by the application program.
11	17	HELP key alone was pressed.
51	81	SHIFT-HELP key combination was pressed.
91	145	CTRL-HELP key combination was pressed.

The HELP key can be used during the power-on display and during the self test feature. See those sections for more information.

3.2 What The FUNCTION Keys Do

The A1200 is provided with a set of four function keys. You may redefine the ATASCII values which these keys produce if you desire. As a matter of fact, the entire keyboard ATASCII output may be redefined as will be seen later. This section shows the normal definition of the F1-F4 keys, their functions and the ATASCII codes which they produce (if any) as a result of the power-on reset assignment. All values in the table below are given in hexadecimal.

FUNCTION KEY ASSIGNMENT SUMMARY

Key	If pressed alone
F1	Produces the Cursor-up function, returns ATASCII 1C
F2	Produces the Cursor-down function, returns ATASCII 1D
F3	Produces the Cursor-left function, returns ATASCII 1E
F4	Produces the Cursor-right function, returns ATASCII 1F

FUNCTION KEY ASSIGNMENT SUMMARY (cont'd)

Key	If pressed with SHIFT
F1	See HOME CURSOR below
F2	See CURSOR TO LOWER LEFT CORNER below
F3	See CURSOR TO BEGINNING OF PHYSICAL LINE below
F4	See CURSOR TO FAR RIGHT OF PHYSICAL LINE below

Key	If pressed with CTRL
F1	See KEYBOARD ENABLE/DISABLE below
F2	See SCREEN DMA ENABLE/DISABLE below
F3	See KEY-CLICK ENABLE/DISABLE below
F4	See DOMESTIC/INTERNATIONAL CHARACTER SET below

Key	If pressed with CTRL+SHIFT
F1	Ignored
F2	Ignored
F3	Ignored
F3	Ignored

HOME CURSOR FUNCTION

SHIFT-F1 causes the cursor to move to the home position of the screen as well as producing the default ATASCII code 1C. The default code is reassignable, however the home cursor function will remain assigned to this key combination regardless of the code to be produced.

CURSOR TO LOWER LEFT CORNER

SHIFT-F2 causes the cursor to move to the lower left corner of the screen as well as producing the default ATASCII code 1D. The default code is reassignable, however this cursor move function will remain assigned to this key combination regardless of the code to be produced.

CURSOR TO BEGINNING OF PHYSICAL LINE

SHIFT-F3 causes the cursor to move to the far left of the physical line on which it is located (note, not the logical line which, in the screen editor, could be as many as 3 physical lines.) This function is performed by the screen editor as well as generating the default ATASCII code 1E. The default code is reassignable, however this cursor move function will remain assigned to this key combination regardless of the code to be produced.

CURSOR TO FAR RIGHT WITHIN PHYSICAL LINE

SHIFT-F4 causes the cursor to move to the far right side of the physical line on which it is located. This function is performed by the screen editor as well as generating the default ATASCII code 1F. The default code is reassignable, however this cursor move function will remain assigned to this key combination regardless of the code to be produced.

KEYBOARD ENABLE/DISABLE

CTRL-F1 controls the keyboard enable/disable function. It produces no ATASCII code. This key combination affects the operating system handling of the keyboard and is not reassignable.

SCREEN DMA ENABLE/DISABLE

CTRL-F2 controls the Screen Enable/Disable Direct Memory Access (DMA). It produces no ATASCII code. This key combination affects the operating system handling of the display function. This key combination is not reassignable.

The A1200, on power-up, always enables the screen DMA. What this means is that the system will always initialize itself to display anything which has been defined for the screen display during power up. This same screen DMA enable will also occur if you touch any keyboard key other than the CTRL-F2 combination.

Various types of programs which you write may be heavily involved in arithmetic computations. To speed up the processing, in the A400 or A800, you may disable the screen DMA. When it is disabled, the ANTIC processor does not steal memory cycles from the 6502 to get its data for the screen. Therefore during disable mode, the screen remains blank. When it is enabled, the full display which you have defined is visible, however, the processor is slowed down by anywhere from 10 to 40 percent as explained in the section on ANTIC DMA in the Atari Hardware Manual.

On the A1200, to start the higher speed/ no display function, press the CTRL-F2 key combination. The display will go blank. To restore the display again at any time, you can press any other key.

During your arithmetic calculations, you may be in continuous process of updating the memory area where the display data is contained. You can then get a status of the operation in process at any time simply by pressing any key other than CTRL-F2, then again press CTRL-F2 to re-enter the higher speed mode.

Your program, then, on completion of the calculation, could exercise direct program control over the ANTIC DMA variable to restore the display when the arithmetic intensive part is over. (See the ATARI Home Computer Hardware Manual for data on programmed control of this variable.)

KEY-CLICK ENABLE/DISABLE

CTRL-F3 controls the Key-Click enable/disable function. If pressed once, it disables the audible feedback on keystrokes. Pressed again reenables it. This function combination only affects an OS database variable and produces no ATASCII code. It is not reassignable.

You may control the key click enable/disable from your program. All that needs to be done is to change the same flag which the operating system uses to indicate whether a key click is required. This flag is called NOCLIK. It is one of the OS database variables, contained at location $02DB.
731

On power up and reset, the operating system initializes this variable to a value of 00, meaning that key click is enabled. This location, when it contains the value $FF, indicates that no key click is desired. The key combination CTRL-F3 toggles it between the values 00 and FF.

In addition to this flag, when the operating system controls the keyboard, it tells you the enable/disable status using the light emitting diode number 1 (referred to as LED 1.) Whenever the operating system disables the keyboard, it will light LED 1; whenever it enables the keyboard, it will turn it off. The operating system does not change the status of the light if YOU disable or re-enable the keyboard under program control.

DOMESTIC/INTERNATIONAL CHARACTER SELECTION

CTRL-F4 controls the domestic/international character selection. Default is domestic. It affects an OS database variable only and produces no ATASCII code. It is not reassignable. It toggles the display of character sets, changing between the two each time the key combination is pressed. When the international character set is selected, LED number 2 will be lit.

The international version of the character set is located in the ROM beginning at location $CC00. You can cause the international character set to be selected by storing the constant $CC to location $02F4. This is the location CHBAS. The normal character set is located in the ROM starting at $E000. If a program stores $E0 to CHBAS, it selects the display of the normal characters.

If you have defined your own character set, however, pressing CTRL-F4 will display the international character set. This is because the operating system will test CHBAS and find that the value $C8 is not there. Therefore $C8 must be the next value which is to be used (selects int'l set). When it tests CHBAS and finds $C8 stored there, it knows that $E0 is the next value to use during the toggle between character sets.

3.3 KEY REDEFINITION

You may redefine most of the A1200 console keys if desired. The redefinition process consists of setting up a pair of tables which can be referenced by the operating system when it translates your keystroke into an ATASCII value.

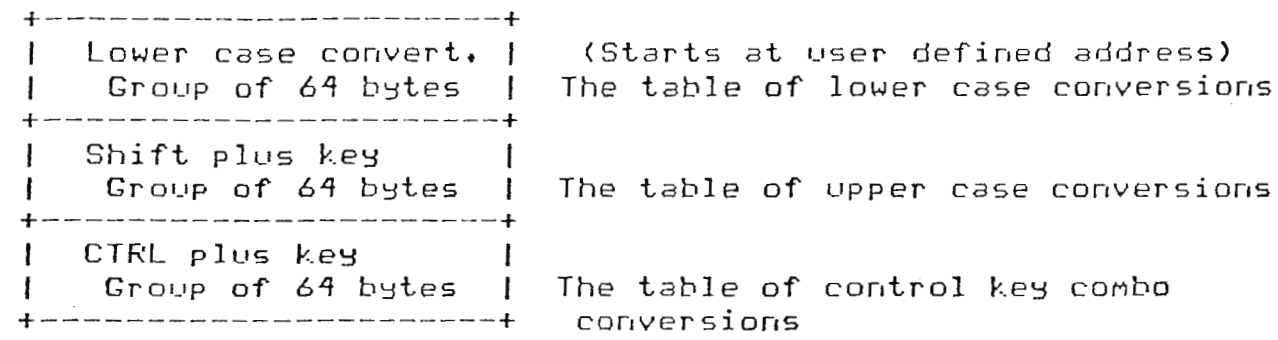
The two tables are the KEY Definition Table and the Function Key Definition Table. The operating system has a pair of data tables from which the normal definitions are made. You may define your own set of tables however, then simply tell the operating system where they are located in memory.

One such use of key redefinition might be to experiment with other, possibly more efficient keyboard layouts, such as perhaps the Dvorak keyboard. An example is given in Appendix A of a keyboard redefinition to allow you to do such an experiment. (Over the years, the QWERTY key layout has been the accepted standard however many people have found DVORAK to be more efficient. This would allow you to try it for yourself.)

CONTENTS OF THE KEY DEFINITION TABLE

This table allows most of the keys of the A1200 to generate any desired ATASCII code or special internal function. The exceptions to this are listed at the end of this section. To redefine the keys, it is necessary first to define an area in memory where a 192 byte table may be stored. Into this table, you will store the definitiions of the keys which you desire. Later you will tell the operating system where this table is located so that future references may be made to it instead of the standard definition table.

The organization of this table is as follows:

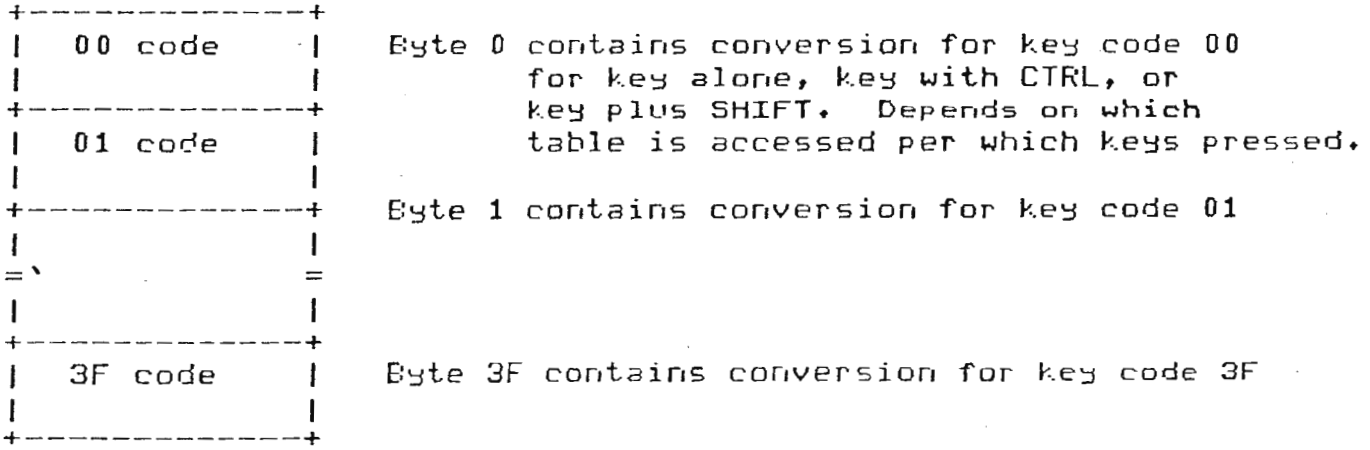


The bottom-most byte in the table shown above is at the address KEYTABLE_START + 191.

The reason that each of the subdivisions of the table has 64 bytes in it is that the hardware can generate a total of 64 hardware keycodes. These codes, numbered 00-63 decimal (00-3F hexadecimal) are used to index directly into one of the three keycode tables. Which table is referenced depends on whether the CTRL or SHIFT keys are pressed.

Note that there is no table for the combination of both CTRL and SHIFT. This combination is invalid and is ignored by the operating system.

Each of the three 64 byte subsections of the table has the form:



The codes which you place in your table will either generate an ATASCII code (for direct character translation) or they will tell the system to perform a specific function. Specifically any code in the range of 80 to 92 hexadecimal will be treated as special by the system. This is illustrated in the table below.

