THE COMPLETE AND ESSENTIAL MAP FOR THE XL / XE

Written by
ANDREW C. THOMPSON ©1994

Published and Distributed by
TWAUG PUBLISHING™ ©1994

PART I

Welcome to a new book. It is based upon Mapping the Atari-Revised by Ian Chadwick. This book has been written to cover the XL/XE Machines only, in Mapping the Atari there is only a small part that had been revised to cover the XL/XE Machines.

I have corrected in this book all incorrect information and errors and I've included a fair bit more information that is not covered in Mapping the Atari. In addition to the MAP section, you will find an XL/XE Operating System source listing with descriptive remarks alongside and there are several appendices that I hope will expand your knowledge and be readily available for future reference. Most of the information in the MAP section that references other sources in Mapping the Atari, you will find amongst the appendices.

I hope this book will help the beginners and intermediate programmers, by explaining the subjects with small straight forward Basic programs. The book should also be a indispensable reference manual for the more advanced programmer.

The author - ANDREW C. THOMPSON

Publishers - TWAUG PUBLISHING

C O N T E N T S

PART I

Subject:	Page:
APPENDIX LIST	
PROGRAM LISTINGS IN MAP	
INTRODUCTION	1-1.5
MAP	2-166
ANTIC	139-147
Attract mode	11
Basic	106-109
Basic disable/enable	87
Basic errors	26
Cartridges	106
Character sets	66-69
Colour	55-59
Console keys	122
Device Control Block	72-75
Display List (DL)	39-41
DMA	38
DOS	87-104
DUP	104-106
Floating Point package	28-30
Graphics priority	46
GTIA	112-123
Handler address tables	77-79
Hardware memory	112-147
Help key	60
IOCB's	79-86
IRQ's	32-35, 42
Joysticks	48-49
Keyboard disable/enable	45
Key definition table	19
Light pen	146
Margins	12
NMI's	31, 36
OS ROM	110-111, 148-165
OS variables, vectors	30
Paddles	47, 49
Page-0 (Loc's 0-255)	2-30
Page-6 (Loc's 1536-1791)	89
PIA	133-139
PMG's	113-121
POKEY	123-132
Realtime clock	5
Scrolling	141-144
Self test	106
Sound	124-128, 132
Stack	30
Top of free memory	17
Variable name table	21
VTOC sector	98
INDEX BY LABEL	IBC
INDEX BY SUBJECT	IBC

PART II

APPENDICES	167 onward

APPENDIX LIST

Here's the appendices which make up the reverse half of the book. Appendix E06 is quite a big one as you can imagine, since it is the Operating System source listing for the XL and XE machines.

APC	NAME:
A01	Basic Keywords
A02	Basic Tokenization
A03	Basic Alterations
A04	Program Improvement
A05	Turbo Basic
A06	Handy Tricks
B01	Sound and Music
B02	Volume-Only Sound
B03	Pokey in Stereo
C01	Character Codes
C02	Number Systems
C03	LSBs and MSBs
C04	Boundaries
C05	Boolean Expressions
C06	Logic
C07	Error Codes
C08	Trigonometric Formulas
C09	Display Modes
C10	Player/Missile Graphics
C11	Display List Interrupts
C12	Boot Process
C13	Graphics 12 and 13
C14	Display Lists

APC	NAME:
D01	Vertical-Blank Processes
D02	Critical Timings
D03	Cycle Stealing
D04	Machine-Language
D05	Vertical-Blank Interrupts
D06	Register/Location Loading
E01	Enhancements and Bugs
E02	Changing a RAM OS
E03	130XE Memory Management
E04	DOS 2.5 Memory
E05	Free Bytes
E06	XL/XE OS Source Listing
F01	Hardware Chips
F02	1050 Specifications
F03	Pinouts
F04	Port Input
G01	Other Software
G02	Atari Support
G03	Glossary
G04	Useful Listings

P R O G R A M   L I S T I N G S

Here are all the listings from the map section of the book only. I've given the page number, location and a small explanation of the listing found there.

PAGE	LOC	DESCRIPTION	PAGE	LOC	DESCRIPTION
5	20	Timing	13	87	GTIA T/Window
13	88	Display Memory	14	88	Bulk memory clearer
14		Picture loading	17	106	Protecting memory
19	121	Unkown editors	21	131	Variable displayer
22	134	Variable valuer	22	131	String clearing
23	136	Line addr. finder	24	138	Program protection
24	140	Strings vs Arrays	26	186	Error detection
31	512	DLI action	34		Hardware timers
36	546	Immediate VBI	46	623	GTIA interraction
47	623	GTIA modes	53	675	TAB setting
57	710	Artifacting	58	710	Artifacting extra
58	712	Rainbow border	61	736	Binary file control
63	743	Hiding low memory	65	755	Inverse flashing
66	755	Flashing cursor	67	756	Character decoding
68	756	Characterset copy	68	756	Character redefining
68	756	Font file creator	69	756	Font file loader
70	764	Key detection	71	764	RAW key converter
74	779	Sector loading	75	779	Formatting
77	794	Null handler	80	832	LIST output toggler
80	832	Return key mode	85		OPENing graphics
86		M/Code drawing	96	3889	DOS 3 corrector
97		File un-deleter	107		Statement token list
113	53248	PMG horizontals	116	53261	PMG without DMA
117	53261	PMG non DMA 2	122	53279	Console speaker
126	53768	8+ Octave sound	127	53768	Filtering
132		Default checks	135		RAM OS
138		RAM Basic/S-Test	140	54272	Screen bending
141	54276	Coarse Hscrol	141	54276	Fine Hscrol
142	54276	Hscrol timing	142	54277	Mode 0 Vscrol
143	54277	Fine Vscrol	144	54277	Split-font mode
146		Diagonal scroller	145		Horizontal sync
146		Vertical sync	152		Mode 8 text
152		360' printing			

INTRODUCTION

Greetings fellow Atarian dudes and Welcome to the biggest brain killing book released in ages. This book as some of you will already know by now is heavily based upon the Revised version of Mapping the Atari, but fear not my indulgent beings for you have not wasted your cool investment, in fact you have made a most excellent step in your life as you know it (or at least as you will know it)! If you own an Atari XL or XE system and you're into programming in a big way then you WILL need this book. It is essential to all levels of programmers.

IN THE BEGINNING:

11 minutes after midnight on Tuesday the 20th of April 1993 saw the beginning of this book, and being a nocturnal kind of guy there couldn't be anytime better! What the next 9 months had in store for me I never would have known although I could guess what I was letting myself in for. The initial phase of writing this book was the MAP itself since this was the main aim of the book, to create a REAL XL/XE MAP reference book. Of course, for this I had to rely heavily on the Revised version of Mapping the Atari, but, at the same time I had to check and compare every location within the MAP section of that book with its appendices 11 and 12; "Addenda and errata to the first edition" and "The XL/XE memory map", but as you might realize, this was only the beginning. To cut a long story short there involved much inclusion of missing material as well as a whole hogwash of extracted information from many books and magazines. Some of these sources include Technical Reference Notes, many of Compute!'s books, De Re Atari, Inside Atari DOS, Your Atari Computer, the DOS 2.5 manual, the XL handbook and a whole host of other magazines and sources including Atari User, Page-6's New Atari User, Megamagazine, TWAUG newsletter and last but not least, a few of my penfriends even supplied me with little titbits here and there. My thanks to you all.

THE APPENDICES:

In addition to the MAP section, you will find perhaps one of the most comprehensive appendices selections ever produced, and whilst there may be 43 appendices, they have been broken down into 7 groups as follows:

Group A: Basic

The first group relates directly to Basic. Here you'll find a complete list of standard and Turbo Basic commands with a short description; some techniques to improving your Basic programs, the tokenizing process, some handy tricks that you can use in your programs and in addition I've included some information as to altering the Basic language itself.

Group B: Sound

The second group is entirely to do with sound. Here you'll find some very useful information which will take you from simple sound affects, through fairly complex music and into the way digitized speech is achieved. You'll also find a fairly straightforward machine-code program to play 2 samples simultaneously. Upgrading your system to stereo is also possible and here you'll find an appendix to do just this.

Group C: Common Reference

Group C is the biggest of all, summing 14 appendices. Here you'll find the explanation of commonly used subjects such as decimal to hex. conversions, DL and PMG boundaries, logic structures etc.. In addition you'll find you may be referencing this group time and time again, since there is a complete list of error codes for Basic and DOS, a chart of trigonometric formulas, a complete list of character codes, display modes and display lists memory usage and assignment etc..

Group D: Machine-Code

This one is probably the most technical, describing everything relating to machine-code and critical timings within the working system. The Vertical Blank process is explained along with information to creating a Vertical Blank Interrupt yourself. There is also some explicitly detailed reference information relating to cycle loss per frame depending on which graphics mode you are using. Also in this group is the most detailed machine-code reference charts you will ever see.

Group E: Memory & OS Listing

Relates to a few selected subjects including information about the Operating System, any bugs it's now overcomed and 130XE memory management. There are 2 appendices giving an in-depth list of correct DOS 2.5 addresses and free bytes in your machine depending upon the programming environment. Last, but not least you'll find a complete XL/XE Operating System source listing with descriptive remarks alongside.

Group F: Hardware

Involves information at the hardware level, including descriptions of the hardware inside the computer and the specifications of the 1050 disk drive. You'll also find some information relating to the use of the joystick ports (for I/O) and the pinouts of the various ports connected to the Atari.

Group G: Miscellaneous

The final group explains multiple uses for various items of software, it gives a list of presently supporting companies still alive and kicking and a glossary of any terms used in this book that you might not be familiar with. The very last appendix of this group and indeed the book contains some program listings that you might have some use for.

Well, this is my first ever book and to be honest with you I almost took on too much. You see, as I was creating the MAP section I was writing down any relevant appendices that I would like to include in the book. Of course, I went about this by including comments throughout the MAP such as "...see the so-and-so appendix" whilst jotting a small note down on paper about what appendix I now had to write! I kept this up throughout the MAP and got just a bit carried away... I had an A4 sheet of paper full of appendices names and comments and stuff I had to put in each one, it's just as well I never lost that sheet eh!? Anyway, looking at the work I had to do a few months ago wasn't very funny, but now I am very pleased with myself it is finally finished. Little did I realise when I had all those appendices to write that I still had to totally re-write 2 decent index's and fully error-check the book because Mappings index's were for the old map (one serious letdown). Even a lot of its programs wouldn't work as shown. But you shouldn't find that with any of the listings in this book, since they have all been typed and RUN, and knowing that they work fine they were then LISTed to disk and merged directly into the books files, thus avoiding any typing or editing errors.

A little torment:

Still on the subject of problems, some of the appendices proved to be a right pain in the neck to write. One such appendix is to do with CYCLE STEALING. I used Technical Reference Notes and De Re Atari as reference, but they were just not accurate enough to obtain a proper explanation. Another one is the MACHINE-CODE reference appendix. For me to assemble the illegal OP-CODES into a table alike the standard ones I had to know how many cycles each illegal instruction took to process. The sad thing is that this information did not exist (until now). I had to work these out myself and the way I went about this was quite unique I think. I wrote a small assembly program as such:

```
10      *=$600
15      LDA #$00
20      STA 710
25 L    LDA  $D40B
30      CMP #$30
35      BNE L
40      LDA #$FF
45      LDX #$00
50      STA  $D40A
55      JSR W           ;KILL TIME TO
60      JSR W           ;SHOW FLYSCAN
65      STA  $D018
70      NOP             ;TIMED INSTR.
75      LDX #$00
80      STX  $D018
85      STX  $D40A
90      JMP L
95 W    RTS
```

What it does is to bring the flyscan to a clear and visible area on the screen. It then places a small coloured line of a particular length. This small length of colour can be considered as 2 machine-cycles, now to time all the illegal instructions you must firstly make chalk marks on your TV screen at the point where the colour ends. Repeat this process for not just one NOP instruction, but for 2, 3 and even 4 of them. This way the chalk marks on your screen will represent timed lengths of 2 cycles, 4 cycles and so on. You can now replace the NOP instruction/s with any single illegal instruction to time it. You should note that the chalk marks are NOT at regular distances from each other, you needn't concern yourself too much with this phenomena but if you really want to know, then consult the CYCLE STEALING appendix. To type illegal instructions replace the NOP's with a line like: .BYTE $BF,$FF,$FF. Anyhow, I'm sure you understand the method.

And now, I've aired my mind and I've nothing much else to say. Hmmm, A thought just occured to me about how I used to think games were made (a long time ago). Good grief....Get a load of this:

```
create man1; red shirt, blue trousers
position man1 at screen-centre
make man1 wave and then walk to left edge of screen
```

I doubt you'd believe me in a million years, but this is seriously how I thought you'd program a computer. It's not exact, since I can't remember over 10 years ago, although, it does carry certain principles how I thought; such as: create a man, wave and move him left etc.. If only it was... I'd have made a thousand games by now!!

CREDITS:

Anyway, credits for this book must go to the anonymous Joe XXXX in London who's help has proved too valuable to mention, the TWAUG team who have kept my ego alive (and the producers of this book), Ann O. who did try to think up some tips that I hadn't compiled, the rest of my penfriends who didn't lend a hand whatsoever, Derek Fern because I liked his quick service, Phil A. coz I liked his attitude and last but not least, I would like to thank my cup of coffee who was always there for me (thanks mum).

There aren't any anti-credits except those to my printer. I had this introduction and 3 other sheets of paper to print out to complete this book and the printer decides it wants my cup of coffee. In getting it fixed my platten decides that it doesn't want to feed the paper through correctly so I decide to hand feed it myself (get it!?), anyway, as I start succeeding in this the printer ribbon decided to go fady. And if that's not enough the power switch shorted out! A sad case of Murphy's law don't you think?

Rightyo, if you want to get in contact with me about anything in this book then write to me at this address only:

"Concerning the book"

MR. AC. BOOK
135 HENLLYS WAY
CWMBRAN
GWENT NP44 7NF
SOUTH WALES

COMPLETE & ESSENTIAL MAP

00 - 06

This is where the map takes the 1st step and where-else than at the lowest number upwards.

Locations 0 - 255 are Page 0 and is probably the most important page of the entire memory except for the ROM because it is especially fast to access for machine-code programmers. Time is a critical factor for us Atari programmers and should be a point of note for programmers wishing to learn machine-code.

Locations 0 - 127 are reserved for the Operating system (OS), they can be used as RAM, but a strict control of non-interrupt and direct processing must be achieved. This is also only possible in machine-code. Locations 128 - 255 are used by Basic when installed including the Floating point (FP) package and user RAM. They can be used as RAM if the FP package isn't used and if Basic is not used.

00 00 LNFLG

Used by the Atari in-house debugging programs when they were clearing the OS from bed bugs, also used during power-up.

01 01 NGFLAG

Used for memory testing during power-up, if zero then a memory failure is present.

02,3 02,3 CASINI

Cassette initialization vector. If Cassette boot is successful, OS JSR's through here. This vector comes from bytes 5 and 6 of the cassette boot record. See the BOOT appendix for further information on the importance of the 1st 6 bytes of a boot file.

04,5 04,5 RAMLO

RAM pointer for the memory test during power-up. Also used to store the disk boot address - normally 1798 ($706) for the boot continuation routine.

06 06 TRAMSZ

Temporarily used for RAM size during power-up. This is the value moved to RAMTOP, Location 106 ($6A). It reads 1 when Basic is on.

07 - 13

07 07 CMCMD

Command flag for 835 and 1030 modems. Set to nonzero to pass commands to the modem.

08 08 WARMST

Warmstart reads 0 during power-up. Set to 255 on pressing Reset. Warmstart normally vectors to 58484 ($E474). WARMST is checked by the NMI status register at 54287 ($D40F) when Reset is pressed to see whether to re-initialise the softare in memory or to re-boot the disk, cassette or Basic.

09 09 BOOT

Boot flag success indicator. If set to 1 then disk-boot was successful, 2 means cassette-boot and 3 means both were successful. It reads 0 if no peripheral was booted.
If user-set to 255 then pressing Reset will lock-up the system. By setting this location to 2, location 2 to 52 and location 3 to 185 then the Reset key can be TRAPped in Basic to run at any particular line-number. Machine-code user's simply use locations 2 and 3 as a vector for Reset when having set this location to 2.

10,11 0A,B DOSVEC

Start vector for disk or non-cartridge software. Also the address Basic jumps to when DOS is called. This address can be user-set, but Reset will return DOSVEC to it's original address unless the values placed here are also loaded into locations 5446 and 5450 ($1546 and $154A). Locations 10 and 11 are normally set to 159 and 23. Without DOS loaded, typing DOS will pass control to the inbuilt Selftest at 58481 ($E471).

12,13 0C,D DOSINI

Initialization address for the disk boot, bytes 5 and 6 of the 1st sector. Also used to store the cassette boot RUN address which is then moved to CASINI (2 and 3). Set to 0 if no peripheral was booted.
You can also use these locations as a vector on pressing Reset alike Locations 9,2 and 3.