CODES AND THEIR EFFECT ON THE SYSTEM AFTER TRANSLATION

CODE	EFFECT (if any)
00 thru 7F	Used as the ATASCII code only.
92 thru FF	Used as the ATASCII code only.
80	Ignore, invalid key combination.
81	Invert the video output to the screen.
82	Alpha lock/Lower case toggle.
83	Alpha lock
84	Control Lock
85	End of file
86	ATASCII code
87	ATASCII code
88	"Gonzo" function
89	Key click on/off
8A	Function 1 *
8B	Function 2 *
8C	Function 3 *
8D	Function 4 *

* NOTE: When it sees these keycode translations, it is told to DO the function which is described in the Function Key descriptions. This function will be a cursor move and is independent of the ATASCII code which the specific Function Key will produce. The ATASCII coded generation for the normal and shifted function keys is handled in a different table, whose description follows that for the keycode hardware translate table.

8E	Cursor to home
8F	Cursor to bottom
90	Cursor to the left margin
91	Cursor to the right margin

The table below shows the key cap corresponding to each key code. The physical position of each key switch within the table determines the hardware code which it will generate. To determine what code it is, take the row address of the cap, and add it to the column address. The result is the hexadecimal value returned to the operating system (range 00-3F) for use in the table lookup for that key.

KEYCODE DEFINITIONS TABLE

	0	1	2	3	4	5	6	7
00	L	J	;	F1	F2	K	+	*
08	O		P	U	RET	I	-	=
10	V	HLP	C	F3	F4	B	X	Z
18	4		3	6	ESC	5	2	1
20	,	SPACE	.	N		M	/	)| (
28	R		E	Y	TAB	T	W	Q
30	9		0	7	BACKS	8	<	>
38	F	H	D		CAPS	G	S	A

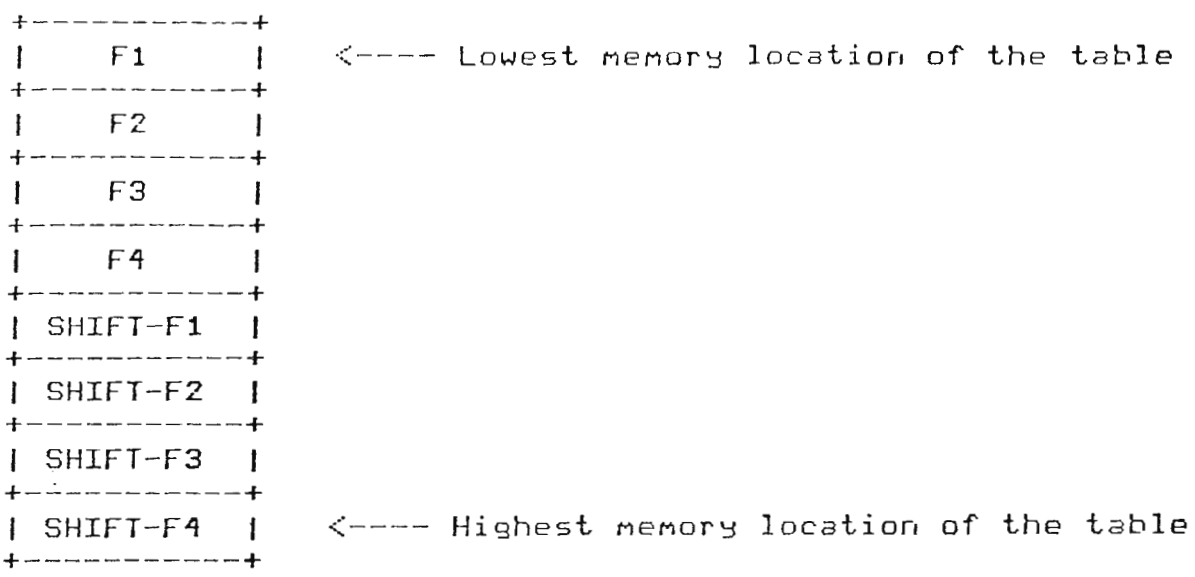
As an example the key cap "C" is in the table in row 10, column 2. This means that the hardware generates a hardware code 10 + 2 or 12 hexadecimal. Therefore, in the translation tables shown above, the function code or ATASCII code for this character will be stored in the key definition table position $12 for each of the three types of "C" which are valid (c alone, Shifted C, or Control C). You may cause each of these perform a separate function or generate a separate ATASCII code by revising the tables.

When you have decided on how you want your keys to be redefined, you tell the operating system where it may find the definitions by storing the address of those definitions in locations 79 and 7A hexadecimal.

The low byte of the hexadecimal address where you have stored the keys should be placed in location 79, the high byte is location 7A. This is defined as one of the system vectors. It will point to the default, or original key definition table at power-on reset time.

REASSIGNMENT OF THE FUNCTION KEYS ONLY

There may be times when you only want to redefine the function keys and not redefine the rest of the keyboard. The A1200 operating system allows you to redefine these keys only by setting up an 8-byte table in place of the 192 byte table which would have otherwise been required. The format of this table is as follows:



When you have decided what functions each combination must perform and have built the table, change the system vector FKDEF to point to the lowest address of your table. This vector is located at memory locations 60 and 61 hexadecimal. Location 60 gets the low byte of the hex address, location 61 gets the high byte.

NON-REASSIGNABLE KEYS AND KEY COMBINATIONS

The following keys or key combinations are either specifically wired for special functions or are subjected to special handling by the operating system.

Even though there might be a hardware-generated key code shown in the table above, and a corresponding space in the translate tables, there is no way to reassign these functions. This is because the operating system traps the hardware code directly to perform the specified function and it never gets to the translate mode. These keys or combinations are as follows:

BREAK -- This function is fixed as a special case in the operating system. It is sensed by the hardware.

SHIFT -- This key is an integral part of the hardware encoding of any key function.

CTRL -- This key is an integral part of the hardware encoding of any key function.

OPTION --+
SELECT |---- All of these are directly wired to and are
START --+ sensed by the GTIA circuitry.

RESET -- Directly wired to the 6502 reset line.

HELP -- Function is fixed by the operating system. The help function handling is described elsewhere in this manual.

CTRL-1 Screen output start/stop function. Trapped by the operating system at the hardware key decode level, controls the listing start/stop function. See the Users Manual for the A1200.

CTRL-F1 This key combination is used as the keyboard enable/disable function. If it is pressed while the keyboard is enabled, it will disable all keyboard functions with exception of the following:

CTRL-F1 can still be used to re-enable;
RESET is the 6502 reset key, cannot be disabled;
OPTION/START/SELECT not controlled by the operating system.

CTRL-F2 See SCREEN DMA CONTROL above. As noted there, this function is not reassignable.

CTRL-F3 See KEY-CLICK ENABLE/DISABLE above. As noted there, this function is not reassignable.

CTRL-F4 See DOMESTIC/INTERNATIONAL CHARACTER SET above.

3.4 USER-ALTERABLE KEY AUTO-REPEAT RATE

The A1200 operating system allows you to control the rate at which a key, continuously held down, will repeat its entry to the system. This change may be done under program control by modifying the OS database variable called KEYREP. It is located at hex address 02DA.

This variable determines the repetition rate by counting the number of VBLANK (vertical blanking) intervals which occur. For the NTSC (60 Hz) system, the initial value of this variable is 6; for PAL systems, the value is 5. This assures a uniform repeat rate of 10 characters per second for either system.

Under control of this variable, the maximum "controllable" key repeat rate would be 50 characters per second on the PAL, and 60 characters per second on the NTSC (screen refresh rate). This would occur with a value of 1 in this variable.

You may also control the initial delay which occurs before the key repeat starts. The OS database variable which controls this is called KRPDEL. Its hex address is 02D9.

It controls the number of VBLANKs which must occur between the sensing of the key pressed until the first repeat occurs. From that time on, the repeat rate is controlled as described above. The initial values used by the OS provide a 0.8 second initial delay for either NTSC (count = 48) or PAL (count = 40) systems.

3.5 CAPS/LOWR KEY TOGGLE ACTION

The CAPS/LOWR key on the A1200 functions as shown in the chart below:

KEY COMBINATION	CURRENT STATE	NEW STATE
CAPS	Control Lock	Lower Case
CAPS	Alpha Lock	Lower Case
CAPS	Lower Case	Alpha Lock
SHIFT-CAPS	- any -	Alpha Lock
CTRL-CAPS	- any -	Control Lock
CTRL-SHIFT-CAPS	- any -	- no change -

The meaning of the terms is as follows:

Lower Case - All key caps respond in lower case mode
Alpha Lock - All alphabetic keys (A-Z) respond in upper case mode, all other keys lower case
Control Lock - All keys respond as though the control key is being held down as well as the selected key

3.6 LED INITIALIZATION

The A1200 has two LED's on the front panel, called LED 1, and LED 2. LED 1, when lit, indicates that the Keyboard is disabled. LED 2, when lit, indicates that the international character set is selected. The operating system enables the keyboard and selects the domestic character set on power up and reset. Therefore these LED's will both be off.

3.7 GAME CARTRIDGE REMOVE/INSERT INTERLOCK

In the A400 and A800, the cartridge interlock mechanism physically removed the power from the entire system when the cartridge door was opened.

The A1200 no longer requires this power down of the entire system. It does, however, automatically cause a power-up initialization sequence to occur if a cartridge change is detected while the power is on.

The initialization sequence itself contains a jump through the cartridge initialization address which adjusts the A1200 to this cartridge immediately upon its insertion. Likewise, if a cartridge is removed, the system reconfigures itself through the power on sequence, to be a no-cartridge system. This initialization is handled by the Stage 2 VBLANK routine.

3.8 POWER-ON SELF-TEST

During the initial power-on, the A1200 operating system will perform the following quick check of the integrity of the system RAM and ROM:

a. Is it possible to write $FF (all ones) to all RAM locations?

b. Is it possible to write $00 (all zeros) to all RAM locations?

c. Does a checksum of the two ROM's compare to that stored within each ROM?

If any of these tests fail, the operating system will transfer control to the self-test memory test routine. Here a more thorough test of both RAM and ROM can take place.

3.9 OPTION JUMPERS

The A1200 is provided with a set of four hardware jumpers which are designed to tell the operating system how the system is configured. As of the date of this writing, only one of the four jumpers has been assigned, specifically J1. This is specified in the table below. During the power-on sequence, the A1200 operating system reads the state of these jumpers and stores this state in the OS database variable JMPERS, location 030E.

The bit assignments for each of the four jumpers is as specified below. The bits are all active low, meaning that if a line reads a digital zero, the jumper is installed.

BIT	FUNCTION	HARDWARE NAME
0	Self test enable (will run self test if low)	J1 (pot 4)
1-3	Reserved for future use	
4-7	Unused	

3.10 ADDITIONAL HARDWARE SCREEN MODES

The A1200 adds direct access to the remaining special purpose display processor operating modes. The table below shows the current mapping which had been provided for the A400 and A800. The table which follows thereafter shows the added modes and the numbers which the software can use to access the extra modes.

Mode mapping common to A400/A800:

Software Mode	ANTIC MODE	GTIA MODE
0 ($00)	2 ($02)	0
1 ($01)	6 ($06)	0
2 ($02)	7 ($07)	0
3 ($03)	8 ($08)	0
4 ($04)	9 ($09)	0
5 ($05)	10 ($0A)	0
6 ($06)	11 ($0B)	0
7 ($07)	13 ($0D)	0
8 ($08)	15 ($0F)	0
9 ($09)	15 ($0F)	1
10 ($0A)	15 ($0F)	2
11 ($0B)	15 ($0F)	3

Mode mapping for A1200 (additional):

Software Mode	ANTIC MODE	GTIA MODE
12 ($0C)	4 ($04)	0 (note 1)
13 ($0D)	5 ($05)	0 (note 1)
14 ($0E)	12 ($0C)	0
15 ($0F)	14 ($0E)	0

Note 1: The existing character sets will not provide recognizable characters for these new modes. Therefore you will have to provide the character set if you use these modes. This is done by defining the full character set, then modifying the OS database variable CHBAS to point to the most significant byte of the address at which the character set starts.

Appendix B of this manual contains some suggestions on the method for designing a new character set to support these added modes.

3.11 TEXT SCREEN FINE SCROLLING

The screen editor (E:) now supports fine scrolling of the text screen data as an option. This fine scrolling option will be enabled if the database variable FINE (hex location 026E) is set nonzero prior to issuing the OPEN command to the screen editor. Likewise the feature will be disabled if this location is set to 00 before issuing the OPEN.

3.12 DISK COMMUNICATIONS ENHANCEMENTS

The A1200 adds the capability for the resident disk handler to read and write disk sectors having variable length from 1 to 65536 bytes. The default length, as is used on the A400 and A800 currently, is 128 bytes. Both at power-on and RESET (warm start), the 128 byte sector length is established. Your program can alter this length by modifying the OS database variable DSCTLN. The location of this two-byte variable is 02D5 and 02D6 (lo byte in 02D5, hi in 02D6).

In addition to the capability to read and write variable length sectors, the A1200 also adds the capability to write a sector to the disk without a read-verify operation always following it. This is the command 'P' which was specifically excluded in the previous releases of the operating system.

With this capability added, you have a choice of either using the verify, for system integrity (always read after write). Or you can take a chance of writing a bad sector on rare occasions but increasing your average speed of disk usage by some value related to the verify time. You may want to experiment with some of your programs with and without verify to see the results.

3.13 POWER-ON DISPLAY ENHANCEMENT

In place of the original power-on memo pad display used by the A400 and A800 (in the absence of a cartridge or disk), the A1200 displays a dynamic ATARI rainbow. If you press the HELP key while the rainbow is displayed, the A1200 will enter the self-test mode.

4.0 MEMORY MAP OF THE A1200

The following table shows how the 6502 processor perceives the various address spaces which it can access. The maximum allowable address range, with the 16 bit address of the 6502 is hexadecimal 0000-FFFF. This address range is split, by the hardware memory management circuitry, as follows:

(Note: The A1200 uses 64K RAM's as the main system writeable memory. Addresses within those RAM's, which would normally have filled the entire memory access space of 0000-FFFF of the processor, are prevented from access by the memory manager. This allows ROM's, cartridge memory, and peripherals to occupy a part of the memory space as is noted below.)

A1200 MEMORY MAP

HEX ADDRESS	WHAT IS ACCESSED THERE	NOTES
FFFF-D800	OS-ROM or RAM if ROM disabled	1
D7FF-D000	Active low chip selects are produced for the peripheral chips through accesses in this memory page.	
	Memory Mapped I/O Space split as D000-D0FF GTIA D200-D2FF POKEY D300-D3FF PIA D400-D4FF ANTIC D500-D5FF Any access read or write to an address in this range enables the cartridge control line CCNTL on the cartridge interface (same as A400/A800).	
	D100-D1FF, D600-D6FF, and D700-D7FF are reserved for future use.	
	OS-ROM physically present, but cannot be accessed here.	2
CFFF-C000	OS-ROM or RAM if ROM is disabled	1

HEX ADDRESS	WHAT IS ACCESSED THERE	NOTES
BFFF-A000	RAM, or cartridge interface if RD5 line is pulled up to +5V by the cartridge board. A1200 MEMORY MAP (cont'd)	
9FFF-8000	RAM, or cartridge interface if RD4 line is pulled up to +5V by the cartridge board.	
7FFF-5800	RAM	
57FF-5000	RAM, unless in self-test mode	2
4FFF-0000	RAM	

NOTES:

1. Access to the OS ROM may be disabled by writing a zero to port B of the PIA, bit PB0. Access is normally enabled, with a 1 present in this bit. (When changing this bit in the register, other bits should not be changed.)

2. The self-test ROM code is physically present in the OS ROM at actual address D000-D7FF. However, this area is used for the access to the memory mapped I/O devices. When the self-test feature is invoked, the RAM located from 5000-57FF is disabled. The memory manager remaps the memory access such that the OS ROM physical addresses D000-D7FF are accessed at 5000-57FF. The memory manager uses port B of the PIA, bit PB7 to determine whether to access RAM or ROM in the region 5000-57FF. PB7, if high, accesses RAM. If low, causes an OS-ROM access instead. (When changing this bit in the register, other bits should not be changed.)

(Port B was used in the A400/800 to service the game ports 3 and 4. The use of the remaining bits of this port are specified elsewhere in this manual.)

5.0 ENHANCEMENTS TO THE A400/800 REV.B OPERATING SYSTEM INCORPORATED IN THE A1200

This section describes a set of enhancements which include new methods of handling peripheral products and, in a separate section, improvements in basic operations of the system. The latter might be referred to as "bug fixes".

PERIPHERAL HANDLER ADDITIONS

To accommodate a new class of peripheral devices, the operating system now includes a relocating loader, used to upload peripheral handlers through the serial I/O interface.

In the A400/800, device handlers for the peripherals were uploaded as fixed location (absolute) object code. These handlers were loaded using a set of device inquiries, or polls, known as types 0, 1 and 2. These polls are described further in the Atari 400 and 800 OS Manual.

The A1200 adds two other types of polls to its operating system. One poll, known as type 3, is issued at power-on or reset time. The other, type 4, can be issued as a result of an OPEN command by an application program.

Type 3 Poll Command

The type 3 poll command itself is used as an "Are You There?" type of command. Associated with the type 3 poll are two other types, specifically the:

a) Poll Reset

and b) Null Poll

Poll Reset consists of the following SIO command byte sequence (refer to the SIO document for further explanation of the byte types):

Byte Position	Value (hex)
Device Address	4F
Command Byte	40
AUX1	4F
AUX2	4F
Command Checksum	Normal (checked by peripheral)

The 4F in AUX1 and AUX2 define this sequence to all peripherals as a poll reset.

After responding to a type 3 poll by sending a handler to the system, a peripheral is not supposed to respond again to a type 3 poll. The Poll Reset command, at power-up, resets all type 3 peripherals, freeing them to respond to the poll request. However, no serial bus device sends back any data as a result of a poll reset command.

Type 3 Poll (Are you there?)

There may be several types of peripherals which can respond to a type 3 poll. In types 0, 1 and 2, the device address sent on the serial line specifies which exact device is being called. In the type 3 poll processing, however, the address remains fixed (4F) and the devices each respond after a specific number of poll 3 retries. In other words, during poll 3 operations, the computer doesn't know which peripherals are actually attached, but will keep asking "is anybody there" until it has reached its last retry and no peripheral has responded.

Each peripheral which does respond to the type 3 poll must be designed to count the number of retries of type 3 polls, then to respond as described below on its own specified retry slot. Each time it sees a command other than a type 3 poll, these peripherals must reset their retry counters. This allows the computer to load the handler for each peripheral which responds, then restart its poll 3 sequence (original retry number restored) to look for another poll 3 response from the next peripheral (if any).

Since each peripheral responds only once (after a poll reset), a second request at a specific retry slot causes no peripheral response and allows the next retry slot to be polled.

This poll ("are you there?") is sent as follows:

Byte Position	Value (hex)
Device address	4F
Command Byte	40
AUX1	00
AUX2	00
Command checksum	Normal, checked by peripheral

When, after checking the retry count, it is a peripheral's turn to respond, it sends back the following data to the computer on the serial interface:

a) An ACK response byte, and

b) 1. Low byte of handler size in bytes (must be EVEN)

2. High byte of handler size

3. Device Serial I/O Address to be used for loading

4. Peripheral Revision Number

These four bytes, if sent by the peripheral, will be stored in OS variables DVSTAT (02EA hex) through DVSTAT+3. If there is a successful return to the OS (not a timeout or other problem), it indicates that there is a handler to be loaded. The loading is performed, then the type 3 poll is repeated until all retries are exhausted and no peripheral responds.

Once the device address data is received from the peripheral during this type 3 poll, it can thereafter be referenced directly on the serial bus by its address in place of the original poll address 4F.

Specific details of the actions taken by the OS after receiving an answer from a peripheral may be found in Appendix C.

Null Poll Command

This command is used as a serial bus no-operation. If any error should occur during loading of a peripheral handler or by the relocator, (see appendix C and D), the system should be free to "back out" of the linking of the faulty loader and tell the peripherals that it is ready for the next one to be loaded. Since this null poll is a non-type-3 poll, all peripherals will have reset their retry counters and should be ready for another sequence of retries, looking for their own response retry slot. This maintains synchronization between the computer and the peripherals.

This poll differs from the Type 3 Poll in that the device name and number is included in the poll. Therefore the peripheral need not count retries of the type 4 poll and should answer the poll as soon as the poll command is recognized. There is no limitation on the type 4 poll; the peripheral should answer its type 4 poll each time it is issued.

The peripheral response to a type 4 poll is the same as for the type 3 poll. The four response bytes are placed, by the computer, into DVSTAT through DVSTAT+3 (02EA through 02ED hex.).

GENERAL ENHANCEMENTS TO THE REV. B OS FUNCTIONS

The following functions which are supported by the A400/800 Rev. B Operating System have been further enhanced by the addition of the following features:

Printer CLOSE with data in the buffer -

The printer handler will insert an EOL (end-of-line) character in the printer buffer, if one is not there, before sending the buffer to the printer on a CLOSE. This assures that the last line will be printed immediately rather than having the printer forced offline to output the final line.

Printer Unit Number Handling -

The printer handler has been changed so that it will process the unit number in the IOCB, allowing separate addressing for printers P1 through P8.

CIO Handling of Truncated Records on Read -

The CIO now places an EOL in the users input buffer on the occurrence of either a record longer than the buffer being read or an EOF being encountered during the read attempt. This assures that all records are accessible, even if the user has not provided a sufficient buffer size, he will at least get as much of the record as he has provided for.

CIO Error Handling With Zero Length Buffer -

The CIO will return a buffer length of zero (in the 6502 A-register) when there is a handler error while effecting a zero length buffer transfer. (See CIO section in the OS manual.)

Display Handler Cursor Handling -

The display handler now accepts a screen clear code no matter what value is in the cursor X and Y coordinates.

Display Handler/Screen Editor Memory Clearing -

The Display handler and Screen editor will not clear memory beyond the end of memory as indicated by RAMTOP. Now it is possible for the user to specify the top of memory to be used by the system and to store device handlers or personal machine code in the memory area above the display. Changing display graphics modes, then, will not erase any data which has been placed in the RAM area above that assigned for use by the display or screen editor.

Rework of the Floating Point Package -

The A1200 operating system corrects a bug in the Rev B OS. It now produces an error status when an attempt is made to calculate the LOG or LOG10 of zero.

New ROM Vectors -

The following fixed entry point vectors have been added to the A1200 ROM set:

```
E480   JMP FUPDIS      entry to power-on display
E483   JMP SLFTST      entry to the self test pgm
E486   JMP PHENTR      entry to uploaded handler enter.
E489   JMP PHULNK      entry to uploaded handler unlink.
E48C   JMP PHINIS      entry to uploaded handler init.
```

6.0 OTHER CHANGES/GENERAL INFORMATION

This section deals with items which involve operating system changes, but which do not easily fit into any other category.

IMPROVED HANDLING OF OS DATABASE VARIABLES

During normal power-on sequence (cold start), the OS database variables from $03ED-$03FF are set to zero. During a RESET (warm start), they are NOT changed by the OS. This means that an enhanced version of the operating system in the future will be able to make use of these locations without reloading them after any RESET operation.

These bytes are all reserved for use in future OS revisions.

NTSC/PAL VERSION TIMING PROVISIONS

There are various timing differences between the NTSC (60 hz) and the PAL (50 hz) versions. To eliminate the necessity for providing a special operating system ROM set for each one, the specific timing adjustment values are handled within the single ROM set.

To determine which type of system the ROM is operating on, the operating system checks a flag within the GTIA chip and adjusts all timings accordingly. This was possible because the GTIA must be different to handle the modified display format for the 50 Hz version. By making certain timings a function of the state of this flag, it was possible to make external timings independent of the NTSC or PAL system itself.