14,15 OE,F APPMHI

Applications memory high limit and pointer to the end of your Basic program, used by OS and Basic. This contains the lowest address you can use to set up a screen and display list (DL) (which is also the highest address usable for programs and data below which the display memory (DM) may not be placed). The screen handler will not open the "S:" device if it would extend the screen RAM or the DL below this address; memory above this address may be used for the screen display and other data (PMG's etc.).
If an attempted screen mode change would extend the screen memory below APPMHI, then the screen is set up for Graphics 0; MEMTOP (741, 742) is updated and an error is returned to the user. Otherwise the memory is not too small for the screen editor, the mode change will take affect and MEMTOP will be updated. This is one of 5 locations used by the OS to keep track of the user and DM.
If you use the area below the DL for your character sets, PMG's etc. then be sure to set APPMHI above the last address used so that the DL data will not descend and destroy your own data. See RAMTOP at 106, MEMTOP at 741 and 742, PMBASE at 54279 and CHBASE at 54281.

16 10 POKMSK

Pokey interrupts: the IRQ service uses and alters this location. Shadow for 53774 ($D20E). Poke with 112 ($70; also poke 53774 with the same value) to disable the Break key. The bits in this register have the following purpose (1 meaning enable and 0 meaning disable):

BIT:	DEC:	ACTION:
7	128	Break key enable.
6	64	'Other-key' enable,
5	32	Serial input data ready enable,
4	16	Serial output data required enable,
3	8	Serial out transmission finish enable,
2	4	Pokey timer 4 enable,
1	2	Pokey timer 2 enable,
0	1	Pokey timer 1 enable.

Timer interrupt enable means that the associated AUDF registers are used as timers and will generate an interrupt request when they have counted down to 0. See locations 528 - 535 ($210 - $217) and the Pokey chip locations 53760 ($D200) onward for a further explanation. Default value is 192 ($C0).
Break is re-enabled on Reset, the first Print statement to the screen, any Open statement that addresses "S:" or "E:" and any Graphics call. The Break interrupt bit should therefore be checked regularly in order to retain it disabled. Also see locations 566 and 567 ($236 and $237) about writing a new vector routine for the Break key.

17 11 BRKKEY

Nonzero means the Break key is pressed, 0 otherwise. Break during I/O returns the value 128. The Break key abort status code is stored in STATUS at 48 ($30). It's also checked during all I/O and scroll/draw routines. During the keyboard handler routine the status code is stored in DSTAT at 76 ($4C). BRKKEY is turned off at power-up and the abort status is flagged by setting bit-7 of 53774 ($D20E). See location 16 ($10), above.

18,19,20 12,13,14 RTCLOK

Internal realtime clock. Location 20 increments every stage one VBI (1/50th second = 1 jiffy) until it reaches 255; then location 19 is incremented and 20 is reset to 0 (every 5.12 seconds). When location 19 reaches 255, it and 20 are reset to 0 and location 18 is incremented (every 21.84 minutes or 65536 TV frames). You can use these locations as a timer, thus:

```
TIME = INT ((PEEK(18)*65536+PEEK(19)*256+PEEK(20))/50)
```

To see the count in jiffies, eliminate the "/50". To see the count in minutes, change "/50" to "/300". The maximum value of RTCLOK is 16,777,215. When it reaches this value it will be reset to 0 at the next VBI. This value is the result of cubing 256 - 1 (i.e. 256 * 256 * 256 -1), the maximum number of increments in each clock register. The RTCLOK is always updated every VBI regardless of the time-critical nature of the code being processed.

In the Atari' terms, a jiffy is 'almost forever'. It can perform up to a maximum of 35568 machine cycles in this time. That's an approximate average of over 9000 machine code instructions!

You can poke these timers with your own suitable values, use them as a delay timer or whatever, ie:

```
10 POKE 20,1
20 IF PEEK(20) THEN 20
```

This example will wait at line 20 for 5.1 seconds.

21,22 15,16 BUFADR

Indirect buffer address register. This is used as a temporary pointer to the current disk buffer.

23 17 ICCOMT

Command for CIO vector. Stores the CIO command which is used to find the offset in the command table for the correct vector to the handler routine.

24,25 18,19 DSKFMS

Disk file manager pointer. Called JMPTBL by DOS; used as vector to FMS.

26,27 1A,1B DSKUTL

This is the disk utilities pointer, called BUFADR by DOS. It points to the reserved buffer from the DUP package at DBUF, or for the program area pointer at MEMLO.

28-31 1C-1F ABUFPT

These bytes were intended as buffer pointers, though are actually unused.

Locations 32 - 47 are the Page 0 I/O control block (ZIOCB). ZIOCB is used to communicate data between the CIO and the device handlers. When a CIO operation is executed, the IOCB channel information is loaded down to here to be used by CIO. Upon completion of the operation, ZIOCB is then transferred back to the correct IOCB.

32 20 ICHIDZ

Handler index number. Set by the OS as an index to the device name table for the currently open file. If there is no file open on this IOCB then the IOCB is free and ICHIDZ will equal 255.

33 21 ICDNOZ

The current device number.

34 22 ICCOMZ

Command code byte. See ICCOM, byte 2 of IOCB for further breakdown.

35 - 46

35	23	ICSTAZ

Status of the last IOCB action, set by the OS.

36,37	24,25	ICBALZ/HZ

Buffer address for data transfer, also used for filename address pointer for Open, Status etc. commands.

38,39	26,27	ICPTLZ/HZ

Put byte routine address set by the OS. It's the address-1 of the 'put one byte' routine. On the Close statement, it points to "IOCB not OPEN".

40,41	28,29	ICBLLZ/HZ

Buffer length byte count used for Put and Get operations. This length is decremented every call. When it reaches 0 then the operation is complete, though, if in the event of a Get operation, EOF is found but this value is still greater than 0 then an error is returned and the file IOCB status remains Open.

42	2A	ICAX1Z

Auxiliary operation 1st byte. In an Open operation the #1 is the IOCB channel used, the next number which is the file access is ICAX1Z.

43	2B	ICAX2Z

Auxiliary 2nd byte. Also used by some serial port functions.

44,45	2C,2D	ICAX3Z/4Z

These auxiliary bytes are used by Basic Note and Point commands for the transfer of disk sector numbers. These last 4 ZIOCB bytes are also labelled ICSPRZ and are spare bytes for local CIO use.

46	2E	ICAX5Z

This refers to the byte being accessed in the sector (noted in ICAX3Z/4Z). Also used for the IOCB index number multiplied by 16.

47 2F ICAX6Z

Spare. Also labelled CIOCHR, it is the temporary storage for the character byte in the current Put operation.
The reason why so many auxiliary bytes exist is for the possible need that future hardware add-ons may need.

Location 48 - 75 are user and OS variables for the Atari I/O routines.

48 30 STATUS

Internal status storage. SIO uses this byte as the status of the current SIO operation. See the ERRORS appendix for status values. STATUS uses location 793 ($319) as temporary storage. STATUS is also used as a storage timeout, Break abort and error values during SIO operations.

49 31 CHKSUM

Data-frame checksum used by SIO: single byte sum with carry to the LSB. Checksum is the value of the number of bytes transmitted. When the number of transmitted bytes equals the checksum, a checksum sent flag is set at 59 ($3B). Uses 53773 and 56 ($D20D and $38) for comparison of values.

50,51 32,33 BUFRLO/HI

Pointer to the data buffer. Used by SIO and the device control block (DCB), and points to the 1st byte of the data to send or area to receive. Bytes are transferred to the 8-bit serial output holding register or from the input holding register at 53773 ($D20D). Location 53773 is used to hold the 8-bits which will be transmitted 1 at a time to or from the device. Note that the bits are only transmitted when the register is full, when empty it is updated with another byte.

52,53 34,35 BFENLO/HI

This is the next byte past the end of the SIO/DCB data buffer described in BUFRLO/HI.

54,55 36 ,37 LTEMP

Temporary buffers for the general purpose peripheral handler loader routines (PHLR). The PHLR helps the OS deal with new handlers and peripherals which load their own handlers. All locations marked as being used by the peripheral handler or loader are for OS use only and should be left alone. As stated earlier, they can be used as RAM, but a very strict programming environment MUST be achieved.

56 38 BUFRFL

Data buffer full flag. 255 equals full.

57 39 RECVDN

Data received done-flag. 255 equals done.

58 3A XMTDON

Transmission done flag. 255 is done.

59 3B CHKSNT

Checksum sent flag. 255 equals sent, 0 if not.

60 3C NOCKSM

Flag for "no checksum follows data". Nonzero means no checksum follows, 0 means checksum follows transmission data.

61 3D BPTR

Cassette buffer pointer: record data index into the portion of data being read or written. Ranges between 0 and the current value at 650 ($28A). When these values are equal, the buffer at 1021 ($3FD) is empty/full depending on the I/O operation. Initialized to 128.

62 3E FTYPE

Inter record gap type (IRG). Copied up from ZIOCB location 43. Normal IRG's have a nonzero number, while the rarer used continuous gaps show up as 0.

63 3F FEOF

Cassette end-of-file flag (EOF). 0 means the EOF has not been reached, nonzero means it has. An EOF record has been reached when the command byte of a data record equals 254. See 1021.

64 40 FREQ

Beep count retain register. Counts the amount of beeps required by the cassette handler during the Open command for Play or Record operations; 1 beep for play and 2 for record.

65 41 SOUNDR

Noisy I/O flag used by SIO to signal the beeping heard during cassette and disk operations. 0 makes the beep Quieter, while nonzero blurts it through the TV speaker. To completely silence the noise then the sound register updates must be removed from the ROM at $EC58 - $EC83.

66 42 CRITIC

Critical flag. When CRITIC is nonzero then the deferred VBI is disabled. This means that all shadow registers are not updated at the stage-2 VBlank. See the VBLANK appendix for a description of the stage-2 VBlank. When 0, then both standard VBI's are enabled, which is also the default value.

67-73 43-49 FMZSPG

Disk file management system (FMS) variables. Re-initialized by FMS each time it takes control.

74,75 4A,4B ZCHAIN

Temporary storage registers for the general purpose peripheral handler loader.

76 4C DSTAT

Display status and keyboard register used by the display handler. Also used for: screen memory too small, cursor out of range and Break abort status.

77 4D ATRACT

Attract mode timer and flag. The Attract mode rotates the display colours at low levels when there has been no keyboard input for approx. 10 minutes. This helps save the TV screen from 'burn-out' damage caused from leaving the computer unused for long periods of time. The keyboard IRQ resets ATRACT to 0 when a key is pressed, otherwise incremented every 5 seconds by VBlank (see 18,19 and 20). When ATRACT reaches 127 it is changed to 254 which is the flag indicating to rotate colours whilst it is sitting idle. You can poke a value greater than 126 to see the affect immediately.
DLI colour changes will not be attracted. To reset it in a program then poke here with 0. If the attract mode is not wanted in programs then it should be cleared regularly.

78 4E DRKMSK

Dark attract mask; Initialized to 254 for normal brightness of colours when attract mode is not activated. Set to 246 when ATRACT is enabled to ensure screen luminances do not exceed 50%.

79 4F COLRSH

Colour shift mask. The colour registers are EOR'd with DRKMSK and COLRSH at the stage-2 VBlank. When set to 0 and DRKMSK to 246, colour luminance is reduced 50%. COLRSH currently equals location 19, thus changes colour every 5.12 seconds.

Locations 80 - 122 are used by the screen editor and display handler.

80 50 TEMP

Temporarily used by the display handler for moving data to and from screen. Also called TMPCHR.

81 51 HOLD1

Alike TEMP, this holds the number of display list entries.

82 52 LMARGN

Left margin column. Initialized to 2 and has a range 0 to 39. It's useful to set this to 0 when typing in large Basic listings in order to have 6 extra spaces per logic line (1 logic line is 3 physical display lines).

83 53 RMARGN

Right margin column. Initialized to 39. When altering margins, it should be known that the screen edit commands like shift+delete don't change from 40 byte line operations to the new columns total according to the setting of the margins. This also applies to narrow and wide screens (see location 559). Although the screen widths alter, the screen handler still operates as if there are 40 columns per physical line.

84 54 ROWCRS

Current graphics/text screen row ranges between 0 - 191 depending on the Graphic mode in use. ROWCRS and COLCRS define the next element to be read/written to the screen.
To draw lines in machine-code you need to use the Draw command on an IOCB channel in conjunction with ROWCRS, COLCRS, OLDROW, OLDCOL also FILFLG and ATACHR. See page 85 in the map for further information.

85,86 55,56 COLCRS

Current graphics/text mode cursor column ranges between 0 - 319 depending on Graphics mode in use. For the text window, values in locations 656 - 667 are exchanged with the current values in locations 84 - 95 and location 123 is set to 255 to indicate swap has taken place.
Basics Locate command not only examines the screen when used, but also moves the cursor forward one position by updating these locations. To avoid this you need to take note of ROWCRS and COLCRS before the Locate command and replace the values afterwards.

87 57 DINDEX

Display mode index to screen mode. DINDEX contains the low 4-bits of the most recent Open AUX1 byte. It can be set to any graphics mode. You can fool the OS into thinking it's in a different Graphics mode by Pokeing the mode you want into DINDEX. Try calling Graphics 8 and Pokeing 7 here, you'll have a split screen of mode 7 on top and mode 8 below. You need to change location 89 to point to the area of the screen you wish to draw in. You may get some unexpected 'cursor out of range' errors changing modes in this manner also so be careful.
You can get a text window in the GTIA modes with this program:

```
10 GRAPHICS 9
20 POKE 87,0:POKE 623,64:POKE 703,4
30 GOTO 30
```

Location 623 can be Poked with 64, 128 or 192 for GTIA modes 9, 10 or 11. You won't be able to read the text in the window but will be able to write to it. It is possible to create a true text window but you have to use a DLI. See the DLI appendix.

88,89 58,59 SAVMSC

The lowest address of screen memory corresponding to the upper left corner of the graphics/text screen. The upper left corner of the text window is at locations 660 and 661. You can check this with:

```
10 GRAPHICS 1
20 SCREEN=PEEK(88)+256*PEEK(89)
30 WINDOW=PEEK(660)+256*PEEK(661)
40 POKE SCREEN,51:POKE WINDOW,55
```

How is each mode configured? Well, take a look at the chart below:

GRAPHIC MODE	ROWS full	/ split	COLUMNS /line	BYTES /line	SCREEN MEMORY	DL MEMORY
0	24	20	40	40	960/960	32/na
1	24	20	20	20	480/640	32/34
2	12	10	20	20	240/400	20/24
3	24	20	40	10	240/400	32/34
4	48	40	80	10	480/640	56/54
5	48	40	80	20	960/1120	56/54
6	96	80	160	20	1920/2080	104/94
7	96	80	160	40	3840/4096	104/94
8	192	160	320	40	7680/7936	202/176

88,89 cont.

9	192	160	80	40	7680/7936	202
10	192	160	80	40	7680/7936	202
11	192	160	80	40	7680/7936	202
12	24	20	40	40	960/1120	32/34
13	12	10	40	40	480/640	20/24
14	192	160	160	20	3840/4096	200/174
15	192	160	160	40	7680/7936	202/176

Note, that the 1st number in the Screen memory is the amount of memory actually needed, where the 2nd number defines the amount set aside due to handler calculations and boundaries. The 1st DL number is the amount of full-screen instructions, the 2nd being the split-screen amount. When the screen clear function is executed the display handler clears the memory between the address given by SAVMSC and RAMTOP. The old-bug of RAM being cleared above RAMTOP with the Screen-CLEAR function and the scrolling of the text-window is now been eradicated, so feel free to protect RAM directly above RAMTOP without any worries of it being lost. SAVMSC and RAMTOP can also be used in your own programs to clear bulks of memory fast. This is especially useful in clearing PMG's or strings, ie:

```
10 POKE 88,0:POKE 89,40
20 POKE 106,PEEK(106)
30 ? CHR$(125):GRAPHICS 0
```

This clears all the memory from location 10240 (40 * 256) to RAMTOP - 1. Be sure to call a graphics mode afterwards so that the screen write address is returned to normal.

Here's a useful routine that can be included in your own programs. It will load a picture file into the Graphics mode in use:

```
10 GRAPHICS 15+16:MEM=7680
20 DATA 104,104,104,170,76,86,228
30 FOR I=0 TO 6
40 READ D:POKE 1536+I,D:NEXT I
50 HI=INT(MEM/256):LO=MEM-HI*256
60 OPEN #1,4,0,"D:FILENAME.PIC"
70 POKE 849,1:POKE 850,7:POKE 852,PEEK(88):POKE
853,PEEK(89)
80 POKE 856,LO:POKE 857,HI:POKE 858,4
90 X=USR(1536)
95 CLOSE #1
```

If you wish to save the picture to disk, then you need to alter it to:

```
60 OPEN #1,8,0,"D:FILENAME.PIC"
```

Line 70 should POKE 850,11 and line 80 should POKE 858,8

90 - 97

The program loads/writes MEM amount of bytes, thus, if you change the Graphics mode then you should also alter the MEM variable according to the memory chart on the previous page.
Note that the colour registers are not saved to the file, so these should be saved by the user. It's recommended to save them at the end of the file to keep it compatible with most graphic packages including XL ART and MICROPAINTER, because you can then use these packages to load your pictures.