The timing values relate to the handling of the 115 Volt cassette player (Atari 410) and the console auto-repeat rate as shown in the table below:

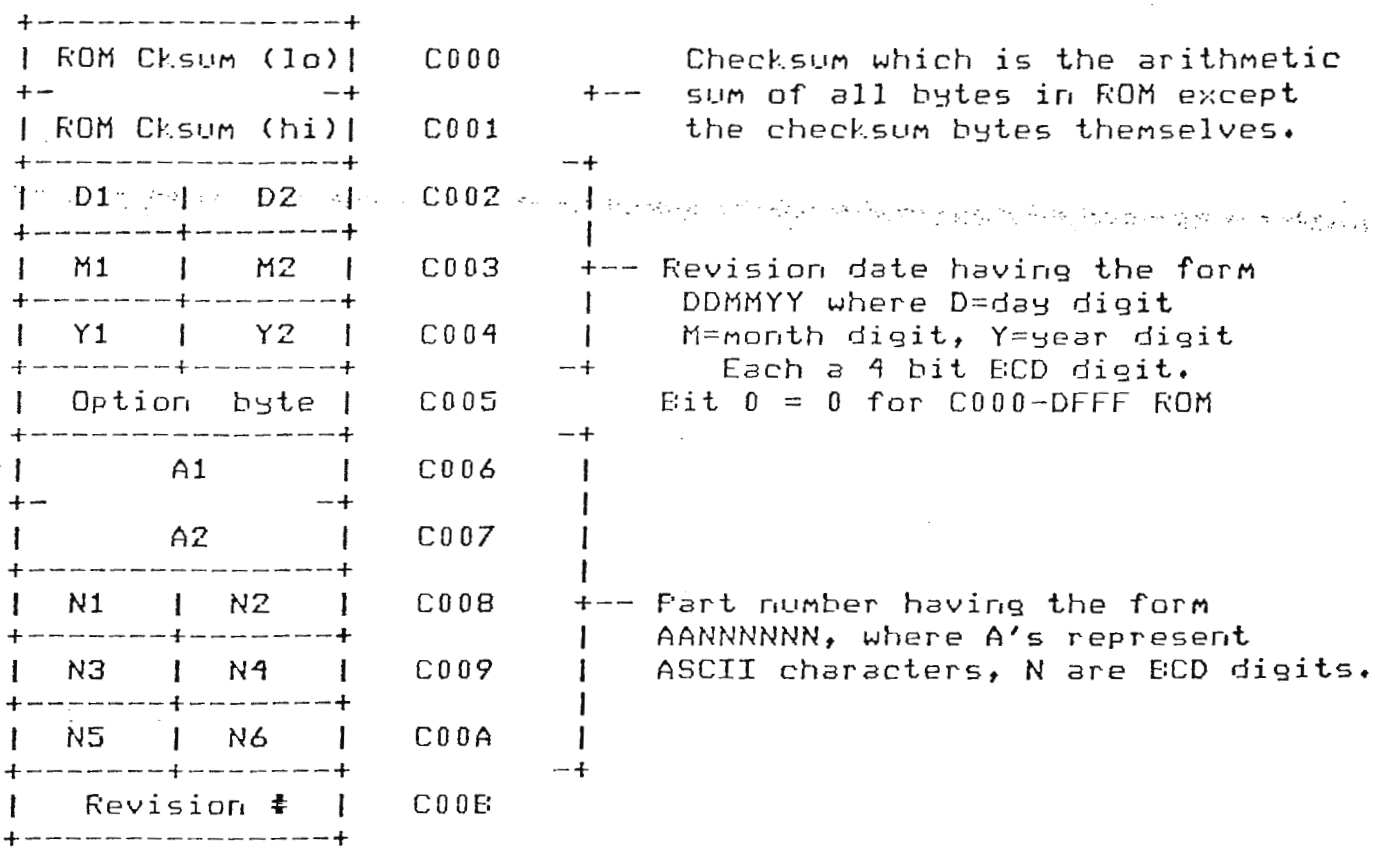
CASSETTE TIMINGS NOW INDEPENDENT	TIMING
Write Inter-record gap (long)	3.0 sec.
Read IRG delay (long)	2.0 sec.
Write IRG (short)	0.25 sec.
Read IRG delay (short)	0.16 sec.
Write File leader	19.2 sec.
Read Leader delay	9.6 sec.
Beep cue duration	0.5 sec.
Beep cue separation	0.16 sec.

Auto-repeat functions now independent	Timing
Initial delay for auto-repeat	0.8 sec.
Repeat rate	10.0 char/sec.

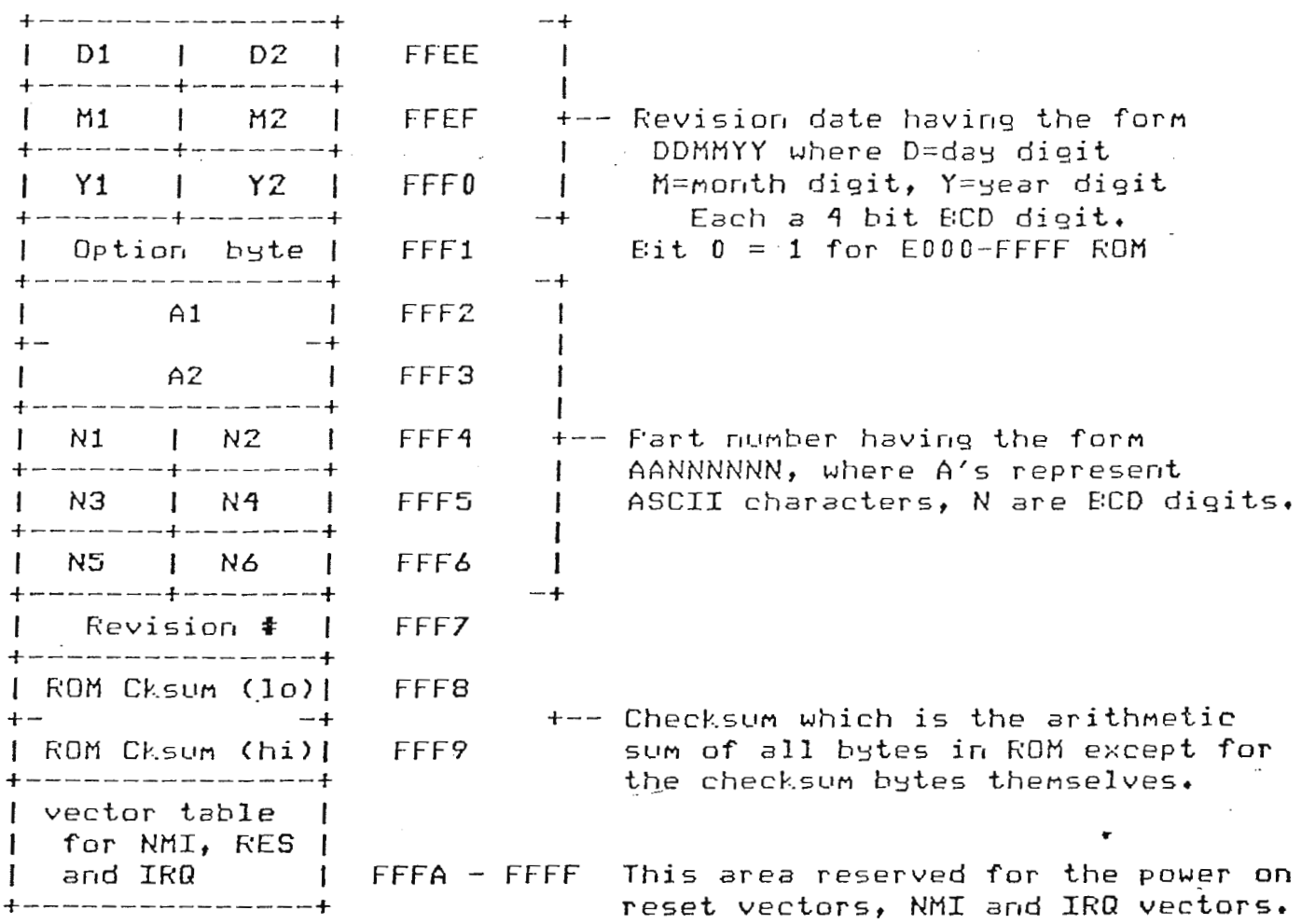
A1200 OS ROM IDENTIFICATION AND CHECKSUM DATA

Each of the two ROM's in which the A1200 operating system is contained has a capacity of 64K bits organized as 8K by 8. Within each of the ROM's is a block of data organized as shown in the diagram below, to identify the ROM and to give its checksum. The checksum is tested by the operating system as part of the power up sequence.

The format of the block for the C000-DFFF ROM is as follows:



The format of the identification block for the E000-FFFF ROM is as follows:



APPENDIX A - AN EXAMPLE OF KEYBOARD REASSIGNMENT

As suggested earlier in this document, the keyboard functions may be reassigned. The table below gives the corresponding keys for the Dvorak (also known as the American Simplified) Keyboard. When the typewriter was first invented in 1867, Christopher L. Sholes chose a layout for the keys which would slow down the good typists of his day and thereby prevent his machine from jamming. This keyboard has endured to this day.

In 1932, August Dvorak invented this key layout which places the most often used characters, including the vowels, on the "home" key line and also redistributes the keystrokes from a 60-70% left-hand activity to an almost 50/50 activity. Certain manufacturers currently offer this key layout as an option. Now you can try it for yourself if you wish. Only the list of key correspondence is given here. It is left to the reader to compose the key function table using the data contained earlier in this manual.

TOP ROW OF KEYBOARD		CENTER ROW		BOTTOM ROW	
Current	Dvorak	Current	Dvorak	Current	Dvorak
Q	?	A	A	Z	:
q	/			z	;
W	,	S	O	X	Q
w	,				
E	.	D	E	C	J
e	.				
R	P	F	U	V	K
T	Y	G	I	B	X
Y	F	H	D	N	B
U	G	J	H	M	M
I	C	K	T	,	W
				,	w
O	R	L	N	.	V
				.	v
P	L	:	S	?	Z
		;	s	/	z
1/4	"	"	_ (underline)		
1/2	'	'	-		

APPENDIX B - SUGGESTIONS FOR THE CONSTRUCTION OF A NEW CHARACTER SET FOR THE NEW GRAPHICS MODES

This appendix covers the new graphics modes 12, 13, 14 and 15 now provided on the A1200. Modes 14 and 15 are pure graphics modes with resolutions of 160 by 20 and 160 by 40 respectively. Since these are not character modes, the discussion below will be limited only to modes 12 and 13.

Graphics 12 and 13 do not produce recognizable characters, for the most part, using the standard character set. One will understand why this is true by examining the following comparison between Graphics mode 0 to 12 and 13.

> Mode 0 is a 40 character mode. Each character is formed out of 8 pixels (smallest division of the screen). Each pixel is 1/2 of a color clock wide.
>
> Modes 12 and 13 are also 40 character modes. However, each character is formed out of only 4 pixels, with each pixel 1 color clock wide. This forces the character to be the same width as that used in Graphics mode 0, but cannot convey the same information within 4 bits as with 8 as far as character recognition is concerned. (It is difficult to form a recognizable character in a four by eight dot matrix).

Lets examine how the 4-pixel character is formed, again comparing the way the 8-pixel character is formed in mode 0:

> Mode 0 has a choice of two colors for each pixel (the hardware manual says 1 1/2 colors, but it is actually either the color and luminance of playfield 2 if there is a zero bit in the selected pixel position, or the background color with the luminance of playfield 1 if there is a 1 bit in the selected pixel position. Therefore each single bit in the character definition byte for a given line occupies a single 1/2-color-clock-wide pixel position. The character set built into the OS defines the characters in an 8 by 8 matrix, with one of the 8 bytes which make up the character selected for each of the 8 scan lines which comprise the character.

Mode 12 also uses 8 scan lines per character. However, it uses the character bytes in a different manner. Each of the character bytes retrieved by the ANTIC is treated as a set of four two-bit quantities, where each bit pair describes the color which is to be applied to one of the 4 single color-clock-wide pixels which are part of the character. Mode 13 is the same in its treatment of the data bytes, but each of the characters is double-length (16 scan lines instead of 8) and each data byte is used twice which effectively doubles the length of the character.

Lets look at a typical character, for example a W. The bits which form a W in the character set are similar to the following:

```
1 0 0 0 0 0 0 1     display: *      *
1 0 0 0 0 0 0 1              *      *
1 0 0 1 1 0 0 1              *  **  *
1 0 0 1 1 0 0 1              *  **  *
1 0 1 0 0 1 0 1              * *  * *
1 1 0 0 0 0 1 1              **    **
1 1 0 0 0 0 1 1              **    **
1 0 0 0 0 0 0 1              *      *
```

(NOTE: This is not the exact representation, but is used as an example of correct interpretation in mode 0 and incorrect interpretation in modes 12 and 13.)

If you view the sample set of bytes, each at consecutive addresses within the defined character set, it actually looks like a W when you trace the outline formed by the 1's in the byte set, as shown in the display example to the right of the byte representation.

In this mode 0 display, each of the 1's would be one color, and each of the zeros would be another color, assuring a readable display.

For the modes 12 and 13, the four (not 8) pixels are controlled as follows:

If two-bit value is:	Then the pixel color is:
00	the background color
01	the playfield 0 color
10	the playfield 1 color
11	the playfield 2 color (if bit 7 of char=0)
11	the playfield 3 color (if bit 7 of char=1)

For the example shown, then, the 4th line from the bottom would display a 10 10 01 01 or 4 pixels of playfield colors 1, 1, 0, 0 in a row, if the standard character set is used. And the bottom-most line would display playfield colors 1, BAK, BAK, 0 in a row. As may be imagined, difficult to recognize such a character. (This character is a mirror image left to right - nonsymmetric characters would be even more difficult to recognize.)

To build a character set for these modes 12 and 13, then, it is suggested that you build each character as double wide, to allow a total of 8 pixels (by 8 lines) to define the character. This would also mean assigning two character set locations for each character and treating each character printed in these modes as two characters to be printed. For the example of the W, the character set might look like this:

Byte set 1:	Byte set 2:
10 00 00 00	00 00 00 10
10 00 00 00	00 00 00 10
10 00 00 10	10 00 00 10
10 00 00 10	10 00 00 10
10 00 10 00	00 10 00 10
10 10 00 00	00 00 10 10
10 10 00 00	00 00 10 10
10 00 00 00	00 00 00 10

Byte set 1 may represent ATASCII value hex 57 within the new character set table, and set 2 may be at ATASCII value hex D7 (hex 57 plus hex 80) if desired. You may feel free, of course, to assign your character sets in any manner you desire.

Therefore if you would print these two characters side by side on the screen, it would become effectively a 20 character per line mode, with the resultant 10-combination treated as the 1-bit in the mode 0 example and the 00-combination as the 0-bit in the mode 0 example, forming a recognizable W in the process.

Note also that you may want to design these new character sets in a 7 by 7 matrix starting the upper left hand corner of the bit-pair set to allow at least one blank row and column between each of the new characters. (This was not done in the example).

Thus many combinations of colorful characters may be formed using this technique, allowing the user of the A1200 additional flexibility for his programs.

APPENDIX C - SERIAL BUS PERIPHERAL HANDLER LOADING, LINKING, USE

This appendix contains technical details regarding the serial device handlers. It is not written for the general user however contains information essential for use by a developer of peripherals for the A1200 system.

A1200 HANDLER LOAD AND RELOCATION DURING POWER-UP PROCESSING

[Note: The loading procedure described here is also used to load handlers when the loading is application-specified after power-on has completed. The only differences are where in RAM the handler is loaded, and handling of loading errors. Accordingly, this single section deals with both loading operations. The major point of view is toward loading at power-on time to the MEMLO boundary; differences for the application-initiated load are noted. See section titled APPLICATION-INITIATED LOAD for more on this subject.]

After a peripheral responds to the Type 3 Poll, the OS will then compare the sum of MEMLO (02E7 and 02E8 hex) and the size of the handler to be loaded (DVSTAT and DVSTAT+1, 02EA and 02EB hex) to MEMTOP (02E5 and 02E6 hex) to determine that there is room to load the handler. If there is insufficient room, the handler will not be loaded, and the OS issues a Null Poll command (section on A1200 POLLING DURING POWER-ON) and proceed with further Type 3 Polling (POWER-ON COLD STARTstep 9).

Otherwise the peripheral handler is loaded, starting at MEMLO and proceeding until the load is completed. (Note: the load address may also be application specified; see section on APPLICATION-INITIATED LOAD.) The loading operation is achieved using the Operating System's relocator (see Appendix D). A call to the relocator is made, specifing all parameters needed:

- o Loading address. This is either a copy of MEMLO, or the application-supplied load address. Before handing this value to the relocator, the OS Type 3 Poll process insures that it is even-valued by adding one if it is found to be odd; (Note: a "Bug" exists in the 6502 processor where a JMP indirect instruction will fail if the two-byte indirect pointer is relocated across a page boundary. This may be avoided by placing all indirect pointers on even addresses; since loading always occurs on even boundaries, the pointer will never cross a page boundary.

- o Zero-page loading address. The handler will not load into page zero. This address is set to 80 hex;

- o Address of get-byte subroutine described below.

The get-byte subroutine supplied to the relocator calls on SIO to get the handler relocatable object records from the peripheral and then pass them a byte at a time to the relocator. The records is read from the peripheral in numbered blocks of 128 bytes each, numbering starting at 0 and going as high as needed (255 max). The cassette buffer is used for storing each block as it is being fed to the relocator. The final block may be unfilled; get-bytes will stop from the relocator when the End record is processed, so the remaining portion of that block is ignored.

Serial port load commands are as follows:

- o Device address taken from Poll response;
- o Load command "&" (26 hex, 038 decimal);
- o Aux1 = block number to be loaded;
- o Aux2 = undefined (must be ignored by peripheral);
- o Appropriate checksum.

If the peripheral is asked to supply a block whose number is out of range, it will either NAK or not respond (preferable action is no response). The reader will then pass error status to the relocator which will pass the error on to the caller. At power-on, the caller is the OS Type 3 Poll routine, which responds to the error by ignoring this peripheral and continuing polling for other peripherals When loading is being called by an application, the IOCB is closed and error 133 (Device Not Open) is returned to the application.

During cold-start processing, the OS ignores all parameters returned by the relocator when relocation completes except the error status. All relocating loader errors produces the results of the preceding paragraph.

No check will be made that the handler is actually relocated properly. Some errors will be detected by the relocator; however it is the responsibility of the peripheral designer to create a proper device handler.

The handler must occupy contiguous RAM, starting at the load address. No restriction is placed on the use of this RAM area (it may be code, variables, or data) except that the linkage conventions (section on INIT. AND LINKING DURING POWER-ON) must be followed.

When loaded under applications request, the size of the area allocated for the handler can be larger than the minimum required, and the handler may make use of this extra RAM as needed (see section titled LOAD,RELOC.,INIT.,USE). When loaded at MEMLO during power-up, the handler will specify its RAM needs (section on INIT. AND LINKING DURING POWER-ON and section titled SYSTEM RESET REINIT).

A1200 INITIALIZATION AND LINKING DURING POWER-UP PROCESSING

[Note: The handler initialization and linking procedures during power-up processing are very similar to those during warm-start reinitialization and application-initiated handler loading. Therefore, this section serves for all processes. It is written in terms of the power-up sequence, with occasional test conditions for the warm-start variations. The major differences in this procedure between power-up and warm-start are described in section titled SYSTEM RESET REINIT. Application-initiated load is described in section titled LOAD, RELOC., INIT., USE.]

Once loaded, a handler will be linked into the system in three ways:

- The handler's RAM usage will be declared;
- The handler's name and linkage table address will be entered into the handler table;
- The handler's linkage table will be entered into a linked-list of known loaded handlers for System Reset (warm start reinitialization).

The handler will have a linkage table at its load address. This table contains the following:

OFFSET	CONTENTS
0 - 14	Standard handler entry vectors (reference A):
	OPEN vector;
	CLOSE vector;
	GETBYTE vector;
	PUTBYTE vector;
	GETSTAT vector;
	SPECIAL vector;
	initialization code JMP;
15	Linkage table checksum;
16 - 17	Handler size in bytes to add to MEMLO.
18 - 19	Handler linkage table chain forward pointer;
20 - 21	Zero (reserved for future expansion).

Byte 15 (checksum) is calculated such that the wrap-around-carry sum of bytes 0 through 17 is FF hex (one's-complement negative zero); it is used by the operating system to check the integrity of the linkage table during system reset (warm start) reinitialization. Since bytes 0-17 may vary depending on load address the checksum will be calculated after the handler is loaded. Bytes 18-19 point to the handler linkage table loaded next. If this is the last handler loaded, this forward pointer is null (zero).

The initialization process for a newly loaded handler immediately follows its loading:

[Note: All steps of this process are performed by a subroutine which is normally used as part of the OS process of linking new handlers into the system. This subroutine can be called by other system routines; the calling sequence is discussed in section titled SUBROUTINE INTERFACES. As used during power-up loading of handlers following Type 3 Polling, the MEMLO parameter used in step 4 is set on, indicating that the handler's size is to be added to MEMLO.]

1. The OS adds the new handler linkage table at the end of the linkage table chain. This is done by starting at the head of the chain, CHLINK (in the OS database) and following the pointers until a null (zero) pointer is found. For each linkage table in the chain (except the last), the checksum is checked to verify the integrity of the linkage table; checksum failure results in failure to initialize the newly loaded handler, and the rest of this initialization procedure is bypassed. No error is reported out of the OS during coldstart (in this case, polling continues with a Null Poll, followed by Type 3 Poll, POWER-ON COLD START step 9). In the case of a non-OS caller, the error is indicated to the caller by returning with carry bit set. If the checksums are OK, the address of the new linkage table, which is the load address of the handler, is placed in the null pointer which was at the end of the chain. Then the pointer in the new linkage table is nulled (zeroed);

2. The OS loader will then JSR to the handler initialization code;

2a. The handler will initialize itself, optionally utilizing the handler table entry subroutine in the resident OS (section titled SUBROUTINE INTERFACES). Errors occurring in the linking process will produce linking failure (discussed below). The handler will initialize itself as follows:

2b. Call the OS-resident handler table entry subroutine to add a handler entry for this new handler;

2c. Optionally establish the linkage table handler size. The handler size could simply have been loaded into the linkage table at relocation time, in which case the handler initialization procedure now takes no further action. Alternatively, the handler can calculate the size and insert the result during this first initialization. The handler will calculate this size only once, and supply the result to the operating system in the linkage table at this point during power-up initialization. The handler will not modify these bytes in the linkage table at any subsequent time. The OS flag WARMST can be used to distinguish power-on initialization from subsequent warm-start reinitialization. The handler size need not be returned to the OS if WARMST is nonzero. If the

handler calculates its RAM needs, it is responsible for insuring that the resulting addition to MEMLO does not exceed MEMTOP. Also, it is the handler's responsibility to ensure that the size set by the handler is even-valued. It is safe if the calculated size does not exceed the size reported by the Type 3 Poll (section on PERIPHERAL POLL DURING POWER-ON);

2d. Return with Carry bit clear if there was no init error; otherwise, return with carry set;

(Note: the handler init need not save any 6502 registers.)

3. If the handler initialized unsuccessfully (Carry returned set) the new handler linkage table is removed from the linkage table chain using the routine described in section titled SUBROUTINE INTERFACES for that purpose, and the handler installation is terminated. In this case, none of the following steps is performed; no error indication is given out of the OS during coldstart, and polling continues with a Poll Reset followed by further Type 3 Polling. In the case of a non-OS call to this initialization process, the error is returned to the caller by returning with the carry bit set.

4. If the handler initialization was successful (Carry returned clear) the OS will then check the parameter to see what mode of initialization is being performed, to determine whether or not the handler size should be added to MEMLO. If the parameter is set, then the handler size should be added to MEMLO. If the parameter is not set, the handler size should not be added to MEMLO. In the latter case, the handler size entry in the handler linkage table is cleared to zero;

5. The handler size is added from the handler linkage table to MEMLO (02E7 and 02E8 hex);

6. The linkage table checksum is calculated and inserted into the table. This is done by first zeroing the checksum; then calculating the checksum of the first 18 bytes of the table; then storing the one's complement of the resulting sum as the calculated checksum of the linkage table.

In step 2, above, the handler may interrogate the system flag WARMST to determine the time of initialization. WARMST (0008 hex) is zeroed by the OS at the beginning of power-on processing. Unless modified by other code in the system, WARMST remains zero until the [SYSTEM.RESET] key is pressed, when it is set to FF (hex). Should this be unacceptable to the handler initialization, the handler should keep an internal variable to keep track of which initialization is occurring.

Handler table overflow error is a possibility in step 2b, above. The handler should return with Carry set to indicate initialization failure, unless it performs some reasonable error recovery.

Note: The above procedure uses DVSTAT+2 and DVSTAT+3 (02EC and 02ED hex).

A1200 APPLICATION-INITIATED LOAD

Most of the loading and initialization processes of an application-initiated load are identical to those used for power-up load. Those differences between the two (MEMLO handling) which affect the handler are discussed in section on INIT. AND LINKING DURING POWER-ON. The major difference lies in the polling processes used.