90 5A OLDROW

Previous graphics cursor row updated from ROWCRS before every operation. Used to determine the starting row for DRAWTO or FILL. See the IOCB DRAW appendix.

91,92 5B,5C OLDCOL

Previous graphics cursor column updated from COLCRS before every operation. See page 97 of the map.

93 5D OLDCHR

Retains the character under the cursor. Used to restore that character after the cursor has moved.

94,95 5E,5F OLDADR

Retains the memory location where the cursor currently is. Also used with OLDCHR in the replacing of the character under the cursor.

96,97 60,61 FKDEF

The 1200XL has 4 redefinable function keys. FKDEF points to 64529 ($FC11) which is their definition table. An 8-byte table for keys F1 - F4. Each value is in internal codes and not Ascii which are values 138 - 141, but you must not assign a key it's own value since it will generate an endless loop.

KEY combination:
F1 Cursor up, Atascii 28,
F2 Cursor down, code 29,
F3 Cursor left, code 30,
F4 Cursor right, code 31.

With SHIFT:
F1 Home (Cursor to top-left),
F2 Cursor to lower left corner.
F3 Cursor to start of physical line,
F4 Cursor to right of physical line.

With CONTROL:
F1 Keyboard enable/disable toggle,
F2 Screen display enable/disable,
F3 Key click on/off,
F4 Domestic/International character-set toggle.

This also appears in the 800XL, but there are no function keys! The HOME function also exists in all XL's and XE's but is not on the keyboard, see 764, 121 and 122.

98 62 PALNTS

Flag to determine PAL or NTSC and (I think) SECAM also. 0 means North American Standard, 1 means PAL, but SECAM otherwise.

99 63 LOGCOL

Position of the cursor within a logical line. A logical line is 3 physical lines whatever their width between 1 and 40 columns. The maximum range of LOGCOL is 0 - 119.

100,101 64,65 ADRESS

Temporary storage used by the display handler for the display list address; line buffer, new MEMTOP value after DL entry, row column address, DMASK value, data to the right of the cursor, scroll, delete, clear screen routine and for the screen address memory.

102,103 66,67 MLTTMP

Also called OPNTMP and TOADR; first byte used in Open as temporary storage, also used by the display handler.

104,105 68,69 SAVADR

Also called FRMADR, used temporarily with ADRESS for the data under the cursor and in moving line data on the screen.

106 - 108

106 6A RAMTOP

Pointer to the top of RAM (RAM size). Defined by the power-up sequence and passed here from TRAMSZ. The value here is the amount of PAGES free in your machine, where 1 page is 256 bytes.
In the 48K Atari, this is initialized to 160 with Basic, 192 without. Note that MEMTOP should not extend below this value otherwise the DL and display memory can destroy program or data memory.
You can fool the OS into thinking it has less RAM than it really does by lowering this value. This technique is useful to protect data loaded into memory from being overwritten by program memory. This is widely used for placing character sets behind or PMG's, see SAVMSC at 88 and 89.
If you wish to protect data behind RAMTOP, then you need to POKE 106,PEEK(106)-X where X is the amount of pages to be protected, ie:

```
10 POKE 106,PEEK(106)-1
20 GRAPHICS 0
30 PADR=PEEK(106)*256
```

Where after protecting 256 bytes of memory, PADR equals the 1st address of the reserved area. Character-sets require 4 pages, PMG's take 4 Pages for double line resolution and 8 for single line res. See PMBASE and the BOUNDARY appendix also.
If you do use RAMTOP to protect memory, you should call the graphics mode in use immediately afterwads so the OS can re-calculate the DL and Display Memory into it's new area.
One caution: apparently, Basic cannot always handle setting up a DL and DM for Graphics 7 and 8 when you modify this location by less than 4K (16 pages). Some bizarre results occur if you use PEEK(106)-8 in these modes, for example. Use a minimum of 4K to avoid trouble. This could explain why some people have had trouble with PMG's in these modes.
An alternative to reserving/protecting memory in high RAM is to making an area below MEMLO at 743. See also MEMTOP at 741.

107 6B BUFCNT

Buffer count; the screen editor current logical line size counter.

108 6C ...

According to Mapping, this location and location 109 is BUFSTR which is the display editor GETCH routine pointer and temporary storage for the character pointed to by BUFCNT, however, I find that in my 800XL, this is the pointer to the current cursor row.

109 - 120

109 6D ...

See above. Initialized to 2, user alterable but restored on Reset.

110 6E BITMSK

Bit mask used in bit mapping routines by the OS display handler. Also a display handler temporary storage register.

111 6F SHFAMT

Pixel justification: the amount to shift the right justified pixel data on output or the amount to shift the input data to right justify it. Prior to justification, the value is always the same as that in location 672.

112,113 70,71 ROWAC

ROWAC and COLAC are both working accumulators for the control of row and column point plotting and the increment and decrement functions.

114,115 72,73 COLAC

Controls column point plotting.

116,117 74,75 ENDPT

End point of the line to be drawn. Contains the larger value of either DELTAR or DELTAC to be used along with ROWAC and COLAC to control the plotting of line points.

118 76 DELTAR

This is the change of vertical position when drawing a sloped line.

119,120 77,78 DELTAC

Delta column; contains the absolute value of NEWCOL minus the value in COLCRS. These delta register values along with ROWINC and COLINC are used to define the slope of the line to be drawn.

121,122 79,7A KEYDEF

Pointer to the keyboard definition table, initialized to 64337 ($FB51), where the system keyboard table resides. You can redefine the keyboard by writing a 192-byte table and POKEing its address here; the table consists of 3 64-byte portions: lowercase keys, SHIFTed keys and CTRLed keys, assigned in the manner below:

00	l	16	v	32	,	48	9
01	j	17	HELP	33	SPACE	49	(128)
02	;	18	c	34	.	50	0
03	F1	19	F3	35	n	51	7
04	F2	20	F4	36	(128)	52	B/SPACE
05	k	21	b	37	m	53	8
06	+	22	x	38	/	54	<
07	*	23	z	39	INVERSE	55	>
08	o	24	4	40	r	56	f
09	(128)	25	(128)	41	(128)	57	h
10	p	26	3	42	e	58	d
11	u	27	6	43	y	59	(128)
12	RETURN	28	ESC	44	TAB	60	CAPS
13	i	29	5	45	t	61	g
14	-	30	2	46	w	62	s
15	=	31	1	47	q	63	a

The next 64 characters are SHIFTed, ie. a becomes A, 5 becomes % etc. Followed after that are the CTRLed characters: many graphics characters.
Several values have specific meaning to the keyboard decoder, thus:

ATASCII:	USE:
128	Unused; invalid value
129	Inverse output
130	Upper/lower case toggle
131	Caps lock
132	CTRL key lock
133	End of file (EOF)
137	Keyboard click toggle
138 - 141	1200XL function keys F1-F4
142	Cursor HOME
143	Cursor to bottom left
144	Cursor to left margin
145	Cursor to right margin

You can create your own table, or better still, just include those normally unobtainable keyboard functions to the standard table. Type in the program on the next page and try pressing CTRL and a number key between 4 and 8.

```
10 KEYDEF=PEEK(121)+256*PEEK(122)
20 FOR I=0 TO 191
30 POKE 1536+I,PEEK(KEYDEF+I)
40 NEXT I
50 POKE 121,0:POKE 122,6
60 POKE 1536+128+24,142
70 POKE 1536+128+29,143
80 POKE 1536+128+27,144
90 POKE 1536+128+51,145
92 POKE 1536+128+53,137
```

You will now find that you have the following keyboard functions:

CTRL+4	Cursor HOME
CTRL+5	Cursor to bottom left
CTRL+6	Cursor to left margin
CTRL+7	Cursor to right margin
CTRL+8	Keyboard click toggle

The new keyboard table occupies page 6 of memory (locations 1536 - 1791), but you can turn the ROM into RAM and alter the original table. See the RAM-OS appendix.

123 7B SWPFLG

Split-screen cursor control. Equal to 255 in the text window RAM and regular screen RAM are swapped; otherwise equal to 0. In split screen modes, the graphics cursor data and the text window data are frequently swapped in order to get the values associated with the area being accessed into the OS data-base at locations 84 - 95. SWPFLG helps to keep a track of which data set is in these locations.

124 79 HOLDCH

The keyboard character value is moved here before the CTRL and SHIFT logic are processed for it.

125 7A INSDAT

Temporarily used by the display handler for the character under the cursor and the end of line (EOL) detection.

126,127 7B,7C COUNTR

Counter for the amount of iterations/steps to draw a line. As each point of the line is drawn, this value is decremented. 0 means the line is complete.

128,129   7D,7E   LOMEM

Pointer the Basics low memory which is at the end of the RAM. The 1st 256 bytes pointed to are the TOKEN output buffer, which is used by Basic to convert Basic statements into numeric representation. See STMTAB and the TOKENIZATION appendix.
This value is loaded down from MEMLO on initialization or the execution of a NEW command. Remember to update LOMEM when changing MEMLO in reserving memory space.
When a Basic SAVE is initiated, two blocks of information are written to the output device: the 1st block is the 7 pointers from LOMEM to STARP at 140,141. The value of LOMEM is subtracted from each of these 2-byte pointers in the process, thus, the 1st two bytes written will be 0's (LOMEM - LOMEM). The 2nd block contains: the variable name table, the variable value table, the Basic program in its TOKENized form and lastly the immediate mode line number, which is 32768 (1 number higher than the highest accessible line number). When the Basic LOAD is initiated, Basic adds the value at MEMLO to each of the 2-byte pointers as in the reverse of the SAVE operation. The pointers are placed back in page-0 and the values in RUNSTK at 142,143 and MEMTOP at 144,145 are set to the value in STARP. Next, 256 bytes are reserved above the value in MEMLO for the token output buffer, and the program is read in to the memory following this buffer.
Without DOS loaded, LOMEM points to 1792, but points to 7676 with DOS. Changing the drive and data-buffers will raise/lower this value by 128 bytes per buffer accordingly. The RS232 takes a further 1728 bytes.
LOMEM is called ARGOPS by Basic when used in expression evaluation. When Basic encounters any kind of expression, it puts the immediate results into a stack. ARGOPS points to the same 256 byte area; for this operation it is reserved for both the argument and operator stack. It's also called OUTBUFF for another operation pointing to the same 256 byte area as ARGOPS. Used by Basic when checking a line for syntax and TOKEN conversion. Also temporary token store.

130,131   82,83   VNTP

Beginning address of the variable name table. Variable names are stored in the order they are entered into your Basic program, in Atascii format. You can have up to 128 variable names and these are stored as tokens representing the variable number within the tokenized Basic program, numbered 128 - 255.
The table continues to store all variables: from immediate mode, program mode, even deleted ones remain in memory.

It is not cleared upon SAVE, but is replaced with the VNT obtained from a LOADed file. The only way to renew the table is by first LISTing your program to the output device as this stores the file in a different manner and does not save the VNT. Then you can ENTER the file, to SAVE it with it's new VNT. Before ENTERing the file back in, be sure to use a NEW statement to erase the old program and VNT, or better still, give the Atari a coldstart.
With numeric (scalar) variables, bit-7 (the MSB) is set on the last character in the name. String variables have a "$" for the last character with the MSB set. Array variables have a "(" for the last character also with the MSB set. With the MSB being set, it just inverses the character mentioned in each case.
Here's a short routine to display all the variables of a resident program:

```
10 POKE 203,PEEK(130):POKE 204,PEEK(131)
11 IF PEEK(203)=PEEK(132) AND PEEK(204)=PEEK(133) THEN STOP
12 ? CHR$(PEEK(PEEK(203)+256*PEEK(204)));
13 IF (PEEK(PEEK(203)+256 * PEEK(204)))-127 THEN ?
14 IF PEEK(203)=255 THEN POK.203,0:POKE
204,PEEK(204)+1:GO.11
15 POKE 203,PEEK(203)+1:GOTO 11
```

You can also directly change the variable names by POKEing the Atascii values accordingly. If you renamed the variable in the Basic program, the old name would still exist which is occupying 1 of the 128 variables allowed.

132,133 84,85 VNTD

Pointer to the ending address of the variable name table + 1. When less than 128 variables are present, then it points to a 0 value.

134,135 86,87 VVTP

Address of the variable value table. 8-bytes are allocated for each variable in the name table as follows:

BYTE		1	2	3 4	5	6	7	8
VARIABLE								
Scalar		00	var#	six byte BCD constant				
Array;	DIMed	65	var#	offset		first		second
	unDIMed	64		from STARP		DIM+1		DIM+1
String;	DIMed	129	var#	offset		length		DIM
	unDIMed	128		from STARP				

135 - 137

In scalar (unDIMensioned numeric), bytes 3-8 are the FP number; byte-3 is the exponent, byte-4 contains the least significant 2 decimal digits and byte-8 contains the most significant 2 decimal digits.
In array variables, bytes 5 and 6 contain the size+1 of the 1st dimension of the array (DIM+1; LSB/MSB) while bytes 7 and 8 contain the size+1 of the 2nd dimension (the 2nd DIM+1; LSB/MSB).
String variables bytes 5 and 6 contain the current length of the variable (LSB/MSB) while bytes 7 and 8 contain the actual dimension (up to 32767).
In all cases, the first byte is always one of the numbers listed on the chart above (you will rarely see the undimensioned values in a program). This number defines what type of variable information will follow. The next byte, var# (variable number), is in the range 0 - 127. Offset is the number of bytes from the beginning of STARP at 140,141. Since each variable is 8-bytes, you can find the values for each variable by:

```
10 VVTP=PEEK(134)+256*PEEK(135)
11 ? "ENTER VARIABLE NUMBER ";
12 INPUT VAR
13 FOR L=0 TO 7
14 ? PEEK(VVTP+8*VAR+L);
15 NEXT L
```

A very handy and widely used technique to clearing a string or assigning a particular character throughout each element within a string can be achieved with this program:

```
10 DIM TEST$(100)
20 TEST$="*":TEST$(100)=TEST$:TEST$(2)=TEST$
30 ? TEST$
```

136,137 88,89 STMTAB

Address of the statement table which is also the beginning of your Basic program, containing all the TOKENized lines of code including the immediate mode lines entered by the user. Line numbers are stored as 2-byte integers, while immediate mode lines are given the default value of 32768. The structure of a TOKEN line is as follows:

BYTE:
1-2 Program line number
3 Dummy, reserved for byte count/offest from the start of this line to the start of the next.
4 2nd counter for the start of this line to the start of the next statement. These count values are set only when tokenization for the line and statement are complete.

138 - 141

To see the starting address of your Basic line numbers, use this routine:

```
10 STMTAB=PEEK(136)+256*PEEK(137)
20 NUM=PEEK(STMTAB)+256*PEEK(STMTAB+1)
30 IF NUM=32768 THEN STOP
40 ? "LINE NUMBER - ";NUM;", ADDRESS ";STMTAB
50 STMTAB=STMTAB+PEEK(STMTAB+2)
60 GOTO 20
```

138,139 8A,8A STMCUR

Current Basic statement pointer, used to access the tokens currently being processed within a line of the statement table. While Basic is awaiting input, this pointer is set to 32768. Using the address of the variable name table, the length and the current statement you can protect your Basic programs from being listed or even loaded. They can only be RUN! Be sure to save an unchanged version of your program because this process is irreversable once done:

```
32763 FOR V=PEEK(130)+256*PEEK(131) TO
PEEK(131)+256*PEEK(132)
32764 POKE V,155:NEXT V
32765 POKE PEEK(138)+256*PEEK(139)+2,0
32766 SAVE"D:FILENAME.EXT"
32767 NEW
```

Include this on your program to protect. Note, in future, you must RUN it directly from disk.

140,141 8C,8D STARP

The string and array table address and a pointer to the end of your Basic program. The address of the strings in the table are the same as those returned by the Basic ADR function. Always use this function under program control, since the addresses in the table change along with your program size. Each dimension of an array requires 6 bytes, thus, DIM A(100) takes up 100*6 = 600 bytes, because each element of the array can store a number of up to 6 figures in length.
A string of the same format, DIM A$(100) only requires 100 bytes because each element is just the 1 byte. It would save considerable memory to use strings as opposed to arrays, ie:

```
10 DIM A(2)
20 A(0)=36:A(1)=9:A(2)=8
30 ? A(1)+A(2),A(0)
```

```
10 DIM A$(4)
20 A$="3698"
30 ? VAL(A$(3,3))+VAL(A$(4,4)),VAL(A$(1,2))
```

The 1st program takes 6*10 = 60 bytes for array memory, but the 2nd program just takes 10 bytes for string memory.