A1200 APPLICATION-INITIATED OPEN POLL (TYPE 4)

When an application calls CIO to perform an open, the following occurs:

1. The OS flag HNDLOD (02E9 hex) is interrogated to determine whether the application desires a Type 4 Poll for the device being opened. HNDLOD=zero means conditional poll (step 3); anything else means unconditional poll (step 2);

[Note: the operating system sets HNDLOD zero at power-on and system reset. If the application does not modify HNDLOD, conditional poll will always be selected by any OPEN.]

2. If unconditional poll is selected, a Type 4 Poll (see below) occurs. If no peripheral answers, step 7 is performed. If a peripheral answers, its 4-byte answer is returned by CIO to the application in DVSTAT through DVSTAT+3 (02EA through 02ED hex) (proceed to step 6);

3. If conditional poll is specified, CIO checks for the device in the handler table. If an entry is found, the handler already exists and normal open processing continues. Proceed to step 5;

4. If conditional poll is specified and no handler entry is found, a Type 4 Poll is issued. Everything proceeds from here as in step 2;

5. If no poll was issued, this fact is flagged to the calling application by setting DVSTAT and DVSTAT+1 (02EA and 02EB hex) to zero. I/O status returned indicates either successful OPEN, or open failure for any of the standard set of possible reasons;

6. If a poll was issued and successful, the IOCB is "provisionally" opened. This includes all normal CIO OPEN processing, but includes none of the handler open processing since the handler is not loaded at this time. The IOCB is marked "provisionally" open in the following ways:

   o The handler table pointer ICHID is set to 7F (hex);

- The put address ICPTL,ICPTH is set pointing to the OS-resident application loader routine;

- ICAX3 contains the device name for the handler loader table;

- ICAX4 contains the device serial address for loading.

Normal status (01) is returned following a provisional open, and DVSTAT through DVSTAT+4 (02EA through 02ED hex) contain information needed by the application to provide RAM for the handler load which will follow (see below);

7. If a poll was issued and no device answered, the IOCB is not opened and error 130, Non-existent Device, is returned.

The OS flag HNDLOD (02E9 hex) is set to zero each time CIO returns to the application, regardless of what call was made or the results of the call.

Following a "provisional" open the application must check the DVSTAT bytes to determine the need to allocate an area for the handler which is to be loaded. The application must set aside an area, on an even address, at least as large as the handler size specified in DVSTAT and DVSTAT+1 (02EA and 02EB hex). Then the application must place the address of this area in DVSTAT+2 and DVSTAT+3 (02EC and 02ED hex) and the length of the area in DVSTAT and DVSTAT+1 (02EA and 02EB hex). (The application may allocate the minimum area by leaving DVSTAT and DVSTAT+1 alone.) If the even starting boundary cannot be assured by the application, it must allocate one more byte than it reports in DVSTAT/DVSTAT+1. The application signals the completion of these steps by setting the flag HNDLOD (02E9 hex) nonzero.

The handler load occurs automatically when the application calls CIO to perform any I/O operation except CLOSE via the "provisionally" open IOCB, when HNDLOD is nonzero (the CLOSE command will simply close the IOCB without loading the handler). The steps taken by CIO is as follows:

1. The IOCB is checked to see is it is provisionally open. If it is not, normal I/O processing continues;

2. If the IOCB is provisionally open, the flag HNDLOD is checked. If the flag is zero, error 130, Non-existent Device, is returned;

3. If the IOCB is provisionally open and HNDLOD is nonzero, the handler is loaded (using the procedure of section on HANDLER LOAD AND RELOCATION DURING POWER-UP) and linked (using the procedure of section on INIT. AND LINKING DURING POWER-ON). Prior to the load, the load address in DVSTAT+2 & DVSTAT+3 is forced even. The initialization process is called with the MEMLO parameter off, indicating that the handler size is not added to MEMLO;

4. If the loading or initialization fails, the IOCB is closed and error 130, Non-existent Device, is returned;

5. If the loading and initialization succeeds, the IOCB is modified to indicate it is properly opened:

   - Handler ID, ICHID, is set to point to the proper handler table entry. If the entry is not found, error 130, Non-existent Device, is returned, and the IOCB is closed;

   - Normal CIO OPEN processing is performed, thus filling the IOCB properly, including the put address ICPTL, ICPTH which is set to point to the handler put-byte entry. Additionally, the handler OPEN entry point is called by CIO.

6. CIO completes processing of the I/O command originally called by the application.

[Note: it is extremely important that the application not modify the handler once it has been loaded. Users of high-level languages such as BASIC or PASCAL must remain aware of how the language environment, particularly the language memory usage, may affect the handler. DOS 2 users must be aware that the DUP overlays memory which could contain I/O handlers. [SYSTEM.RESET] "uses" loaded handlers via the process of reinitialization; therefore, system reset processing could fail if any loaded handlers have been modified. Also note that unpredictable results will occur should the handler be loaded more than once by an application.]

A1200 SYSTEM RESET (WARM START) REINITIALIZATION

This section describes the sequence of events taken by the operating system during system reset (warm start) reinitialization. This consists of actions which have existed in the 400/800 revision B operating system plus new operations which are the A1200 enhancements being described in this document. Only that degree of detail needed here is included.

1. The OS sets the warm start flag WARMST (0008 hex) to FF hex;

2. Certain variables in the OS database are cleared to zero. RAM outside the OS database is left untouched. In particular, the handler table and all IOCB's is zeroed;

3. MEMLO (02E7 and 02E8 hex) is set to 0700 hex;

4. OS resident handlers is initialized and entered into the handler table;

5. The application cartridge "A" is initialized, if present;

6. Cassette or disk initialization occurs (CASINI or DOSINI). At this time, the DOS updates MEMLO by adding its size, and any handlers within the DOS are initialized and entered into the handler table;

7. Upon return from the cassette-booted or disk-booted reinitialization, the operating system will reinitialize all handlers which have been loaded into RAM. The procedure is described in detail below;

8. The OS will start the cartridge or jump through DOSVEC.

To perform the initialization of loaded handlers (step 8 above), the operating system will proceed as follows:

1. The internal pointer CHLINK is checked to see if any handlers have been loaded. This pointer is null (zero) if there are no loaded handlers, or it points to the linkage table of the first such handler;

2. If a loaded handler exists, its linkage table checksum is calculated and checked. If the sum is not two's-complement zero, the handler has been destroyed and this portion of the OS initialization terminates (no error is reported);

3. If the linkage table checksum is OK, the handler is re-linked and re-initialized according to the procedure of steps 2 through 6 of section on INIT. AND LINKING DURING POWER-ON; the MEMLO parameter is set on so that the handler size will be added to MEMLO;

4. If an error occurs while re-initializing the handler, this portion of OS initialization is terminated (no error is reported);

5. The forward pointer for the handler linkage table chain in this handler's linkage table is checked. If it is null (zero), this phase of initialization is complete. If it points to another handler, steps 2 through 5 are repeated for each handler in the chain.

A1200 SUBROUTINE INTERFACES

Three subroutines are added to aid the initialization process for loaded handlers. The first searches the handler table for an empty slot and makes the entry for the handler. The second follows the handler linkage table chain to remove a handler from the chain. The third performs initialization processing for a loaded handler.

All three routines are called via JSR to the appropriate entry vectors (below). All parameters are passed through the machine registers.

The entry addresses for these routines is as follows:

E489 hex	Handler Entry Routine
E48C hex	Handler Linkage Removal Routine
E48F hex	Handler Initialization Routine

Parameters for this routine are provided in the machine registers.
The routine is written for use by the OS handlers.
The parameters it uses are passed as follows:

X: Handler name;

A: High byte of linkage table start address;

Y: Low byte of linkage table start address.

This routine searches the handler table from start to the first empty slot. If no empty slot is found (the table is full), the carry is set on return to the handler to indicate an error. If a duplicate handler name is found, a different error is returned (also see below). If neither of these error occurs, the handler entry is inserted into the table at the first empty slot.

If the entry was successful made, the Carry bit is cleared on return to the handler.

If the handler table is full, error return is indicated by setting the carry bit. This error is distinguished from the duplicate-entry error by also setting the Negative bit. The registers are undefined when this return is made. The handler should not proceed with initialization; see section on INIT. AND LINKING DURING POWER-ON.

If there is a duplicate handler name in the table, the condition is indicated to the calling handler by returning with Carry set and Negative clear. In this case the A and Y registers are returned to the handler unchanged from the call, and the X register is an offset, relative to the first byte of the handler table, pointing to the second byte of the 3-byte table entry where the matching device name was found. The handler has the choice of discontinuing initialization, replacing the older handler entry, or chaining itself in (replacing the old entry but saving it in order to call the older handler whenever an I/O call belongs to the older handler).

This routine undoes the handler linkage performed by the HANDLER ENTRY routine. Its parameters are also passed to it within the machine registers. The parameters required are as follows:

A: High byte of address of handler linkage table;

Y: Low byte of address of handler linkage table.

This subroutine searches the handler linkage table chain for the linkage table having the address passed in A and Y. The linkage table checksums is computed and checked along the way to verify the integrity of the chain. When the proper linkage table is found, the handler size is checked to determine whether or not the handler was loaded at MEMLO. If the handler size is nonzero, the handler was loaded during power up at MEMLO, and it is illegal to remove it. In this case, the subroutine returns with the Carry set. Otherwise, the linkage table is removed from the chain by copying its forward chain pointer contents into the forward chain pointer of its predecessor in the chain.

If the chain search terminates either by finding the end of the chain (null pointer) or a bad linkage table, no action is taken and the Carry bit is returned set to indicate the error. Carry is cleared to indicate that the table was found and removed. The other registers are undefined upon return.

This subroutine is supplied to allow an application to request removal of a previously loaded handler when it is no longer needed or when the RAM must be reclaimed. It is suggested that the handler CLOSE routine check the flag HNDLOD (02E9 hex); it may be set nonzero by the application before CLOSE to indicate that the application wishes the handler unloaded. The handler is responsible for removing itself when unloading is requested: the handler table entry should be deleted, and the linkage table must be removed from the chain. The IOCB byte ICHID may be used to find the handler table entry, and this subroutine is used to remove the link from the chain. [Note: The OS variable COLDST is interrogated by this routine to determine when the caller is the operating system itself at cold start time. In this case, the handler is unlinked even though it is loaded at MEMLO.]

Note that, except as described in the paragraph above, the handler must NOT remove itself if it has been loaded at MEMLO. This is the reason that this subroutine checks the handler size for application-loaded handlers. If the handler receives error status from this subroutine, it should NOT remove itself from the system (except it is still permissible to remove the handler table entry).

Handler table removal is done by zeroing the device name byte in the handler table.

INITIALIZATION SUBROUTINE

An INITIALIZATION subroutine entry point is included in the OS to provide the handler initialization function to be easily performed when handlers are loaded by a non-OS routine, for example by AUTORUN.SYS.

The INITIALIZATION subroutine performs all the tasks (steps 1-6) for initialization described in section on INIT. AND LINKING DURING POWER-ON. This routine requires the following parameters to be passed to it in the machine registers:

A: High byte of address of handler linkage table;

Y: Low byte of address of handler linkage table.

In addition, the Carry bit must be set by the caller to indicate whether the handler size should be added to MEMLO: Carry set on means the subroutine allows the adding of the handler size to MEMLO. Carry clear means the handler size is zeroed, thus suppressing its addition to MEMLO.

This subroutine returns to its caller with the Carry set if a linking error occurred (and the linking is not performed). Carry will be clear if linking was successful.

The A1200 Operating System ROM includes a subroutine which can be used to load certain types of object code.

Due to the limited amount of space available in the OS ROM, only a limited amount of error checking can be done. Therefore, a strict set of rules has been established for the format of object code which can be relocated using the facilities of this built-in loader. If the format is not properly followed, you will obtain unpredictable results. This loader is not accessible to user programs. It is only described here to provide peripheral designers with data appropriate for correct structure of the handler object code.

FORMAT OF THE LOADER PARAMETER BLOCK

Before executing the relocating loader subroutine, the OS provides the loader with certain information. This is a table of data located at hexadecimal 02CF within the OS RAM area. A total of 5 bytes of data must be provided. They are organized as shown here:

low byte	GETBYTE ADDRESS	$02CF
high byte		
low byte	LOAD ADDRESS	$02D1
high byte		
one byte	ZLOAD ADDRESS	$02D3

The interpretation of the bytes in this table is as follows:

The GETBYTE address is a two byte address of the entry point for the Get Byte routine. This may refer to an existing GETBYTE routine for a peripheral already supported with coresident code. SUBROUTINE" below.

The LOAD ADDRESS parameter specifies the base address from which the calculation of actual object code placement and cross reference will be made. For example, if the relocatable object code was all assembled to be relocated with respect to its own location 0000 and the LOAD ADDRESS specifies 00 (low byte), 90 (high byte), then the code will be loaded beginning at $9000. All relative relocatable address references will then be changed to reference the new code location at $9000.

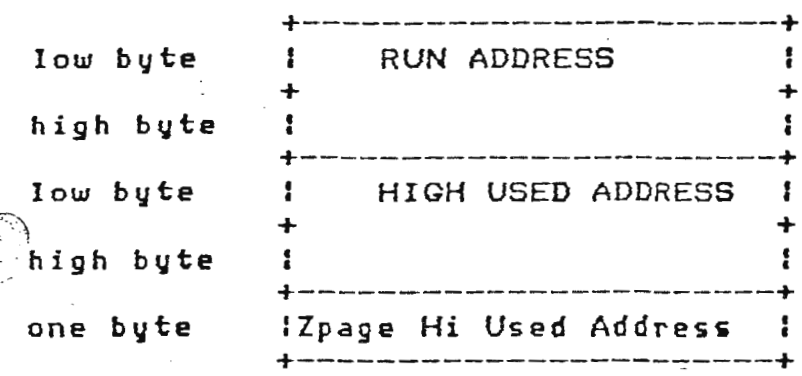
The ZLOAD ADDRESS is a one byte zero page address which is used as the base address for the relocation of any zero page references used in the relocatable code. Any references to page zero variables are adjusted during relocation by adding this ZLOAD ADDRESS as an offset to the relocatable address. This forms the actual load address for the variable and its references.

LOADER-TO-USER PARAMETER BLOCK

Before the OS called the loader routine, it had to provide a block of parameters to give the loader various information. The loader, in turn, provides a return set of parameters.

These parameters will allow the OS to determine where the next relocatable subroutine may be loaded, if desired, to allow a sequenced loading of many such routines. It also provides you with the absolute RUN address once the relocation has taken place to allow a jump into the now resident routine.

Here is a diagram showing the way the table appears in memory. It begins at hexadecimal address $02C9.



RUN ADDRESS is the execution entry point. It has been calculated by the Loader as the absolute address which was specified from the data in the END record. If the RUN ADDRESS is zero, then you did not specify a run address in the END record. (Record structure is covered in the next section).

HIGH USED is the address of the next available memory location above that which has just been used by the loader. If there are multiple relocatable routines to be loaded, the information in the low and high bytes of this parameter may be moved directly into the User-to-loader parameter block to direct where the NEXT routine is to be loaded. The equivalent locations within that parameter block are $02D1 (low byte) and $02D2 (high byte) of the LOAD ADDRESS.

Zpage Hi Used is the address of the next available zero page memory location above that which has just been used by the loader. If there are multiple relocatable routines to be loaded, the data in this parameter may be moved directly to location $02D3, the ZLOAD ADDRESS. This allows a chain of relocatable files to dynamically configure themselves in the memory using the loader's output as the input of the next loader call.

RECORD STRUCTURES

The relocatable object file consists of a sequence of one or more segments. An object segment consists of a single TEXT record followed by one or more INFORMATION records. The format of the TEXT and INFORMATION records is discussed in the sections which follow.

The Loader processes the data obtained by the GETBYTE routine as object segments. The TEXT record is a sequence of machine instructions and data. The INFORMATION record(s) associated with each TEXT record specify exactly which of the bytes within the TEXT record will have to be modified in order to relocate the code to its intended location.

The relocation process begins with the Loader taking a TEXT record and loading it into the memory area at the absolute address calculated from the load address provided. (There may be many TEXT records in any single relocatable object file). Then the loader reads the next record to see if it is an INFORMATION record. An INFORMATION record will show the loader which bytes in the loaded code will have to be modified. If there is no INFORMATION record associated with this TEXT record, no modification takes place.

This will occur if you have written the machine code to be fully relocatable...that is, no matter where in the memory it is placed, it still will execute the same function. It would also occur for TEXT segments containing strictly data, when it doesn't matter where the data resides as long as it can be referenced. Such a code segment might be one containing an alternate character set or such data.

If the TEXT record does include address references which must be relocated, the INFORMATION records which cause the modification must immediately follow that TEXT record in the record grouping. You may therefore consider one TEXT record and a number of INFORMATION records as though it is one complete segment.

The entire relocatable file processed by the Loader will consist of any number of TEXT/INFO record groupings, then a final record known as the END record. The record file begins with a TEXT record and ends with the END record. The Loader exits to the calling routine immediately after the END record has been processed.

RECORD FORMAT DEFINITION

The loader expects the input records to be formatted in a specific manner. The individual formats for the TEXT, INFORMATION, and END records are given below. The common element between them is the first byte of the record, which specifies what type of record is to be processed. The first byte of the record is known as the Type ID. As a summary, the various Type ID's associated with each record type are as follows:

TYPE ID	RECORD TYPE
00	TEXT - Contains Non-zero-page Relocatable Text
01	TEXT - Contains Zero-page Relocatable Text
02	INFO - Points to non-zero page low byte references to non-zero page data in a text record
03	INFO - Points to zero page low byte references to non-zero page data in a text record
04	INFO - Points to non-zero page single byte reference to zero page address within a text record
05	INFO - Points to zero page one byte reference to zero page data in a text record
06	INFO - Points to non-zero page word references to non-zero page data in a text record
07	INFO - Points to zero page word references to non-zero page data in a text record
08	INFO - Points to non-zero page high byte references to non-zero page data in a text record
09	INFO - Points to zero page high byte reference to non-zero page data in a text record
0A	TEXT - Contains absolute, nonrelocatble object code
0B	END - Is an END record

All of these various record types and pointers are illustrated by example in the sections on TEXT RECORDS, INFORMATION RECORDS and END RECORD below.

TEXT RECORDS

A TEXT RECORD is a group of bytes containing machine language instructions and data. It will be loaded intact to a specific area of memory, then the Loader will modify some or none of the bytes AFTER placement into RAM according to instructions provided in the INFORMATION records which immediately follow this TEXT RECORD.

There are three types of TEXT records which may be specified:

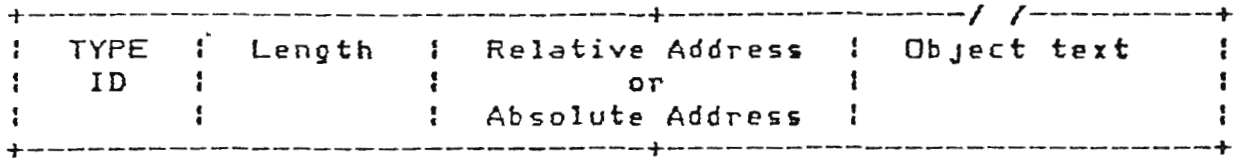
A. A record containing non-zero page relocatable text. This is data which is loaded into an area in an area other than zero page ($0100-$FFFF) somewhere and whose address references must be modified to reflect the actual area into which it, and its corresponding zero page segment, have been loaded.

B. A record containing zero page relocatable text. This is data which is loaded into an area within zero page ($0000-$00FF) somewhere and whose address references must be modified to reflect the actual area into which it, and its corresponding non-zero page segment, have been loaded.

C. A record containing Absolute text. This type of data does not need any adjustment to its address references. A TEXT record of this type will not have any INFORMATION records following it. (INFO records specify the relocation data and this type of text does not need any.)

TEXT RECORD FORMAT

Here is a representation of the content of the typical TEXT record:



low byte high byte

1 byte 1 byte 2 bytes 0-253 bytes

The TYPE ID field for a TEXT record will contain one of the three following values. For a complete description of the meaning of each record type, see TEXT RECORDS above.

VALUE TYPE OF TEXT RECORD

00	Non-zero page relocatable text
01	Zero page relocatable text
0A	Absolute text

The LENGTH field for a TEXT record will have a value from 2 to 255. It is computed as the total count of bytes contained in the record counting from the first byte following the Length byte to the end of the record. (A complete text record therefore can consist of a minimum of 4 bytes, to a maximum of 257 bytes).

The ADDRESS field specifies either an Absolute or a Relative Address.

If it is an Absolute Address record type, the object text contained in this record is to be loaded to memory at the starting address specified in this absolute address field. Each byte in the text is then to be loaded into the next higher address until the entire record has been loaded.

If it is a Relative Address record type, the object text contained in this record is to be loaded to memory at the specified address RELATIVE to the starting address of the relocatable code. The address field is specified as relative to starting address 0000 which is assumed to the the first location within a code segment. The actual address to which this code will be loaded is calculated by the Loader by adding the LOAD ADDRESS offset (See USER-LOADER PARAMETER BLOCK) to the relative address contained in the record itself. The relative address is the 16-bit offset from the beginning of the actual program so the placement in RAM will therefore be relative to the starting location which you specified in the parameter block.

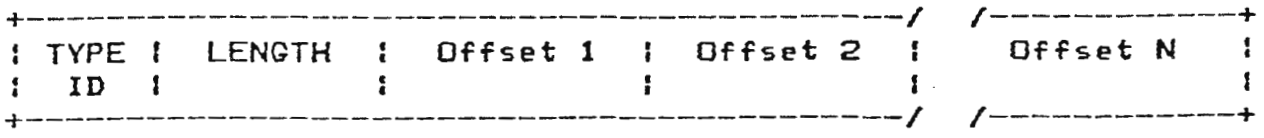
INFORMATION RECORDS

INFORMATION records are the modifiers for the TEXT records. There may be no information records or many of them.

There are two basic types of information records: those which reference single byte data or low byte of an address, and those which reference the high byte of an address reference.

LOW BYTE, ONE BYTE, AND WORD REFERENCE INFORMATION RECORDS

The format of an information record which can modify low byte address references, one byte (page zero) addresses or word references (those which modify a 16-bit address and point to the low byte of that quantity) is shown here:



The TYPE ID field will specify the type of reference for which the offset specifies the location. The TYPE ID's for which this format is valid are the following:

TYPE ID	REFERENCE TYPE

02 Non-zero page low byte reference to a non-zero page address. This means that you may have referenced something similar to the following:

```
LDA  #L,NZREF  ;get the low byte of the
               ;16-bit integer assigned
               ;to address NZREF
```

This will be an address relative to the beginning of the relocatable file. If the offset points to the immediate value, it is this value which will be modified when the LOAD ADDRESS low byte is added to it to obtain the actual current load address. The TYPE ID indicates that this instruction is loaded into a non-zero page area.

Example: Code loaded into location $1000,
LOAD ADDR is $0D01,
NZREF is located at relative address 0050

If code is LDA #L,NZREF, then loader sees:

```
$1000    A9        (LDA)

$1001    50   <----- offset points here
```

Load address low byte is $01
Value found at pointer is $50

Loader adds them, replaces value at pointer with $51

03 Zero page low byte reference to a non-zero page address. This is exactly as described for Type ID 02 above except that the code which is to be relocated has been loaded into a page zero area instead. The rest of the explanation remains exactly the same.

04 Non-zero page one byte references to a zero page address. This means that a code segment such as:

```
LDA  ZPAGE1 , where ZPAGE1 is an address
              within page zero,
```

produces a relocatable code. This code, when stored in a non-zero page area, may have to be relocated if ZPAGE1 is a relative address. In this case, the example might show the following:

Example: Assume that the code specified above is loaded at $1000, and ZPAGE1 is zero page address $0045. Also assume that the ZLOAD ADDRESS you specified earlier (see USER-LOADER PARAMETER BLOCK) contains $10.

```
$1000     A5    (LDA ... zpage)

$1001     45    <------  Offset points here
```

The Loader will take the byte at the pointer, add the ZLOAD Address offset, and replace the value at the pointer with the newly calculated relocated value. In this example, $45 is fetched, adds the offset $10, so the relocated address value is $55 stored into location $1001.

05 Zero page one byte reference to a zero page address. This is exactly like the relocation example shown for Type ID 04 above. The only difference is that the code which has been loaded resides in a page zero area and is modified there. In the example, the load address, then, could have been $00A1, instead of $1000. All else remains the same.

06 Non-zero page word reference to a non-zero page address. This means the offset points to the low byte of an object code address reference of code which has been loaded into an area not in page zero.