142,143 8E,8F RUNSTK

Address of the runtime stack which holds the GOSUB entries (4-bytes) and the FOR/NEXT loops (16-bytes).
The structure of the GOSUB is: byte-1 = 0, bytes 2 and 3 = line number on which the actual GOSUB call exists and byte 4 = an offset so that the Basic RETURN statement can return to the correct position in the GOSUB line.
FOR/NEXT is structured as: bytes 1 to 6 = counter variable limit, bytes 7 to 12 = the step increment, byte 13 = counter variable number with the MSB set, byte 14 and 15 = is the FOR part line number and byte-16 = is the offset for the line where the FOR is so that the next statement on the same line (if one exists) can be executed.

RUNSTK is also called ENDSTAR by Basic to point to the end of the string/array space pointed to by STARP.

144,145 90,91 MEMTOP

Pointer to the top of Basic memory, the end of the space the program takes up. There may still be space between this address and the display list which is also the value returned by the Basic FRE command. This is also called TOPSTK; it points to the top of the stack space pointed to by RUNSTK.

146 92 MEOLFLG

Basics modified EOL flag register. The Atari BASIC source-book (pages 144 - 147) lists all the RAM locations used by Basic, if I had the book I would have listed them here, but unfortunately it's one of the few I don't have.

147 93 ...

Unused (apparently).

148,149 94,95 ...

This is one from my own book, I don't think it's purpose is meant to be, but it's the address of the screen editor entry point. Weird.

149,150 95,96 POKADR

According to mapping (which I'm sure is right), this is the address of the last POKE location. If no POKE command has been given then it is the address of the last operator token (often 155 for EOL).
I find that when I tried to find out what this was, it's address points directly to the 2nd of the 2 Basic statement tables. The 1st is at 42145. Turbo Basics is at 60251 and 63857.

Locations 146 - 202 are reserved for the 8K BASIC ROM. Locations 176 - 207 are reserved by the Assembler/Editor cartridge for the user's Page-0 use. The DEBUG routine also reserves 30 bytes in page-0, the locations are: $A4, $A5, $AD, $AE, $DB - $E5, $EA - $F1, $F5, $F6, $F9 - $FB, $FE and $FF. Should you affect these locations and re-enter the Editor then don't expect the system to be kind to you.

182 B6 DATAD

The data element being read. Registers the number of the element in the DATA line.

183,184 B7,B8 DATALN

Data statement line number; the Basic line number of the DATA statement currently being read. The RESTORE statement resets DATAD and DATALN back to 0.

186,187 BA,BB STOPLN

This is the line where a Basic program stopped either due to an error or the use of the Break key. Also due to a Basic STOP or where a TRAP statement occured. Try the following:

```
10 TRAP 30
20 ;this is a deliberate error
30 LINE=PEEK(186)+256*PEEK(187)
40 ? "Are you aware of error ";PEEK(195);" at line ";LINE
50 TRAP 40000
```

190 - 202

190 BE SAVCUR

Saves current line address.

192 C0 IOCMD

I/O command.

193 C1 IODVC

I/O device.

194 C2 PROMPT

Prompt character.

195 C3 ERRSAVE

This is the most recent error. See STOPLN.

200 CA COLOUR

Stores the colour number used in a Plot or Drawto operation. The statement COLOR X can be replaced with POKE 200,X. Same as 763, except that Basic takes the value from here to load into 763.

201 8D PTABW

This location specifies the number of columns between TAB stops. The 1st tab is at PEEK(201), the default is 10. Note that this is the value used by the "," after the PRINT statement and NOT the actual tab stops used by the TAB key. The minimum value here is 3, a 2 POKEd here will give 4 spaces and 1 is treated as 3. A POKE 201,0 will cause the system to hang at the next PRINT statement using the ",".

202 CA LOADFLG

Load in progress flag. Initialized to 0, if you POKE here with 1 within your Basic program then the program will wipe itself from memory upon return to direct mode (program break).

203 - 229

203-209 CB-41 ...

Unused; free for use.

210,211 D2,D3 ...

Basics floating point work area; $D2 is used for the variable type and $D3 for the variable number and length of the mantissa.

212,217 D4,D5 FRO

Used by the USR command to return a 2-byte number to Basic. If you store nothing here (212 and 213), then the equation: "I=USR(address,variables)" returns the address of the USR subroutine. Otherwise, you can store an integer (range 0 - 65535) here which becomes the value of the USR function.
To use 16-bit values in FP, you would place the 2-bytes of the number into the least 2-bytes of FRO at 212 and 213, and then do a JSR $D9AA, which will convert the integer to its FP representation, leaving the result in FRO. To reverse the operation, do a JSR $D9D2.

Locations 212 - 255 are reserved for page-0 floating point package use. The FP routines are in ROM at 55296 - 57393 ($D800 - $E031). These page-0 locations may be used if the FP package is not called by the users program, however, do not use these locations for an interrupt routine since such routines might occur during an FP routine called by Basic which will cause the system to crash.
Floating point uses a 6-byte precision. The 1st byte of the Binary Coded Decimal (BCD) number is the exponent (where if bit-7 equals 0, the number is positive, and 1 for negative). The next 5-bytes are the mantissa. See De Re Atari for an explanation of BCD (or take up a City and Guilds 223 course!). Also see the NUMBER SYSTEMS appendix

218-223 DA-DF FRE

FP extra register (?).

224-229 E0-E5 FR1

FP register 1; holds a 6-byte internal form of the FP number as does FRO. The FP package frequently transfers data between these 2 registers and uses both for 2 number arithmetic operations.

230 - 246

230-235 E6-EB FR2

FP register 2.

236 EC FRX

FP spare register.

237 ED EEXP

The value of E (the exponent).

238 EE NSIGN

The sign of the FP number.

239 EF ESIGN

The sign of the exponent.

240 F0 FCHRFLG

The 1st character flag.

241 F1 DIGRT

The number of digits to the right of the decimal.

242 F2 CIX

Character (current input) index. Used as an offset to the input text buffer pointed to by INBUFF.

243,244 F3,F4 INBUFF

Input Ascii text buffer pointer; the users program line input buffer, used in the translation of Atascii code to FP values. The result output buffer is at 1408 - 1535 ($580 - $5FF).

245,246 F5,F6 ZTEMP1

Temporary register.

247,248 F7,F8 ZTEMP4

Temporary register.

249,250 F9,FA ZTEMP3

Temporary register.

251 FB RADFLG

Also called DEGFLG. When set to 0, all trigonometric functions are performed in radians; when set to 6, they are done in degrees. Basics NEW command and Reset restore RADFLG to radians.

252,253 FC,FD FLPTR

Points to the users FP number.

254,255 FE,FF FPTR2

Pointer to the users 2nd FP number to be used in an operation.

END OF PAGE-0 RAM

PAGE-1: THE STACK

Locations 256 - 511 ($100 - $1FF) is the stack area for the OS, DOS and BASIC. Machine language JSR, PHA, PHP and interrupts all cause data to be written to the stack, while RTS, PLA, and PLP instructions all cause data to be read from the stack. Upon power-up, the stack-pointer points to location 511, but as items are pushed onto the stack the pointer is lowered and the item is pushed on top. In the case of the pointer going below location 256, it is wrapped-around to point back to location 511.

PAGES 2 - 4

Locations 512 - 1151 ($200 - $47F) are used by the OS for working variables, tables and data buffers. In this area, 512 - 553 are used for interrupt vectors, 554 - 623 are for miscellaneous use. Much of pages 2 - 5 cannot be used except by the OS unless specifically noted.

There are 2 types of interrupts: Non-Maskable Interrupts (NMI) processed by the ANTIC chip, and Interrupt ReQuests (IRQ) processed by POKEY and PIA. NMI's are for the Vertical Blank Interrupt (VBI), Display List Interrupt (DLI) and Reset Key Interrupt (RKI) at locations 546 - 549, 512 - 513, and 12 - 13, respectively.
IRQ's are for the TIMER interrupts, peripheral and serial-bus interrupts, break and 'other' key interrupts. See NMIST at 54287 and IRQEN at 53774.

512,513   200,201   VDSLST

Vector for the NMI display list interrupts; containing the address of the instructions to be executed during a DLI. It's needless me trying to explain DLI's to you if you don't understand them because they are for the people who know what they are doing! If you want to find out about them then you should get hold of a good book such as DE RE ATARI (which is now out of print, like most Atari books really!), on the other hand, you can write to me and ask for my TUTORIAL on DLI's for the Basic programmer which I consider to be a good introduction to DLI's. Anyway, a DLI is best used in altering COLOUR registers at various points across or down the screen, hence, you can have more than 4 colours in GRAPHICS 15 or whatever mode you like.
The OS doesn't use DLI's, they must be user enabled at $D40E, written into protected memory (such like Page-6) and Vectored to through VDSLST.
VDSLST is initialized to 49358 which is just an RTI instruction. As an example for those who are really enthusiastic about learning DLI'S, try this program:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 FOR I=0 TO 13
40 READ D:POKE 1536+I,D:NEXT I
50 DATA 72,173,10,210,41,240,141,10,212,141,24,208,104,64
60 POKE DL+2,240:POKE DL+3,194
70 FOR I=6 TO 28:POKE DL+I,130:NEXT I
80 POKE 512,0:POKE 513,6
90 POKE 54286,192
```

You may notice that after running the program, when you press a key, the colours tend to flick down one line. This is because the keyboard interrupt stores a value into WSYNC at 54282. There are several solutions, see the DLI appendix.

514 - 523

There is only the 1 DLI vector, so if you wished to execute more than 1 DLI you must include within each DLI, address changes to VDSLST to link the DLI's.

514,515 202,203 VPRCED

Serial (peripheral) proceed line vector, initialized to 49357 which is PLA, RTI. It is used when an IRQ interrupt occurs due to the serial I/O bus proceed line which is available for peripheral use. This interrupt is handled by the PIA chip and can be used to provide more control over external devices.

516,517 204,205 VINTER

Serial (peripheral) interrupt vector, initialized to 49357. Used for the IRQ interrupt due to a serial bus I/O interrupt. Processed by PIA.

518,519 206,207 VBREAK

Software break instruction vector for the 6502 BRK command. This IRQ vector is normally used for setting break points in an assembly language debug operation. You can use it when executing a BRK instruction in your own machine language programs, very handy for LOSING hackers trying to hack your machine-code program, just take program flow into a BRK instruction, ensuring that you have setup this vector. Only the more knowledgeable hackers will realise where to go when a DEAD-END is encountered (the BRK instruction).

520,521 208,209 VKEYBD

POKEY keyboard interrupt vector, used for an interrupt generated when any keyboard key is pressed excluding the Break key and the Console buttons. The OS doesn't generate an interrupt for the console keys, see 53279.
VKEYBD can be used to process the key-code prior to it undergoing Atascii conversion. Initialized to the OS keyboard IRQ routine at 64537.

522,523 20A,20B VSERIN

POKEY IRQ serial input ready vector, initialized to 60204 which is the OS routine to place a byte from the serial input port into a buffer. Called INTRVEC by DOS, it is used as an interrupt vector location for an SIO patch. DOS changes this vector to point to 6691, the start of the DOS interrupt ready service routine.

524,525 20C,20D VSEROR

POKEY IRQ serial output ready vector, initialized to 60077 which is the OS routine to provide the next byte in a buffer to the serial output port. DOS changes this vector to 6630, the start of the DOS output needed interrupt routine.

526,527 20E,20F VSEROC

POKEY IRQ serial bus transmit complete interrupt vector, initialized to 60140 which sets a transmission done flag after the checksum byte is sent.
SIO uses VSERIN, VSEROR and VSEROC to control serial bus communication with the serial bus devices. During serial bus communication all program execution is paused. Only stage-1 VBlank and the various IRQ's are constant, even DLI's are inactive during actual transmission of bits. The actual serial I/O is interrupt driven; POKEY waits and watches for a flag to be set when the requested I/O operation is complete. During this wait, POKEY is sending/receiving bits along the serial bus. When an entire byte has been transmitted the necessary IRQ is generated according to data-flow, causing the next byte to be transmitted until the entire buffer has been sent/received whereby the "transmission done" flag is set. At this time SIO exits back to the calling routine, re-enabling DLI's and stage-2 VBI. If the buffer is greater than "X" bytes then there will be a momentary update in any activated DLI's. Where "X" is the seperation of a sector for a disk device (128 bytes) or a record for a cassette device (132 bytes) etc..
It can also be seen that SIO is a serious time-waster where it waits for POKEY to handle its I/O of bits.

528,529 210,211 VTIMR1

POKEY IRQ timer-1 interrupt vector initialized to 49357 (PLA, RTI). Timer interrupts are executed (if enabled at IRQEN) when their associated AUDF register counts down and reaches 0. VTIMR1 uses AUDF1 at 53760. Values in the AUDF registers are loaded into STIMER at 53769 according to mapping, but you can't read it because it has a different purpose.

530,531 212,213 VTIMR2

POKEY IRQ timer-2 interrupt vector for AUDF2. Initialized to 49357. AUDF2 is its associated counter.

532,533 214,215 VTIMR4

POKEY IRQ timer-4 interrupt vector for AUDF4, initialized to 49357. Associated counter is AUDF4.

The HARDWARE-TIMERS are used to count intervals less than a jiffy (1 fiftieth of a second). They count down from a user set value until they reach 0 whereby they vector to the appropriate address. These are very handy for many applications including music durations, game I/O clock, colour alterations, timing and even digitized speech (see the VOLUME-BIT appendix).
On the next page there is a series of steps helping you to make your own hardware interrupt. I've also written a program that uses hardware timer-1 where other manuals couldn't be bothered:

```
10 POKE 53768,0
20 FOR I=0 TO 12
30 READ D:POKE 1536+I,D:NEXT I
40 DATA 173,10,210,41,240,141,10,212,141,24,208,104,64
50 POKE 528,0:POKE 529,6
60 POKE 53760,30
70 POKE 16,193:POKE 53774,193
80 POKE 53769,1
```

1. POKE AUDCTL with the clock frequency you wish to operate in: 0=64KHz, 1=15KHz and 96=1.79MHz. The PAL system actually works at 2.217MHz, but it seems that POKEYs IRQ' are strapped to this strict timing circuit! (It doesn't seem possible to disable Pokeys internal clock for faster processing IRQ'!??).
2. Mapping says to set the channel control register at 53761, to what and why it doesn't say, but when I was fiddling around with it I found that it has no use at all!
3. Place your machine-language interrupt routine into a safe place of memory making sure it ends with a PLA and RTI. Note that if you use the X or Y registers then you should PHA them and restore them at the end of the interrupt.
4. Address your routine with the appropriate Timer-vector.
5. POKE a value between 0 - 255 into the relevant AUDF register. This is the delay (in clock-pulses) before the interrupt routine is re-executed. You should be very careful with this value because if it is shorter than the amount of time your interrupt-routine needs to fully execute then you are dicing with trouble. The system can CRASH.

6. Enable your interrupt by setting its bit in IRQEN at 53774 and its shadow POKMSK at 16.
7. Finally, POKE a nonzero value into STIMER at 53769 so that your counter (the AUDF register) is reset to the value you poked here in step-5.

Hows that for a full description of the hardware timers? Why couldn't mapping do this!

534,535 216,217 VIMIRQ

The IRQ IMMEDIATE vector (general), initialized to 49200. This interrupt is used by the OS to determine the cause of the IRQ so that it can process the correct one.
When playing a sampled file (digitized sounds), the VIMIRQ IRQ can be used quite affectively. See the VOLUME-BIT appendix.

Locations 536 - 558 exluding VVBLKI, VVBLKD, SRTIMR and INTEMP are used for the SOFTWARE-TIMERS. These timers are used to count intervals in jiffies (frames). When they are set, their counters are decremented every 50th of a second and when 0 is reached then depending on the timer, either a flag will be set or a JMP to address will be executed.

536,537 218,219 CDTMV1

System Timer-1 value. This timer is decremented every stage-1 VBlank. When it reaches 0, a flag is set and a JSR is made through the address in CDTMA1 at 550,551. Since the OS uses this timer for its SIO routines, it's best to avoid use of this timer. If you have to use it then do not have it interfering with SIO operations.

538,539 21A,21B CDTMV2

System Timer-2 value. Decremented every stage-2 VBlank. It can be decremented every stage-1 VBlank, subject to the status of CRITIC at 66. This timer may miss (skip) a count when time-critical code is being executed, see the VBI appendix. CDTMV2 performs a JSR through CDTMA2 at 552,553 when its value is 0.

540,541 21C,21D CDTMV3

System Timer-3 value. Same as CDTMV2, timers 2, 3, 4 and 5 are all stopped (from decrementing) when CRITIC is nonzero. Of course, you can write your own VBI and change any of the software timers so that they all use stage-1 (never CRITICal code and always active) VBlanks. CDTMV3 is used by the OS to Open the cassette recorder and also to set the length of time to read/write tape headers.

542 - 547

542,543 21E,21F CDTMV4

System Timer-4; Same as CDTMV2 except that this timer sets a flag to indicate its counted to 0.

544,545 220,221 CDTMV5

System Timer-5; Same as CDTMV4. NOTE that timers 3, 4 and 5 set flags when they have reached 0, where the 1st 2 timers JSR through its appropriate address.

546,547 222,223 VVBLKI

VBlank immediate vector. Initialized to 49378 which is the OS NMI interrupt processor routine. The NMI status register NMIST at 54287 is tested by the hardware to find the cause of the NMI. If the cause is the DLI NMI then vector through VDSLST. If not, then a test is made to see if it's the VBI NMI, if so, then vector through VVBLKI which in turn vectors through VVBLKD if CRITIC is 0. If the NMI isn't any of the above then process the Reset-key routine and vector through DOSVEC.
See the VBlank appendix for a full description of the OS VBlank processes.
If you wish to write your own immediate VBI, then you should put its address here and enable the VBI bit in NMIEN at 54286. Note, however, that to set the address in VVBLKI, you should load the Accumulator with 6, the X register with the HI byte, the Y register with the LO byte and JSR SETVBV at $E45C. Your interrupt program doesn't need to PHA or PLA any registers, but it does need to exit the routine with a JMP to SYSVBV at $E45F. Also see appendix D5.