Example: Code loaded to location $1000, consisting of LDA $1234, loaded as:

```
$1000     AD

$1001     34  <---- offset points here

$1002     12
```

(note that the address $1234 is an address relative to the start of the object code file itself, which starts, relative to itself, at location 0000)

07 Zero page word reference to a non-zero page address. This means the offset points to the low byte of an object code address reference of code which has been loaded into an area not

in page zero.

Example: Code loaded to location $0023, consisting of LDA $1234, loaded as:

```
$0023    AD
$0024    34   <---- offset points here
$0025    12
```

Now that we've gone over the TYPE ID's for 02-07, the other fields in this INFORMATION record can be explained. Recall from above they are: TYPE ID, LENGTH, and OFFSET.

The LENGTH field in this record type specifies the total byte count of the number of OFFSETS which are contained in this record. In other words, it specifies how many of the bytes within the previously loaded TEXT record are to be modified by the Loader using this specific TYPE ID. There will be that number of pointers as a part of this record. The length field may specify a value from 0 to 255.

The OFFSET field specifies a value from 0 to 255, one byte for each offset. This forms a pointer which, when added to the starting address for the text record just loaded, gives the address of the byte which is to be modified per the relocating instructions as illustrated above. As noted, there may be as many as 255 offsets total contained in any INFORMATION record.

SUMMARY OF LOADER PROCESSING FOR WORD, LOW BYTE AND SINGLE BYTE INFORMATION RECORDS

1. The preceeding TEXT record has been read and its object code has been placed into RAM at the appropriate displacement relative to the preceeding relocatable text record.

2. Each offset is used to obtain a data value (either one or two bytes, depending on record type) from the preceeding object text record.

3. The base address (user specified Load Address) is added to the value obtained.

4. The resulting value (one or two bytes, depending on record type) replaces the data value at the specified offset location in the RAM.

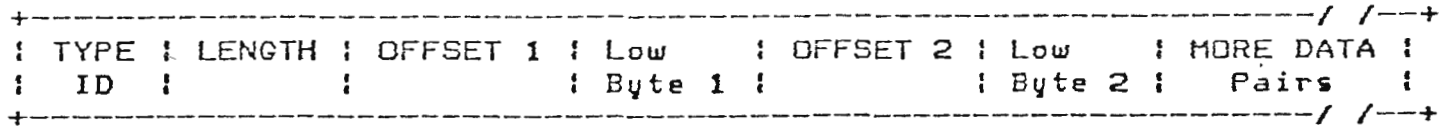
HIGH BYTE REFERENCES IN INFORMATION RECORDS

The record formats for these cases, TYPE ID's 08 and 09, are different from those just discussed. This is due to a different type of data required to calculate the correct address reference.

In the last two cases, (TYPE ID 06 and 07), the pointer specified the low byte of a two byte address. In order to calculate the correct two byte address after adding the offset, and to replace both bytes with the correct relocated address, this single offset pointer is sufficient. The loader will know, in other words, that the high byte immediately follows the low byte, in the next sequential offset location.

However, in the TYPE ID references which follow, the Loader needs more information in order to be able to calculate the correct address.

Therefore the format of the INFORMATION record for TYPE ID's 08 and 09 appears as follows:



If you are referencing the high byte of a relocatable address, the record which contains this reference must also contain a reference to the low byte of that address. This would occur as follows:

Example: The correct way to reference a high byte of a relocatable address...

```
LDA  #L,RADDR  ;get low byte. This must be
               ;located within the SAME TEXT
               ;record as the reference to the
               ;high byte!

STA  TEMP      ;do something with it


LDA  #H,RADDR  ;get the high byte of the
               ;relocatable address

STA  TEMP2     ;do something with it
```

If RADDR is relative location $1234, then the code, when stored at some location (example - $1000), would look as follows:

```
$1000    A9     (LDA ... immediate mode)

$1001    34     <----- TYPE ID 08 offset 1 points here

$1002    8D 22 22  (assumes temp storage spot is absolute
                    address $2222 for example use only)
```

```
$1005    A9    (LDA ... immediate mode)

$1006    12    <----- TYPE ID 08 offset 2 points here
```

What the relative code assembler will do is to organize the code so that both references to the high and low bytes occur within the same 256 byte block of TEXT record. Then a TYPE ID 08 INFORMATION record can be used to reference and modify it as shown above.

The Loader will take the byte pointed to by Offset 1 and treat it as the low byte of a relative address. It will also take the byte pointed to by Offset 2 and treat it as the high byte of a relative address. To this combination relative address, it adds the LOAD ADDRESS (see USER-LOADER PARAMETER BLOCK).

The high byte of the result replaces the high byte of the relative address. If there are any other byte pairs specified as part of this TYPE ID 08 INFORMATION record, they too are processed in the same way.

The low byte of the result is DISCARDED. NOTE, if there is a low byte which must be relocated as well as its high byte, it must be done by a TYPE ID 02 or 03 INFORMATION record. This record MUST FOLLOW that which processed the high byte 08 or 09 type record.

To summarize, then, the record types 08 and 09 are provided for the control of references to the high bytes of relocatable addresses. The only difference between a type 08 and 09 INFORMATION record is that the TYPE ID 08 is used to process TEXT records loaded into non-zero page areas of memory. A TYPE ID 09 record accompanies a TEXT record loaded into zero page.

END RECORD

The END record is the last record processed by the Loader. It has a TYPE ID of hexadecimal 0B.

The END record always consists of four bytes. The first is, as usual, the TYPE ID. The second byte is called the self-start flag. The value in the self-start flag has the following meaning:

VALUE	INTERPRETATION
00	Program execution after relocation is not required. The two bytes which follow the self start flag in the END record are ignored, however must still have been provided to the Loader. The RUN ADDRESS (See LOADER-USER PARAMETER BLOCK) is left as 0000.
01	This tells the Loader that the execution entry point address contained in the END record is an absolute address.

02 This tells the Loader that the execution entry point address contained in the END record is a relative address. To obtain the absolute address, the user provided LOAD ADDRESS is added to the relative address contained in the END record.

The calculated absolute start address (or 0000 if none is required) is placed into the RUN ADDRESS location within the LOADER-USER parameter block. After the processing of the END record, the Loader returns to the calling routine with an RTS.

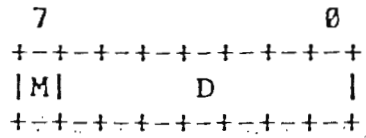
DATA BASE CHANGES FROM REV. B TO 1200

LOCATION	REV.B USE	1200 USE
0000	reserved	LNFLG -- for inhouse debugger.
0001	"	NGFLAG -- for power-up self test.
001C	PTIMOT -- to 0314	ABUFPT -- reserved.
001D	PBPNT -- to 02DE	"
001E	PBUFSZ -- to 02DF	"
001F	PTEMP -- eliminated	"
0036	CRETRY -- to 029C	LTEMP -- loader temp.
0037	DRETRY -- to 02BD	"
004A	CKEY -- to 03E9	ZCHAIN -- handler loader temp.
004B	CASSBT -- to 03EA	"
0060	NEWROW -- to 02F5	FKDEF -- func key def ptr.
0061	NEWCOL -- to 02F6	"
0062	"	PALNTS -- PAL/NTSC flag.
0079	ROWINC -- to 02F8	KEYDEF -- key def ptr.
007A	COLINC -- to 02F9	"
0233	reserved	LCOUNT -- loader temp.
0238-0239	"	RELADR -- loader.
0245	"	RECLEN -- loader.
0247	LINBUF -- eliminated	~~reserved~~
0248-026A	"	reserved.
026C	"	VSFLAG -- fine scroll temp.
026D	"	KEYDIS -- keyboard disable.
026E	"	FINE -- fine scroll flag.
0288	CSTAT -- eliminated	HIBYTE -- loader.
028E	reserved	NEWADR -- loader.
029C	TMPX1 -- eliminated	CRETRY -- from 0036.
02BD	HOLD5 -- eliminated	DRETRY -- from 0037.
02C9-02CA	reserved	RUNADR -- loader.
02CB-02CC	"	HIUSED -- loader.
02CD-02CE	"	ZHIUSE -- loader.
02CF-02D0	"	GBYTEA -- loader.
02D1-02D2	"	LOADAD -- loader.
02D3-02D4	"	ZLOADA -- loader.
02D5-02D6	"	DSCTLN -- disk sector size.
02D7-02D8	"	ACMISR -- reserved.
02D9	"	KRPDEL -- auto key delay.
02DA	"	KEYREP -- auto key rate.
02DB	"	NOCLIK -- key click disable.
02DC	"	HELPFG -- HELP key flag.
02DD	"	DMASAV -- DMA state save.
02DE	"	PBPNT -- from 001D.
02DF	"	PBUFSZ -- from 001E.
02E9	"	HNDLOD -- handler loader flag.
02F5	"	NEWROW -- from 0060.
02F6-02F7	"	NEWCOL -- from 0061.
02F8	"	ROWINC -- from 0079.
02F9	"	COLINC -- from 007A.
030E	ADDCOR -- eliminated	JMPERS -- option jumpers.

0314	TEMP2 -- to 0313	PTIMOT -- from 001C.
033D	reserved	PUPBT1 -- power-up/RESET.
033E	"	PUPBT2 -- "
033F	"	PUPBT3 -- "
03E8	"	SUPERF -- Screen Editor.
03E9	"	CKEY -- from 004A.
03EA	"	CASSBT -- from 004B.
03EB	"	CARTCK -- cart checksum.
03ED-03F8	"	ACMVAR -- reserved.
03F9	"	MINTLK -- "
03FA	"	GINTLK -- cart interlock.
03FB-03FC	"	CHLINK -- handler chain.

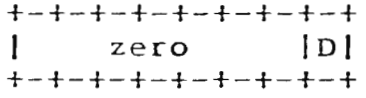
GET CHARACTER DATA FORMATS

Modes 12,13 -- M = color modifier

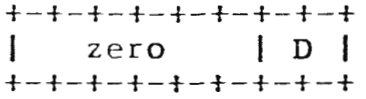


D = truncated ATASCII

Mode 14 -- D = color

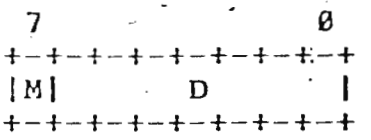


Mode 15 -- D = color



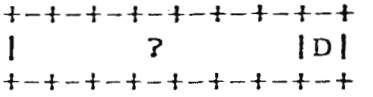
PUT CHARACTER DATA FORMATS

Modes 12,13 -- M = color modifier

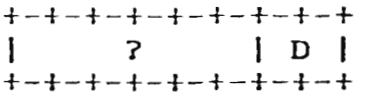


D = truncated ATASCII

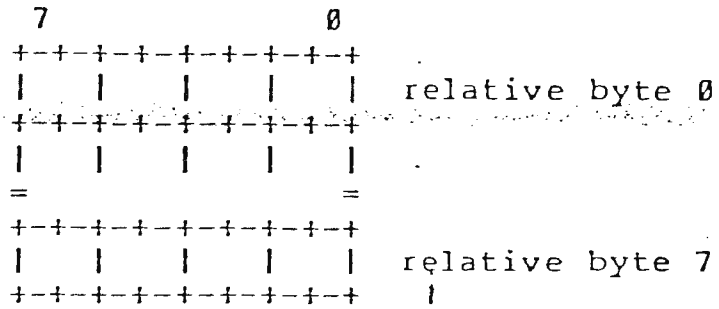
Mode 14 -- D = color



Mode 15 -- D = color



CHARACTER DEFINITION FORMAT FOR MODES 12 & 13



Each 2-bit color specification in the character definition maps to the color registers as shown below:

0 = BAK
1 = PF0
2 = PF1
3 = PF2 if bit-7 of color modifier = 0, or
PF3 if bit-7 of color modifier = 1.

Appendix H (new modes)

Mode #	Horiz. posit.	Vert. w/o sp	Vert. w sp	Colors	Data value	Color reg.	memory reqd. (split)	(full)
12	40	24	20	5	00-7F	*	1154	1152
13	40	12	10	5	00-7F	*	664	660
14	160	192	160	2	0 1	BAK PF0	4270	4296
15	160	192	160	4	0 1 2 3	BAK PF0 PF1 PF2	8112	8138

* See CHARACTER DEFINITION FORMAT FOR MODES 12 & 13.