```
10 DATA 173,242,2,141,26,208,76,98,228
20 DATA 104,169,6,162,6,160,0,32,92,228,96
30 FOR I=0 TO 19
40 READ D:POKE 1536+I,D:NEXT I
50 X=USR(1545)
60 POKE 54286,64
```

This program sets up an immediate VBI to use the value of the last key pressed as a BORDER colour. Not a very ingenious program for a stage-1 VBI, but nonetheless, quite affective. Because the program disables the original stage-1 VBlank, you will notice that the Real-Time Clock is not updated and all the other stage-1 functions are not implemented: for instance, try several PEEKs at location 20.

548 - 555

548,549 224,225 VVBLKD

VBlank deferred vector. Initialized to 49802. You can use the above program in the deferred register by changing the 1st occurrence of 6 placed in the Accumulator to a 7, thus, retaining all the original stage-1 processes AND stage-2. See the TIMINGS appendix for time calculations.

550,551 226,227 CDTMA1

System Timer-1 JuMP address is initialized to 60433. When locations 536,537 have counted down and reached 0, the OS vectors through here. The OS uses this timer from stage-1 VBlank so you can either use another timer so as to reduce any OS conflicts or you can reconfigure the VBlank so that it doesn't use this timer (the latter is probably best avoided!).
Mapping says that you should avoid using numbers greater than 255 because a VBI could occur when the LSB goes negative and the MSB is to be updated, but I fail to see how this is possible because the timers are decremented DURING the VBI, thus, telling us that unless it takes a whole frame to decrement 2 locations then this has no possibility of happening and is undersight by Ian Chadwick.

552,553 228,229 CDTMA2

System Timer-2 JuMP address. Unused by the OS and free for you to write the address of your machine-language routine here. Initialized to 0 on power-up.

554 22A CDTMF3

System Timer-3 FLAG; set positive when CDTMV3 has reached 0. This register is also used by DOS as a time-out flag.

555 22B SRTIMR

Software repeat timer, controlled by the IRQ device routine. It establishes the initial half-second delay before a key will repeat itself if depressed. Stage-2 VBlank establishes the initial 0.8 of a second repeat rate, decrements SRTIMR and implements auto repeat logic. Every time a key is pressed, SRTIMR is loaded with 40. Whenever SRTIMR reaches 0 and a key is still pressed, the value of that key is continually stored in CH at 764.

556 22C CDTMF4

System Timer-4 FLAG; set when CDTMV4 counts down and reaches 0.

557 22D INTEMP

Temporary register used by the SETVBL routine at 58460.

558 22E CDTMF5

System Timer-5 FLAG; set when CDTMV5 counts down and reaches 0.

559 22F SDMCTL

Direct Memory Access (DMA) enable, initialized to 34. Shadow location for 54272 ($D400), POKE with 0 to turn ANTIC off (including the display) to speed processing up 30%. If your performing a routine that needs speeding-up, but you still require some display then there are 2 ways of achieving this: the 1st is simply by replacing the mode lines that you don't need with Blank-Scan Lines (BSL's) or even just shrinking the DL. The other method is to use 1 or more DLI's to turn Antic off during the area's of the screen that is unused.
Here's a list of the bits in this register, just add up the value's to achieve what you want. Note that you can only have 1 playfield:

BITS:	DEC:	OPTION:
0	0	No playfield
0	1	Narrow playfield
1	2	Standard playfield
0,1	3	Wide playfield
2	4	Enable missile DMA
3	8	Enable player DMA
2,3	12	Enable missile and player DMA
4	16	One line player resolution (double-line res. if not set)
5	32	Enable "DMA FETCH INSTRUCTION"

Note that the Double-line res. is default if the Single-line res. is not chosen. Also, if you wish the playfield or the PMG DMA to appear then you must set bit-5 along with the bits you need.

The playfield is the text/graphics area of the screen. Narrow playfield is 128 colour clocks (there are 4 colour clocks to 1 Graphics 0 byte in width), thus, giving 32 columns. The standard playfield is 160 colour clocks and 40 columns across. Wide playfield is 192 colour clocks and 48 columns wide.
A colour-clock is a physical measure of horizontal distance across the screen, there are a total of 228 colour-clocks across 1 scan-line, but only around 176 are visible. A pixel on the other hand is a logical unit which varies in colour clocks depending on the Graphics mode you choose.
Bit-5 should be enabled so that Antic can use its DMA to fetch the DL instructions, the memory bytes and the PMG data. If it's not set, then there will be no display and the processor will work 30% faster as mentioned earlier.
Bits-6 and 7 don't seem to be used for anything and are clear.

560,561 230,231 SDLSTL/H

Starting address of the Display List (DL). The DL is an instruction-set which tells Antic where the screen-data is and how to display it. Shadow for DLISTL/H at 54274,5. You can find the DL 1-byte above free memory by using:

```
DL=PEEK(741)+256*PEEK(742)
```

But, don't get into the habit of using that particular method, the method you should always use is:

```
DL=PEEK(560)+256*PEEK(561)
```

When you call a Graphics mode, the appropriate display is created from the tables at 60957. See locations 88 and 89. You can create your own DL with mixed text/graphic displays, fine-scrolling, BSL's and DLI's. See the table below:

BITS:	DEC:	FUNCTION:
7	128	Display-List Interrupt (DLI)
6,1	65	Jump and wait for vertical blank (JVP)
6	64	Load Memory Scan (LMS)
5	32	Vertical fine-scroll
4	16	Horizontal fine-scroll
0	1	Jump code (JMP) ;not 6502 JMP

The above is a list of the functions available on the DL, the text/graphic modes are in bits 0, 1, 2 and 3 and described on the next page.

561 cont.

BITS 3-0:	DEC:	GRAPHICS:
0 0 1 0	2	0
0 0 1 1	3	0.5
0 1 0 0	4	12
0 1 0 1	5	13
0 1 1 0	6	1
0 1 1 1	7	2
1 0 0 0	8	3
1 0 0 1	9	4
1 0 1 0	10	5
1 0 1 1	11	6
1 1 0 0	12	14
1 1 0 1	13	7
1 1 1 0	14	15
1 1 1 1	15	8,9,10 and 11

The text modes have bit-3 clear, while the graphic modes have bit-3 set. Graphics 0.5 has 10 rows to a byte rather than 8 and is especially useful for true descenders in text. Graphics 9,10 and 11 are obtained by selecting this code, but also by setting the appropriate bits in location 623. See this location for further information.
There are also Blank-Scan Lines (BSL's) in the DL instruction set:

BSL's (amount):	DEC:	BITS:
1	0	none
2	16	4
3	32	5
4	48	4,5
5	64	6
6	80	4,6
7	96	5,6
8	112	4,5,6

You'll notice that the DL instructions are contradictory in some of the bits, for example: fine-scrolling is on bits 4 and 5 whilst 2 and 3 BSL's uses those bits too. This is quite right, but you should know that the fine-scrolling bits are only so, when a text/graphics mode is active. If no mode bits are selected, then they are treated as BSL's. This is also the case for several other bits, and because of the detail needed to describe the DL, this is only meant as a reference. If you want a good explanation of the DL, then you should get hold of De Re Atari or Your Atari computer by Lon Poole. There is also a good tutorial on DL's in Page-6 magazine, issues 18 - 20 by Steve Pedler.

562

If your making a DL of your own you should put your DL in a safe area of memory and POKE its address here, you should also ensure that the DL instructions follow the FACT table below:

DEC:	BIT:	FUNCTION:
128	7	This value is the DLI request. It can be an instruction of its own or SET with any other bits and still means the same.
64	6	This value without any mode bits selected means 5 BSL's, with bit-0 set (65) it becomes the JVP instruction which must always end every DL. You must follow this instruction with the LSB/MSB start address of the DL (the contents of SDLST). If set with mode bits, then it becomes the LMS instruction. LMS is used to point to which memory is to be displayed. It should also be followed with the LSB/MSB address of display memory (usually the address found in SAVMSC at 88,89).
32	5	This value without any mode bits selected means 3 BSL's, with mode bits set it becomes the Vertical fine-scroll enable bit.
16	4	This value without any mode bits set means 2 BSL's, with mode bits set it becomes the Horizontal fine-scroll enable bit.
1	0	This value without bit-6 set is the Antic JMP-instruction, it is used to tell Antic that the DL continues at the address given in the next 2 bytes (LSB/MSB). This must be used to stop your DL going through a 1K boundary. See the BOUNDARIES appendix.

DL's are restored on Reset and Graphics calls, replace yours by re-POKEing its address here.

562 232 SSKCTL

Serial port control register, initialized to 19 which sets bits 0, 1 and 4. Shadow for 53775. The bits in this register are:

BIT:	DEC:	FUNCTION:
0	1	Enable keyboard debounce circuit,
1	2	Enable the keyboard scanning circuit,
2	4	The POT-scan completes a read within 2 scan-lines instead of the usual 1-frame time.
3	8	Serial output transmitted as 2-tone mode instead of logic true/false (POKEY 2-tone mode)
4-6	16-64	Serial port mode control.
7	128	Force BReaK; serial output to 0.

563 233 SPARE

Temporary counter for the peripheral handler loader.

564 234 LPENH

Light-pen horizontal value; shadow for 54284, values range between 0 - 227.

565 235 LPENV

Light-pen vertical value; shadow for 54285. The values here are the same as the VCOUNT register for two-line resolution. Both light-pen values are modified when the trigger is pressed (pulled low). The light-pen positions are not the same as the normal screen row and column positions. There are just 96 vertical positions, numbered from 16 at the top to 111 at the bottom, each one equivalent to a scan-line. There are 228 horizontal positions numbered from 67 at the left. When the LPENH value reaches 255, it is reset to 0 and begins counting again by one to the rightmost edge, which is a value of 7.
Obviously, because of the number of positions readable and the small size of each, some leeway must be given by the programmer when using light-pen read-outs in a program.

566,567 236,237 BRKKEY

BREAK-key IRQ interrupt vector, initialized to 49298. This vector can be used for your own machine-language routine, remember to end your routine with a PLA and RTI sequence.

568,569 238,239 RELADR/VPIRQ

In the 1200XL, this is the address of the relocatable handler routine. In all other XL's and XE's, it's the vector for parallel bus interrupt request and points to 51566 which is the vector for any initialized generic parallel device.

570 23A CDEVIC

The current SIO bus ID (device) number.

571 23B CCOMND

The SIO bus command code.

572 - 580

572 23C CAUX1

Command auxiliary byte-1, loaded down from 778 by SIO.

573 23D CAUX2

Command auxiliary
Command auxiliary byte-2, loaded down from 779 by SIO.

574 23E TEMP

Temporary RAM register used by SIO.

575 23F ERRFLG

SIO error flag; any device error except the time-out error (time = 0).

576 240 DFLAGS

Disk flags read from the 1st byte of the boot file (sector 1) of the disk.

577 241 DBSECT

The number of disk-boot sectors read from byte-2 of the 1st sector.

578,579 242,243 BOOTAD

This is the beginning address in memory to put the disk-boot program. This address is read from bytes 3 and 4 from the 1st sector on a disk. DOS normally has 1792 as its start address. The OS routine to load the disk program is called DOBOOT and is located at 50571.

580 244 COLDST

Coldstart flag. If this register is 0 then pressing Reset results in a warmstart, however, POKEing here with nonzero and pressing Reset results in coldstart (re-booting of the computer).
If you create an AUTORUN.SYS file, it should end with an RTS. If not, then it should clear 580 and set location 9 with 1. You can make any binary file automatically load when booting a DOS disk by renaming it to AUTORUN.SYS.

581 - 584

Be careful not to have more than 1 filename in the directory under the same name, because when you use the delete-file option from DOS, it deletes everything under the name you give to it. In case you do have 2 files on the disk under the same name, then you can POKE 3118 with 0 and then use the rename option of DOS. It will only change the name of the 1st match of the name you give, thus, when you have the 2 files under seperate names, you can delete just the one you don't want.
COLDST can also be used along with locations 16, 566, 567, 138, 139 and 202 to achieve a very affective protection for Basic programs. They can be protected from listing and breaking into. Copy protection is another matter, however, it is really a case of having the right hardware so that particular areas of the disk containing the protected program are unformatted, and even in some cases formatted in an uncopyable manner.

581 245 RECLEN

Relocatable loader routine variable for record length.

582 246 DSKTIM

Disk time-out register (address of the OS's worst time-out). Default is 160, giving a total time-out period of 2 minutes 50 seconds. It's updated after each disk status request to contain the value of the 3rd byte of the status frame (location 748). All disk operations have a 7 second time-out. The old ROMS had a real irritating delay which was a BED-BUG. It occured in the FORMAT operation as well as printers.

Locations 583 - 618 are unused on the 1200XL and therefore free for use. On other XL's/XE's, they are as follows:

583 247 PDVMSK

Shadow mask for the device selection register at 53759, active only when the OS deselects the FP ROM by writing to that address. You can run up to 8 parallel devices through the bus, each bit in this register corresponds to 1 device. The mask must be set for the proper device before the OS will allow an IRQ to be sent to that device.

584 248 SHPDVS

Shadow for the parallel bus register; each bit represents 1 of the 8 parallel devices. This allows the OS to service VBI's while running the device masked by the appropriate bit.

585 - 622

585 249 PDMSK

Parallel bus interrupt mask; allows the OS to service IRQ's from the device masked by the bit in this register.

586,587 24A,24B RELADR

Relocatable loader relative address.

588,589 24C,24D PPTMPA,PPTMPX

1 byte temporary storage registers for the relocatable loader.

590-618 24E-26A ...

Unused; free for use.

619 26B CHSALT

Alternate character set pointer for the 1200XL, initialized to 204 to point to the international character-set as the next set to display on the F4-key toggle. The XL/XE have 2 character sets, the 1st at 52224 and the other at 57344.

620 26C VSFLAG

Fine-scroll temporary register.

621 26D KEYDIS

Keyboard disable. POKE with 255 to disable the keyboard and 0 to re-enable. You have to press Reset to re-enable the keyboard if in Basic except on the 1200XL where you can press CTRL+F1. This is also one cure for removing the DLI flicker. If you disable the keyboard, the OS does not execute the keyboard routine, thus, it does not store any value into WSYNC.

622 261 FINE

Fine-scroll enable for Graphics 0. Poke with 0 for coarse scrolling (default), or with nonzero for fine scrolling. Try POKE 622,255 and calling Graphics 0. When you list a long program you will notice something quite unique when the listing scrolls up the screen. The OS places the address 64708 of a DLI at 512 and 513, replacing any DLI you might already have there. The colour register at 53271 is altered for the last visible screen line.

If you enable fine-scrolling here and go to DOS, you'll see that it remains enabled if you display a directory to the screen.

623 26F GPRIOR

Priority selection register. Shadow for 53275. Priority options select which screen objects will be in front of others. It also allows you to combine the 5 Missiles into a 5th player, certain overlapped players can have an EOR'd colour too. Here are the bit functions:

BITS:	DEC:	FUNCTION:
		(priorities)
0	1	Players 0-3, playfields 0-3, Backbround,
1	2	Players 0-1, playfields 0-3, players 2-3, background,
2	4	Playfields 0-3, players 0-3, background,
3	8	Playfields 0-1, players 0-3, playfields 2-3, background.
		(Other options)
4	16	4 missiles assume same colour for 5th player,
5	32	Overlap of players have 3rd colour,
6	64	GTIA mode 9
7	128	GTIA mode 10
6,7	192	GTIA mode 11

You should normally select only 1 of the priorities, although, if you select more than 1 then any priorities at the same level will just black-out when overlapped. I can't see any useful application to put this to, but I'm sure it can be of some use.
With the 3rd colour overlap you can achieve a multicolour player by using more than 1 player above each other. The overlapping of colours is done on players 0 with 1 and 2 with 3, only these combinations are allowed, thus, you will not get a 3rd colour by overlapping players 0 with 2 or 3, and 1 with 2 or 3. All you will get is a black-out.
Bits 6 and 7 have a completely different meaning, they are used to obtain the GTIA modes. See SDLST at 560,1. When changing the DL to obtain the GTIA modes, you should use the Antic code given in the table and use the appropriate POKE here. The really good thing with this method of achieving GTIA modes is that you don't have to setup the GTIA DL for these POKE values to work. Why not try:

```
10 GRAPHICS 2+16
20 DL=PEEK(560)+256*PEEK(561)
30 DM=PEEK(DL+4)+256*PEEK(DL+5)
40 FOR I=1 TO 20*12
50 POKE DM+I-1,PEEK(53770):NEXT I
60 FOR I=64 TO 255
70 IF I/64=INT(I/64) THEN POKE 623,I
80 NEXT I
90 GOTO 60
```

Here's a program you can use to see all the GTIA modes in action, just change the mode between 9 - 11:

```
10 GRAPHICS 9
20 FOR I=0 TO 6
30 POKE 705+I,I*32+8
40 NEXT I
50 FOR I=0 TO 79
60 COLOR INT(I/5.26)
70 PLOT I,I:DRAWTO 79-I,I
80 DRAWTO 79-I,191-I:DRAWTO I,191-I
90 DRAWTO I,I
94 GOTO 94
```

GTIA mode pixels are long and flat, their ratio being 2:1 (colour clocks to scan-lines), which isn't a very good horizontal resolution for detailed work, curves or circles, but they have a lot of colours/shades which when used affectively can give some remarkable graphic affects! Have you seen the Atari' graphics demonstration disk? There is the Robot and the Spaceship demo which are excellant examples. There are also digitised photo's that give many more colours and shades on the screen at one time. If I had a copy of the program, then I would have found out exactly how it's done and given some introduction to it here, but I don't have it so what can I do. I do know that it sets the fast pot-scan at location 53775, though.

Locations 624 - 647 are used for the games controllers:

624 270 PADDL0

The value returned from the position of PADDLE(0). Paddles are also called POTS (short for Potentiometer). The values range between 0 - 228, increasing as the knob is turned counter-clockwise. All PADDLE registers are shadows for POKEY locations 53760 - 53767.

625 271 PADDL1

Same as 624 but for PADDLE(1), which is also on the same controller jack (0).

626 - 637

626 272 PADDL2

PADDLE(2); which is on controller jack 1.

627 273 PADDL3

PADDLE(3); also on controller jack 1.

Locations 628 - 631 are repeats of the last 4 locations, copied here during VBlank stage-2.

632 278 STICK0

This is the value returned from the Joystick in port 0. All joystick locations are shadow for PIA location 54016. Depending on the position of the joystick, the following values are returned:

5 = DOWN-RIGHT	6 = UP-RIGHT
7 = RIGHT	11 = LEFT
9 = DOWN-LEFT	10 = UP-LEFT
13 = DOWN	14 = UP
15 = CENTRE	

633 279 STICK1

Same as 632 except for joystick port 1.

Locations 634 - 635 are repeats of 632 - 633 and are copied here during stage-2 VBlank.

636 27C PTRIG0

Paddle trigger 0. Used to determine if the trigger/button is pressed (returning 0) or released (returning 1). Since these use the same controller port lines as the jostick left and right directions, you could if wanted use PTRIG for horizontal movement. This is a useful addition that Ian Chadwick wrote in mapping. When this register returns a value of 1, a value of 7 is placed into STICK(0), while a 0 returned here returns an 11 to STICK(0). The PTRIG registers are shadows for 54016.

637 27D PTRIG1

Same as 636, but for PTRIG(1).

638 - 650

638 27E PTRIG2

PTRIG(2) register.

639 27F PTRIG3

PTRIG(3) register.

Locations 640 - 643 are repeats of locations 636 - 639 and copied there from stage-2 VBlank.

644 284 STRIG0

Stick trigger 0. This register returns the same values as the PTRIG register, except for the joystick. STRIGs are shadows for 53264 - 53267.

645 285 STRIG1

Same as 644 but for STRIG(1).

Locations 646 - 647 are repeats of locations 644 - 645 copied there by the stage-2 VBlank.

---

648 288 HIBYTE

Hi-byte register for relocatable loader routine.

649 289 WMODE

Flag to indicate to the cassette handler which mode to be in: READ = 0 and WRITE = 128.

650 28A BLIM

Cassette data record buffer size; contains the amount of active/used bytes in the cassette buffer for the record being read or written at location 1021. Values here range between the size of the cassette record, 0 - 127. The pointer to the actual byte being read/written is at location 61. The value for BLIM is drawn from the control bytes preceding every cassette record, as explained in location 1021.

651 - 658

651 28B IMASK

Mapping calls this IMASK, but also says that it's unused.

652 28C JVECK

Temporary jump vector; unused otherwise.

653 26D ...

Unused; free for use.

654,655 26E,26F NEWADR

Used by the relocatable loader routine; new address vector.

Locations 656 - 703 are used for the screen RAM display handler (depending on the Graphics mode). In split-screen mode, the text-window is controlled by the screen editor (E:), while the graphics region is controlled by the display handler (S:), using 2 seperate IOCB's, even if you have a text-window in Graphics 0 (see location 703). 2 seperate cursors are also maintained, though, only the text-window one is visible.

656 290 TXTROW

Text-window cursor row; this value ranges between 0 - 3 coz there are only 4 lines in it. TXTROW specifies the next row to print on or even read from.

657,658 291,292 TXTCOL

Text-window cursor column; values range from 0 - 39, unless changed by the user at 82 and 83. Location 658 will always be 0 unless you change the mode-lines of the text-window by altering them in the DL, see SDLST. However, if you don't change the mode, then location 658 is unused and free for use.
Since Position, Plot and Locate all refer to the upper screen (not text-window), you'll have to use POKE statements to achieve anything you may not be able to get with the Print or CHR$ functions in the text-window.

659 293 TINDEX

Similar to DINDEX, except for the text-window. This is always 0 when location 128 is 0 and is initialized to 0. Remember to put the same mode number here if you change the text-window DL, see above.

660,661 294,295 TXTMSC

Address of the upper left corner of the text-window, obtained with this expression:

```
DMW=PEEK(660)+256*PEEK(661)
```

See locations 88 and 89 also.

662-667 296-29B TXTOLD

These locations are the split-screen equivalents of OLDROW, OLDCOL, OLDCHR and OLDADR.

668 29C CRETRY

Number of command retries; Initialized to 13, this is the number of times a device will attempt to carry out a command such like sector read.

669 29D HOLD3

Temporary register use.

670 29E SUBTMP

Temporary storage.

671 29F HOLD2

Temporary register use.

672 2A0 DMASK

Pixel location mask. DMASK contains the value of the specific pixel last operated upon (from a Plot, Drawto or Poke) within the screen display byte, leaving the unused pixel/s (bits) equal to 0 and the used bits or pixel/s equal to 1. The size of the pixel, or amount of bits, depends on the Graphics mode being used, as follows:

PIXEL SIZE:	GRAPHIC MODES:
11111111	0, 1, 2, 12 and 13 These modes use all the bits of each screen display byte per pixel.
11110000	9, 10 and 11 GTIA modes are configured this way, having 2 pixels per byte. You must note, however, that the screen X-co Plot position 0 sets the high 4-bits, whilst the 2nd pixel sets the low 4-bits. The next pixel sets the high 4-bits, but in the next screen byte, etc..
11000000	3, 5, 7 and 15 These modes are 4 pixels per byte, thus, the 1st pixel in each byte is as shown, the next is 00110000 and so on.
10000000	4, 6, 8 and 14 These have 8 pixels per byte, whereby the 2nd pixel returns 01000000 and so on.

Here's a chart for all the Graphics pixel details:

GR.MODE	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SCAN LINES PER PIXEL	8	8	16	8	4	4	2	2	1	1	1	1	8	16	1	1
BITS PER PIXEL	8	8	8	2	1	2	1	2	1	4	4	4	8	8	1	2
COLOURCLOCKS PER PIXEL	\	1	1	4	2	2	1	1	\	2	2	2	1	1	1	1
BYTES PER LINE	40	20	20	10	10	20	20	40	40	40	40	40	40	40	20	40

Also see location 559 for playfield size.

673 - 689

673 2A1 TMPLBT

Temporary storage for the bit-mask.

674 2A2 ESCFLG

Escape flag. Normally 0, it is set to 128 when ESC is pressed. It is reset to 0 after the next keypress. See location 766 for forced ESC mode.

675-689 2A3-2B1 TABMAP

Map of the TAB-stop positions. There are 15 bytes (15*8 = 120 bits), each bit corresponds to 1 column in a logical line, where a value of 1 means the TAB is set and a 0 means otherwise. If you wish to clear all the TAB stops then you can either poke all these locations with 0 or press the TAB key to land on each tab-stop and press CTRL+TAB, likewise, if you wish to create one then position the cursor where you want the tab-stop and press SHIFT+TAB (or POKE the appropriate bits in). Try the following program:

```
10 DIM C$(8)
16 DATA 128,64,32,16,8,4,2,1
22 FOR I=1 TO 8
28 READ D:C$(I,I)=CHR$(D):NEXT I
34 FOR J=1 TO 15
40 POKE 675+(J-1),0:NEXT J
46 FOR TAB=1 TO 120 STEP 3
52 GOSUB 70
58 NEXT TAB
64 STOP
70 BYTE=(TAB-1)/8
76 BIT=((BYTE-INT(BYTE))*8)+1
82 V=ASC(C$(BIT,BIT))
88 BYTE=INT(BYTE)
94 POKE 675+BYTE,PEEK(675+BYTE)+V
98 RETURN
```

You can use this program to set any TAB positions you wish. The GOSUB routine between lines 70 - 98 actually sets any TAB-stops given to it by the TAB variable (columns are between 1 - 120). In this case, a TAB-stop is set every 3 positions, try changing the FOR/NEXT loop STEP at line 46. If you wish to revert to normal, just hit Reset or call a Graphics mode.

690 - 699

690-693 2B2-2B5 LOGMAP

Logical line start bit-map. The 1st 3 bytes are used to indicate which physical line is the beginning of a logical line. 3 bytes give 24 bits (3*8 = 24), the amount of physical lines on a Graphics 0 display. Where a bit is set, a logical line begins:

LOC:	BIT:	7	6	5	4	3	2	1	0
690	LINE:	0	1	2	3	4	5	6	7
691		8	9	10	11	12	13	14	15
692		16	17	18	19	20	21	22	23

Location 693 is unused and therefore free for use. All the map bits are set to 1 when the screen is OPENed or CLEARed, when a Graphics call is made or when Reset is pressed. The map is updated as logical lines are entered, edited or deleted.

694 2B6 INVFLG

Inverse character flag, initialized to 0. If you wish to force inverse character mode then POKE with 128. This is also the OS technique when you press the inverse key. The display handler EORs the Atascii codes with the value here at all times. See location 702.
You can poke other values here and mix the keyboard characters around.

695 2B7 FILFLG

Screen Fill or Draw flag. 0 means the current operation is DRAW, nonzero means FILL. Use this location in conjunction with ROWCRS, COLCRS, OLDROW, OLDCOL and ATACHR.

696 2B8 TMPROW

Temporary register for row, used by ROWCRS.

697,698 2B9,2BA TMPCOL

Temporary registers for column, used by COLCRS.

699 2BB SCRFLG

Scroll flag; set if a scroll occurs. It counts the number of physical lines minus 1 that were deleted from the top of the screen. This moves the entire screen up 1 physical line for each line scrolled off the top. Since a logical line is 3 physical lines, SCRFLG ranges between 0 - 2.

700 - 703

Scrolling the text window now only scrolls the correct amount of memory, freeing the system of a nasty bug which used to wipe-out memory above RAMTOP!

700 2BC HOLD4

Temporary register used in the DRAW command only; it's used to save and restore the value in ATACHR during the FILL process.

701 2BD DRETRY

Number of device retries.

702 2BE SHFLOK

Flag for the SHIFT and CTRL keys. 0 means lowercase mode, 64 is uppercase mode and 128 is Control lock mode. Other values POKEd here may cause the system to crash. See also location 694.

703 2BF BOTSCR

Flag for the number of text rows available for printing. In Graphics mode 0 this is 24, while it is 4 for the text window.
You can add a text window in any mode by POKEing here with 4. DOS does this on the DUP.SYS menu when awaiting your input.

Locations 704 - 712 are the shadow colour registers for players, missiles and playfields. The hardware registers are at 53266 - 53274 ($D012 - $D01A). For the playfield registers, locations 708 - 712, you can use the SETCOLOR command from Basic. The other registers you'll need to POKE directly.
The format for POKEing the colour registers is:

COLOUR = HUE*16 + SHADE

Although, you have 16 colours and 16 shades of each colour in the XL/XE, you are limited to the use of these depending on what mode your in. Graphics 0 and 8 are mono-modes, you can normally only have 2 colours in these modes and a 3rd colour which must be a luminance from 1 of the other 2. All other Graphics modes allow a maximum of 5 colours except for the GTIA modes. In the GTIA modes you can either have 16 shades of 1 colour, 9 different colours or shades, or 16 colours of 1 shade. It is possible to actually have all colours and all shades in a GTIA mode if you perfect a technique with the POT-SCAN at location 53775 ($D20F), see GPRIOR.

703 cont.

You can also use DLI's and the Hardware-timers to change the colour registers "on the fly", thus, enabling you to achieve many more colours down and across the screen display (even on the mono modes). The amount to which you can go to is really unknown.
Another method of obtaining more colour in Graphics 8 is by using a technique known as artifacting. See De Re Atari for further info. on this and location 710.

The 16 colours inside the classic Atari are as shown in the table below:

COL. NUM:	COL. VALUES:	DARK - MEDIUM - LIGHT
1	0 - 15	Black through grey to white
2	16 - 31	Dk browny orange through to pale orange
3	32 - 47	Red brown, deep pink to light orangy pink
4	48 - 63	Med brown, reddish brown to pale pink
5	64 - 79	Red through to rich pink
6	80 - 95	Purple through to pale pink
7	96 - 111	Cobalt blue, pale purple to bluey violet
8	112 - 127	Ultramarine to light blue
9	128 - 143	Dk blue to pale blue
10	144 - 159	Dk cyan to pale cyan
11	160 - 175	Dk green to pale green
12	176 - 191	Med green to shallow green
13	192 - 207	Olive green to light green
14	208 - 223	Browny green to yellow green
15	224 - 239	Browny orange to yellow
16	240 - 255	Dk browny orange through to pale orange

You'll notice that colours 2 and 16 are exactly the same, so does this mean that there are only 15 colours on the Atari? You may also notice that by POKEing the values into the registers, only every other value changes the shade, thus, only giving 8 shades of each colour. So, you should see that there are only 15*8 = 120 shades allowed in every mode except GTIA modes. Whether or not you can obtain all the colours in non-GTIA modes, I don't know, but the Atari does have 256 shades accessible in the GTIA modes.

This is a very briefly described topic in every Atari manual, and should really be investigated further. As you can see, in mapping, colours 2 and 16 are labelled differently which is not the case. My choice of colours are not the same as others, but I believe that they are more explicit because when the shades get lighter, the colour tends to shift very slightly also.

704 - 710

704 2C0 PCOLR0

Colour of player 0, missile 0 and the background colour for GTIA mode 10. Shadow for 53266. You cannot use the SETCOLOR command to change any of the PCOLR registers so you'll have to POKE directly to them.

705 2C1 PCOLR1

Colour for player and missile 1. Shadow for 53267.

706 2C2 PCOLR2

Colour for player and missile 2. Shadow for 53268.

707 2C3 PCOLR3

Colour for player and missile 3. Shadow for 53269.

708 2C4 COLOUR0

Colour register 0 which is playfield 0, controlled by SETCOLOR 0. In Graphics 1 and 2, it is the colour of all uppercase letters. Shadow for 53270 ($D016).

709 2C5 COLOUR1

Same as 708, except for playfield 1, controlled by SETCOLOR 1. In Graphics 1 and 2, it is the colour of all lowercase letters. Shadow for 53271 ($D017).

710 2C6 COLOUR2

Same as 708, except for playfield 2, controlled by SETCOLOR 2. Graphics 1 and 2 Inverse-uppercase register and background colour in Graphics 0 and 8. Shadow for 53272 ($D018).
Despite the official limitations of colour selection in Graphics 8, it is possible to generate additional colours by "ARTIFACTING", turning on specific pixels (\ colour-clock each) on the screen. Taking advantage of the physical structure of the TV-set itself, we can see the affects with the following program quite affectively:

711 - 712

```
10 GRAPHICS 8+16
20 POKE 710,0:POKE 709,15
30 COLOR 1
40 FOR I=0 TO 319 STEP 3
50 PLOT 160+I/3,191:DRAWTO I,90
60 DRAWTO 160+I/3,0
70 NEXT I
80 GOTO 80
```

You should be able to make out 6 colours in this example; white, grey, red, cyan, yellow and blue (7 including the black background). In my opinion, this technique is useful as a background affect to foreground text. Try adding the following routine into the above program:

```
20 POKE 710,0:POKE 709,6
80 FOR I=0 TO 33
82 READ D:POKE 1536+I,D:NEXT I
84 DATA 104,173,11,212,201,40,208,249,169,192,141,27,208
86 DATA 141,10,212,173,11,212,201,84,208,249,169,0
88 DATA 141,27,208,141,10,212,76,1,6
90 X=USR(1536)
```

For further information about the artifacting process, get hold of De Re Atari or BYTE 1982 (!!).

711 2C7 COLOUR3

Playfield 3 register, controlled by the SETCOLOR 3 command. Inverse-lowercase colour in Graphics 1 and 2. Shadow for 53273 ($D019).

712 2C8 COLOUR4

Playfield 4 register, controlled by the SETCOLOR 4 command. Shadow for 53274 ($D01A). This is the border in Graphics 0 and 8, and the background in all other modes except GTIA 10. In GTIA 10, 712 becomes a normal colour register.
Here's a program showing extra colours in the border on Graphics 0:

```
10 GRAPHICS 0
20 FOR I=0 TO 15
30 READ D:POKE 1536+I,D:NEXT I
40 DATA 104,173,11,212,74,101,20,141,10,212,141,26,208,76,1
,6
50 X=USR(1536)
```

713 - 728

The default values for the SETCOLOR registers 0 - 4 are:

REGISTER:	COLOUR: =	HUE:	LUM:
0- 708	40	2	8
1- 709	202	12	10
2- 710	148	9	4
3- 711	70	4	6
4- 712	0	0	0

713,714 2C9,2CA RUNADR

Run address register for the relocatable loader routine.

715,716 2CB,2CC HIUSED

Used by the relocatable loader routine.

717,718 2CD,2CE ZHIUSE

Used by the relocatable loader routine.

719,720 2CF,2D0 GBYTEA

Relocatable loader use.

721,722 2D1,2D2 LOADAD

Relocatable loader use.

723,724 2D3,2D4 ZLOADA

Relocatable loader use.

725,726 2D5,2D6 DSCTLN

Disk sector size register; default of 128 bytes, but can be altered to a length from 0 - 65535. Your drive may not support other sizes, however, you can have different drive chips such like the Archiver which will allow you to configure the disk in different ways.

727,728 2D7,2D8 ACMISR

Interrupt service routine address; unused.

729 2D9 KRPDEL

Auto-delay rate; the time elapsed before keyboard repeat begins, initially set at 40 (but 48 on NTSC) for 0.8 seconds. You can POKE this with the amount of stage-2 VBlank intervals before repeat begins. A value of 50 would be a 1-second delay, where a value of 0 turns the key-repeat off.

730 2DA KEYREP

The rate of key-repeat, initialized to 5 which means 10 characters per second (1 each 5 stage-2 VBlank intervals), POKE with the number of VBlank intervals between repeats; a value of 1 gives 50 characters per second. A value of 0 provides 1 key-repeat only per key press.
Try POKE 729,11 and POKE 730,2 in Basic and hold-down a few keys. I find these delays very suitable for my patience when typing large scripts (such like this one).

731 2DB NOCLIK

This is the keyboard click enable/disable register. 0 equals enable, while nonzero equals disable. On the 1200XL, CTRL+F3 can perform this task. Other XL/XE users might like to know that my program at locations 121 and 122 also puts this click-toggle on a single keypress by adding to the existing key-definition table.

732 2DC HELPFG

HELP-key status: A value of 17 here means the help key has been pressed, 81 means shift and help, while 145 means control and help. A rare value of 209 can be found in this location and means all control, shift and help keys have been pressed, but the shift and control keys must be pressed exactly simultaneously (!).
If detecting for the help key in a program, you must remember to clear it before reading and afterwards because it acts similarly to 764 in retaining the value of the last key combination.

733 2DD DMASAV

DMA toggle value. The value from location 559 is saved here when you turn the screen off on the 1200XL with the CTRL+F2 keypress, so that it can be restored at the next toggled press. On other XL/XE's, if you POKE any value here at the next keystroke, the value is moved into location 559.

734 2DE PBPNT

Print buffer pointer.

735 2DF PBUFSZ

Print buffer size.

736-739 2E0-2E3 GLBABS

Without DOS, these bytes are unused and free for use, but when DOS is present they are used as follows:

736,737 2E0,2E1 RUNAD

Used by DOS for the file run-address which can either be bytes 2 and 3 of sector-1 with a value of 6 added to it, or it can be the run address read from the binary file last loaded.
A BINARY FILE has the following structure:

```
FF;    the 1st 2 bytes in a Binary-file MUST be 255's
FF;    ($FF's) which indicate that it is loaded from the
       DOS menu option L.
LLSB   Load-address; LSB: DOS = 0 ($00).
LMSB   Load-address; MSB: DOS = 7 ($07).
       Full address = 0+256*7 = 1792.
ELSB   End-address; LSB: DOS = 252 ($FC).
EMSB   End-address; MSB: DOS = 28 ($1C).
       Full address = 252+256*28 = 7420.
```

With the above information, DOS then loads the Machine-language program. The amount of data = "end-address" - "load-address" bytes, and when all the data has been loaded DOS then searches for more load/end addresses. This is where you should put locations 736 and 737; Ld-address = 224 and 2 ($E0 and $02), End-address = 225 and 2 ($E1 and $02). Follow this with 2-bytes which is the Run-address of the machine-language file loaded.
Of course, you should be able to see that the Binary-file can load several bulks of code or data into various parts of memory before actually initiating at a particular address. If you don't place the RUNAD addresses along with the Run-address itself at the end of the file, then control will pass back to DOS when load is complete.
However, should you put the RUNAD addresses at the end of the file, and leave out the 2-bytes indicating the address to jump to, then DOS will return an End-of-File (EOF) error. This also applies to truncated data/code.

738 - 740

Within DOS, you can specify the Start, End, Initiation and Run addresses when you use Binary-save option K by typing: "FILENAME.EXT,Start,End,Init,Run". If you wish to load a Binary-file into memory without running it, then type: "/N" after the filename on the Binary load option L.
The following program will create a file that can be loaded from DOS Binary-load option L:

```
10 DATA 255,255
12 DATA 0,6
14 DATA 16,6
16 DATA 173,11,212,141,10,212
18 DATA 141,24,208,173,31,208,201,6
20 DATA 208,240,96
22 DATA 224,2
24 DATA 225,2
26 DATA 0,6
30 OPEN #1,8,0,"D:COLR.OBJ"
40 FOR I=0 TO 28
50 READ D:? #1;CHR$(D);:NEXT I
60 CLOSE #1
```

Line 10 contains the 255's, line 12 is the Load-address, line 14 is the End-address, lines 16, 18 and 20 are a machine-language program, line 22 is the RUNAD location 736, line 24 is RUNAD location 737 and line 26 is the actual Run-address. You can exit the routine by pressing START.
If a boot-file is appended to another boot-file, then the FF' beginning the file are not deleted.

738,739 2E2,2E3 INITAD

Initialization address read from the disk. An autoboot file must load an address value into either RUNAD or INITAD. The code pointed to by INITAD will be executed as soon as that location is loaded, where the code pointed to by RUNAD will only be executed when the entire load process is complete. To return to DOS after the execution of your program, end your code with an RTS instruction.

740 2E4 RAMSIZ

RAM size, high byte only; this is the number of pages of available RAM, where each page is 256 bytes. The value here is the same as RAMTOP, passed here from TRAMSZ. Space/memory saved by moving RAMSIZ or RAMTOP has the advantage of being above the display memory (DM), initialized to 160 with Basic and 192 without in the 800XL.

741,742 2E5,2E6 MEMTOP

Pointer to the top of free memory, used by both Basic (which calls it HIMEM) and the OS, passed here from TRAMSZ after power-up. This address is the highest free location in RAM for programs and data. This value is updated on power-up, Reset, Graphics calls and when IOCB's are opened. The display list (DL) starts at the next byte above MEMTOP.
The screem handler will only open the S: device if no RAM is needed below this value (i.e; there is enough free RAM below here to accommodate the requested Graphis change). Also note that, if a screen mode change extends the screen mode memory below APPMHI, then the screen is set to Graphics 0, MEMTOP is updated and an error is returned to the user, otherwise all is ok and the mode change will take place.
Space saved by lowering MEMTOP is below the DL. Be careful not to overwrite it if you change Graphics modes in mid-program. Also ensure that you set APPMHI above your data to avoid having the screen data descend into it and destroying it.

743,744 2E7,2E8 MEMLO

Pointer to the bottom of free memory, initialized to 1792 and updated by the presence of DOS to 7420. It is used by the OS; BASICs pointer to the bottom of free memory is LOMEM at 128,129. The value in MEMLO is never altered by the OS after power-up.
This is the address of the 1st free byte of RAM available for program use. Set after all FMS buffers have been allocated (see 1801 and 1802). The address of the last sector buffer is incremented by 128 (the buffer size in bytes) and the value is placed in MEMLO. This value is passed back to LOMEM on the execution of the Basic NEW command.
If you are reserving space for your own device driver/s or buffers, you load your routine into the address specified by MEMLO and add the size of your routine to the MEMLO value, and POKE the new value + 1 back to MEMLO.
You can alter MEMLO to protect an area of memory below your program as an alternative to protecting an area above RAMTOP. This way can be used to avoid the problems of the screen CLEAR function and the text-window scrolling which destroy data above RAMTOP. However, unless you create a MEM.SAV file, the data will be wiped out when DOS is called.
To alter MEMLO, you start by POKEing WARMST at location 8 with 0, then doing a USR to the BASIC entry point at 40960 ($A000) after defining the area to protect, for example:

```
10 MEMLO=BYTES+PEEK(743)+256*PEEK(744)
20 HI=INT(MEMLO/256)
30 LO=MEMLO-HI*256
40 POKE 743,LO:POKE 744,HI
50 POKE 128,LO:POKE 129,HI
60 POKE 8,0
70 X=USR(40960)
```

The program will erase itself when run so be sure to save it first. The amount of memory protected from the TRUE bottom of RAM to the new MEMLO is given by the BYTES variable, just give it a value according to how many bytes you wish to reserve.

746-749 2EA-2ED DVSTAT

Additional device status registers to contain information returned to the computer by the peripheral after the new type-3 and 4 polls. The bytes are as follows:

746,747	LSB/MSB handler size, must be an even number
748	Device SIO address used for loading
749	Peripheral revision number

The new poll types are fully explained in the 1200XL OS manual; earlier polls you'll find explained in the old-Atari Hardware-manual. Basically, type-3 is an "Are you there?" poll (device address $4F, command byte $40, AUX1 $4F, AUX2 $4F, checksum normal), and the type-4 poll is a Null-poll (values $4F, $40, $4E and $4E, respectively; checksum normal).

750,751 2EE,2EF CBAUDL/H

Cassette baud rates low and high bytes, initialized to 1484 which represents a nominal 600 baud (bits per second). After baud rate calculations, these locations will contain POKEY values for the corrected baud rate. The baud rate is also adjusted by SIO to account for motor variations, tape stretch etc.. The beginning of every cassette record contains a pattern of alternating bits (0 and 1, off and on) which are used solely for this baud (speed) correction.

752 2F0 CRSINH

Cursor inhibit flag. 0 turns the cursor on at the next print, and a nonzero value turns it off at the next print. The cursor is restored to its default value 0 upon Reset, Break or an OPEN to the S: or E: devices (which includes Graphics calls). See location 755 for additional cursor and text alterations.

753 2F1 KEYDEL

Key delay flag or key debounce counter; used to see if any key has been pressed. A value of 0 is returned if no key is pressed. A value 3 is returned if a key is pressed. This value is decremented by the stage-2 VBlank until it reaches 0. If a key is pressed while KEYDEL is greater than 0 then it is ignored and considered as "bounce".

754 2F2 CH1

Prior keyboard character code (most recently read and accepted). This is the previous value passed from 764. If the value of the new key code equals the value in CH1, then the code is accepted only when a suitable key-debounce delay has taken place since the prior value was accepted, see DEYDEL.

755 2F3 CHACT

Character mode register, initialized to 2. Shadow for 54273. See the table of bit functions:

BIT:	DEC:	FUNCTION:
0	1	Blank inverse
1	2	Normal characters
0,1	3	Solid inverse characters
2	4	Invert text

This register also controls the transparency of the cursor because the cursor is simply an inverse space character. By toggling bit-0 on and off, you can make the cursor flash, note that this also flashes all inversed characters on the screen. Try the following program:

```
10 GRAPHICS 0
20 FOR I=0 TO 19
30 X=INT(RND(0)*40):Y=INT(RND(0)*24)
40 POSITION X,Y:? CHR$(128+X);
50 NEXT I
60 POKE 755,(PEEK(20)]128)+1:GOTO 60
```

This program is ok to see the affect in action, but it doesn't keep going while you type. So here is a VBlank cursor flashing routine:

756

```
10 FOR I=0 TO 31
20 READ D:POKE 1536+I,D:NEXT I
30 DATA 104,169,7,162,6,160,11,32,92,228,96
40 DATA 165,203,208,12,169,2,77,243,2,141,243,2
50 DATA 169,25,133,203,198,203,76,98,228
60 X=USR(1536)
70 POKE 54286,64
```

This program uses location 203 as the flashing timer variable, so if you use this routine in your own programs, don't use this location. If you want to change the type of inverse flashing, then change the 1st occurence of the value 2 in line 40, no other occurence! You can also change the speed at which the cursor flashes by changing the value 25. This initial value is a half-second delay for each status of the cursor (on and then off). A value of 12 would be a quarter of a second flash rate.

756 2F4 CHBAS

Character base register, shadow for 54281 ($D409). Initialized to 224 which is the address (224*256 = 57344, the standard character set). To obtain the lowercase and graphics characters in Graphics modes 1 and 2 then POKE here with 226. For the international character set POKE here with 204. In Graphics 0, this character set replaces the graphics characters. On the 1200XL, the value here is switched with that in CHSALT at 619 when CTRL+F4 is used to toggle between these sets.
You can create your own character-set and point to it with this location. Each character is made up of 8 numbers, where each number is the Binary-sum of 8 SET Bits which define the shape. Note that these bits are for each row of the character, in Graphics 0 there are 8 rows per character, the letter "A" looks like this:

```
BIT:  76543210     DEC:
      00000000  =    0 ;no Bits set
      00011000  =   24 ;Bits 4,3
      00111100  =   60 ;Bits 5,4,3,2
      01100110  =  102 ;Bits 6,5,2,1
      01100110  =  102 ;Bits 6,5,2,1
      01111110  =  126 ;Bits 6,5,4,3,2,1
      01100110  =  102 ;Bits 6,5,2,1
      00000000  =    0 ;no Bits set
```

The decimal values are derived from the sum of the SET Bits, where Bit-7 = 128, Bit-6 = 64, Bit-5 = 32, Bit-4 = 16, Bit-3 = 8, Bit-2 = 4, Bit-1 = 2 and Bit-0 = 1.

756 cont.

When altering the character-set, it's important to know which characters to alter and which ones not to. The character-set is stored in memory in a particular order, this order is neither Atascii or the RAW-code order, see 764 and 121,122 for these orders. This order of characters is shown below. Remember, that each character requires 8 values for it's design, so you must multiply the given character-code by 8 to arrive at the data for the character that you want to change, for example, to arrive at the data for the letter "A" and prove that the above information is correct you should perform the following task:

```
10 GRAPHICS 0
20 CHAR=PEEK(756)*256+(33*8)
30 FOR I=0 TO 7
40 ? PEEK(CHAR+I)
50 NEXT I
```

You should find these values exactly the same as those listed earlier.
Here's the table of character codes:

CODES:	CHARACTERS:
0	SPACE
1-9	! " # $ % & ' ( )
10-15	* + , - . /
16-25	Numbers 0 - 9
26-32	: ; < = > ? @
33-58	Capital letters A - Z
59-63	[\ ] ^ _
64	CTRL+","
65-90	CTRL+Letter-keys A/a - Z/z
91	ESCape character
92-95	Up, Down, Left, Right arrows
96	CTRL+"."
97-122	Lowercase letters a - z
123-124	CTRL+";" ¦
125	Clear-screen character
126-127	Delete-character Tab-char.

It is possible to alter the character-set where it is in the ROM, but you need to see location 54017. Otherwise, you'll need to transfer it down into RAM, preferably a protected area. In Basic you would setup a FOR/NEXT loop to copy the ROM into RAM, but because this is quite slow, I've dug-up a routine that will transfer the set using machine-code:

756 cont.

```
10 POKE 106,PEEK(106)-4
20 GRAPHICS 0
30 NEUSET=PEEK(106)*256
40 FOR I=0 TO 31
50 READ D:POKE 1536+I,D:NEXT I
60 DATA 104,104,133,204,104,133,203,169,224,133,206
70 DATA 160,0,132,205,162,4,177,205
80 DATA 145,203,136,208,249,230,204
90 DATA 230,206,202,208,242,96
94 X=USR(1536,NEUSET)
96 POKE 106,NEUSET/256
```

The program protects 4-pages of memory above RAMTOP, and transfers the standard ROM character-set into this area. You don't really need to transfer the old-set down, especially if your going to change the complete character-set, but with this method, the characters that you don't change will at least show up as what they're supposed to be instead of blank spaces. If you'd rather copy the international-set down instead of the standard one, then you can change the value 224 in line 60 to 204.

A simple routine to add to the last program to alter the characters is:

```
100 READ CH:IF CH=-1 THEN STOP
110 FOR I=0 TO 7
120 READ BITSUM
130 POKE NEUSET+(CH*8)+I,BITSUM
140 NEXT I
150 GOTO 100
160 DATA 0,129,66,36,8,16,36,66,129
999 DATA -1
```

The DATA must end with a value of -1 to indicate no more characters are to be altered. Also note, if you wish to use this yourself, then the 1st number on the DATA-line is the code of the character to change, while the remaining 8 numbers is the actual data for the new character shape.
If your changing the whole character-set, then you can always erase line 100 and setup a nested-loop, for example:

```
100 FOR CH=0 TO 127
150 NEXT CH
```

This way, the DATA needn't have the character-code on, but you must define all the characters in the correct order shown in the table on the previous page. When you design your characters, you might find it easier to use graph paper.

Note also, that the above program reserves 4-pages for your character-set. This is because there are 128 different characters as distinguished in the table on page 78. Each character takes 8 bytes (Binary-sums) to define, thus, 8*128 = 1024 bytes of memory. Each page is 256 bytes, so 1024/256 = 4 Pages.
Note that when you press Reset or issue a Graphics call, location 756 is re-initiated to point to the standard character-set in ROM, so just re-POKE 756 with the Page number that your character-set is on to re-enable it.

Listed below is a routine that will save your altered character-set as a 9-sector file, just run your program that defines your character-set and then type-in and RUN the program here:

```
0 DATA 104,104,104,170,76,86,228
1 FOR I=0 TO 6
2 READ D:POKE 1536+I,D:NEXT I
3 OPEN #1,8,0,"D:NAME.FNT"
4 POKE 849,1:POKE 850,11:POKE 852,0:POKE 853,PEEK(106)
5 POKE 856,0:POKE 857,4:POKE 858,8
6 X=USR(1536)
7 CLOSE #1
```

On the other hand, if you wish to use this routine in your own programs to load your 9-sector character-set file, then use the following program:

```
10 POKE 106,PEEK(106)-4
20 GRAPHICS 0
30 FOR I=0 TO 6
40 READ D:POKE 1536+I,D:NEXT I
50 DATA 104,104,104,170,76,86,228
60 OPEN #1,4,0,"D:NAME.FNT"
70 POKE 849,1:POKE 850,7:POKE 852,0:POKE 853,PEEK(106)
80 POKE 856,0:POKE 857,4:POKE 858,4
90 X=USR(1536)
94 CLOSE #1
96 POKE 756,PEEK(106)
```

757 2F5 NEWROW

Point/Row to which DRAWTO and Fill (XIO 18) will go.

758,759 2F6,2F7 NEWCOL

Point/column to which DRAWTO and Fill (XIO 18) will go. NEWROW and NEWCOL are initialized to the values in ROWCRS and COLCRS, which represent the destination end point of the Draw/Fill command used. This is done so that ROWCRS and COLCRS can be altered during the operation being performed.

760 - 764

760 2F8 ROWINC

This is the Row increment or decrement value.

761 2F9 COLINC

The column increment/decrement value. ROWINC and COLINC are used for the line direction. The values represent the signs derived from the value NEWROW minus ROWCRS, and the value NEWCOL minus COLCRS.

762 2FA CHAR

Internal code value for the most recent character read or written (internal code for ATACHR). This register is difficult to use with PEEK statements since it returns the most recent character which is most often the cursor value 128 when visible, and 0 when invisible.

763 2FB ATACHR

Returns the last Atascii character read or written, or the value of a graphics point. ATACHR is used in converting the Atascii code to its internal character code passed to or from CIO. The Fill and DRAWTO commands use this location for the colour of the line drawn, ATACHR being temporarily loaded with the value in location 765. To force a colour change in the line, POKE the desired COLOR number here. It'll then be taken from location 200. Since Basic performs this process, this process won't happen within a machine-language routine.

764 2F9 CH

Internal Hardware value for the last key pressed. The default value here is 255, which also means that no key has been pressed. The keyboard handler gets all of its information from CH, processes all the SHIFT and CTRL codes for the key and passes the keycode value to location 754. If the value in CH is the same as that in CH1, then the key will only be accepted if a suitable key-debounce time-delay has transpired. If the keypress is a CTRL+"1" then the start/stop flag at location 767 is complemented, but the value is not stored in CH.
This is neither the Atascii or the internal code value; it is the RAW keyboard matrix code for the key pressed. The translation table is in KEYDEF at 121 and 122. Try the following program:

```
10 POKE 764,255
20 V=PEEK(764)
30 IF V-254 THEN 20
40 ? CHR$(V);:GOTO 10
```

RUN the program and type some characters; you'll notice that the keyboard is very mixed up. There is a simple way to overcome this problem with the aid of KEYDEF at locations 121 and 122. As an example, try the next program:

```
10 KEYDEF=PEEK(121)+256*PEEK(122)
20 POKE 764,255
30 IF PEEK(764)=255 THEN 30
40 CH=PEEK(KEYDEF+PEEK(764))
50 ? CHR$(CH);
60 GOTO 20
```

Due to the use of the Key-definition table, you can now have an easy access to the RAW characters.

765 2FD FILDAT

Colour register number for the XIO Fill command.

766 2FE DSPFLG

Display flag; used in displaying the control codes not associated with the ESC character, see location 674. If 0 is POKEd here, then pressing the keys of the Atascii codes 27 - 31, 123 - 127, 187 - 191 and 251 - 255 perform their normal screen control functions (ie. clear-screen, delete/insert line, cursor move etc.), however, if any nonzero value is POKEd here, then the actual character itself is displayed (alike pressing ESC first). Try POKEing here with a nonzero number and then pressing CTRL and the arrow keys.

767 2FF SSFLAG

Start/Stop display screen flag, used to stop the scrolling of the screen during a Draw command or Graphics routine, a LISTing or PRINTing, or when INPUT is awaited and a key is pressed. When the value here is 0, then the screen output is not stopped. When the value here is 255 (the "Ones complement"), then the screen output is stopped, or rather paused until the flag is cleared, either by toggling it on/off with the CTRL+"1" keypress, clearing this location with a POKE, or pressing the Break-key. If you wish to prevent this flag being set in any case, then you can expand to the stage-2 VBlank. See locations 546 - 549.

768 - 770

Locations 768 - 828 are used for the device handler and vectors to the handler routines (devices S:, P:, E:, D:, C:, R: and K:). A device handler is a routine used by the OS to control the transfer of data in that particular device for the task allotted (such like Read, Save etc.). The resident D: does not conform entirely with the other handler - SIO calling routines. Instead, you have to use the Device-Control Block (DCB) to communicate directly with the disk handler. The device handler for the R: is loaded in from the 850 interface module. See De Re Atari, the 850 interface manual and the OS listing pages 64 and 65.
Locations 768 - 779 ($300 - $30B) are the resident DCB addresses, used by SIO (I/O operations that require the serial-bus). DUP.SYS also uses this block to interface the FMS with the disk handler. The Old Atari disk-drive uses a serial access rate of 19200 baud (bits per second). It has its own microprocessor, a 6507, plus 128 bytes of RAM, a 2316 2K masked ROM chip (like a 2716), a 2332 RAM I/O timer-chip with another 128 bytes of RAM (like the PIA chip) and a WD1771FD controller chip. See the 1050 SPECS appendix concerning this drive. With the US-doubler fitted, you get true-double density which gives 720 sectors, but each sector is 256 bytes instead of 128. Another improvement is its speed, which is 4-5 times faster. If you have the IS-Plate, however, then the transfer rate is fastest of all, being 118000 baud. Some of this information was from Moje Atari Magazine, Poland.

All of the parameters passed to SIO are contained in the DCB. SIO uses the DCB information and returns a status back to the DCB's 4th byte at location 771.

768 300 DDEVIC

Device serial bus ID (serial device type) set by the handler, not user alterable. Vaues are:

Disk-drives	D1 - D4	49 - 52
Printer	P1,P2	64,79
RS232 Ports	R1 - R4	80 - 83

769 301 DUNIT

Device number currently being used.

770 302 DCOMND

The Command-code for the device operation to be performed, set by the user or by the device-handler prior to calling SIO. The Serial-bus commands are:

771 - 775

OPERATION:	DEC:	HEX:
Read	82	52
Write (verify)	87	57
Status	83	53
Put (no verify)	80	50
Format single	33	21
Format dual	34	22
Download	32	20
Read-address	84	54
Read-spin	81	51
Motor On	85	55
Verify sector	86	56

Here's the US doubler codes. You'll have to work these out yourself!

DEC:	HEX:	OPERATION:
63	3F	
72	48	
78	4E	
79	4F	
102	66	
128	80	
129	81	

Note, that Dual-density format is the new density offered by the 1050 disk-drive. The single-density only offers 720 sectors, each comprising of 128 bytes, dual offers 1040 sectors also 128 bytes each sector.

771 303 DSTATS

The status code upon return from SIO to the user. A value of 1 means good status. This is also used to set the data-direction; whether the device is to send or receive a data-frame. This byte is used by the device handler to indicate to SIO what to do after the command-frame is sent and acknowledged. Prior to the SIO call, the handler examines Bits 6 and 7. If Bit-6 (Dec 64) is SET, then receive data. If Bit-7 is SET, then send data. If both Bits are clear, then no data transfer is associated with the operation. Both Bits being SET is invalid.

772,773 304,305 DBUFLO/HI

Data-buffer address for the source or destination of the data to be transferred. Setup by the user, this need not be set if the operation doesn't require data transfer, as in a Status operation.

774 306 DTIMLO

The Time-out value of the handler. The cassette Time-out value is 35, which is 37 seconds. The Timer-values are 64-seconds per 60-units. Initialized to 31.

775 307 DUNUSE

Unused byte; free for use.

776,777 308,309 DBYTELO,HI

The number of bytes transferred to or from the data-buffer from the last operation, set by the handler. Also used for the count of Bad-sector data.

778,779 30A,30B DAUX1/2

Used for device specific information such as the disk sector number in read and write operations. Loaded down to 572 and 573 by SIO.
There are only 5 commands supported by the disk-handler; Read, write, put, status and format (see DCOMND). There is no way to format particular sectors of a disk, only the whole disk which in the old 810 drive was done with the non-user accessible INS1771-1 formatter/controller chip. Apparently, there was an "E" chip-revision which allowed for selective formatting but what happened to it I don't know. The Archiver chip certainly allows this (is this the "E" chip??). Try this:

```
10 SCTS=10:BUF=PEEK(106)-SCTS/2
14 BUF=BUF-(NOT BUF=INT(BUF))
18 POKE 106,BUF
22 GRAPHICS 0
26 FOR I=0 TO 34
30 READ D:POKE 1536+I,D:NEXT I
34 DATA 104,32,83,228,48,251,24,173,4,3,105,128
36 DATA 141,4,3,144,3,238,5,3,24,238,10,3,144,3
38 DATA 238,11,3,206,7,3,208,223,96
42 SSEC=1
46 SHI=INT(SSEC/256):SLO=SSEC-SHI*256
50 POKE 769,1:POKE 770,82
54 POKE 772,0:POKE 773,BUF
58 POKE 778,SLO:POKE 779,SHI
62 POKE 775,SCTS:X=USR(1536)
```

The program above will load the 1st 10 sectors of a given disk into a protected area of memory above RAMTOP. You can use it for your own routine to load various information for a game off the disk. Just set the SCTS variable for the amount of sectors you wish to load, and set SSEC for the starting sector on the disk. If on the other hand, you'd like to use the routine for saving to sectors, then you need to change the POKE 770,82 on line 50, to POKE 770,80. This way, the example shown would save 10 sectors of protected memory. In this case, there is nothing there, so you would have to put some data into this protected area first. Be sure that you do not use a disk that has not been formatted.

780 - 785

If you try reading or writing to a disk that hasn't been formatted, then the drive will need to be turned off and on. You can format a disk with the following routine, but be sure that the disk you place in the drive has no information on it that you may want, because once formatted, it is completely wiped clean. This is not the same as a DOS format, a DOS format also writes several sectors on the disk, where as this format doesn't:

```
10 FOR I=0 TO 4
20 READ D:POKE 1536+I,D:NEXT I
30 DATA 104,32,83,228,96
40 POKE 769,1:POKE 770,33
50 X=USR(1536)
```

The machine-code on line 30 is just 3 instructions: PLA, JSR $E453 and RTS.

Normal formatted sectors have 128 bytes free for use, but a DOS sector gives you only 125 bytes. This is because the last 3 bytes of every sector is used by DOS. See the BOOT appendix for further information on this.

There are loads of public domain programs that use SIO to edit disk sectors, copy them, even repair them. There was also an SIO tutorial in some back issues of Page-6 magazine.

780,781 30C,30D TIMER1

Initial baud-rate timer value.

782 30E ADDCOR

Addition correction flag for the baud-rate calculations involving the timer registers.

783 30F CASFLG

Cassette mode when set. Used by SIO to control the program flow through shared code. A value of 0 means standard SIO operations. When nonzero, it is a cassette operation.

784,785 310,311 TIMER2

Final timer value. TIMER1 and TIMER2 contain reference times for the start and end of the fixed bit pattern receive period.

786 - 793

The 1st byte of each timer contains the VCOUNT value from location 54283 ($D40B), while the 2nd byte contains the current realtime clock value from location 20. The difference between the timer values is used in a lookup table to compute the interval for the new values for the baud-rate passed on to locations 750 and 751.

786,787 312,313 TEMP1

2-byte temporary storage register used by SIO for the VCOUNT calculation during baud timer routines.

788 314 TEMP2

Temporary storage register.

789 315 TEMP3

Ditto.

790 316 SAVIO

Save serial data-in port used to detect, and updated after, each bit arrival. Used to retain the state of Bit-4 of 53775 ($D20F; serial data-in register).

791 317 TIMFLG

Time-out flag for baud-rate correction, used to define an unsuccessful baud-rate value. Initially set to 1, it is decremented during the I/O operation. If it reaches 0 (after 2 seconds)) before the 1st byte of the cassette record is read, the operation is aborted.

792 318 STACKP

SIO stack pointer register. It points to the byte in the stack being used by the current operation. The stack takes up Page-1 of memory, locations 256 - 511 ($100 - $1FF).

793 319 TSTAT

Temporary status holder for location 48 ($30).

Locations 794 - 828 are the Handler-address tables. There are only 5 handlers normally present in the Atari. They are the Printer (P:), the Cassette (C:), the Display-Editor (E:), the Screen-handler (S:) and the Keyboard (K:).
When DOS is loaded, the D: handler is installed, and the R: handler is installed with the 850 interface connected.

794-828 31A-33C HATABS

Handler address table. 35 bytes are reserved here for up to 11 entries of 3 bytes per handler. The last 2 bytes are 0'd (nulled). On power-up, the HATABS table is copied from ROM. Devices to be booted, such as the disk-drive, add their handler information to the end of the table. Each entry has the character device name (C, D, E, K, P, S, R) in Atascii code and the handler address (LSB/MSB). Unused bytes are all set to 0. FMS searches HATABS from the top for a device "D:" entry, and when it doesn't find it, it then sets the device vector at the end of the table to point to the FMS vector at 1995. CIO searches for a handler character from the bottom up. This allows new handlers to have precedence over the resident ones. Pressing Reset clears HATABS of all but the resident handler entries. The handler entry points are:

LOCATION:

DEC:	HEX:	HANDLER:	VECT.to:
794	31A	Printer (P:)	58416
797	31D	Cassette (C:)	58432
800	320	Disp.editor (E:)	58368
803	323	Screen Disp.(S:)	58384
806	326	Keyboard (K:)	58400
809	329	unused	
812	32C	unused	
815	23F	unused	
818	332	unused	
821	335	unused	
824	338	unused	
827,8	33B,C	nulled 2-bytes; always 0'd	

If you wish to create your own handler, then you should put the Atascii code of the Device name into the handlers 1st byte and the address of your handler routine into the handlers 2nd and 3rd bytes. Example; POKE 809,ASC("X") where "X" is the device-character. POKE 810,0 and POKE 811,6 would have the X-device handler pointing to Page-6 of memory (0+256*6 = 1536 ;$0600). At this address, you must place a table of vectors; the vectors are as follows:

828 cont.

OPEN vector
CLOSE vector
GET vector
PUT vector
STATUS vector
XIO vector
JMP INIT vector

The 1st 6 vectors are 2 bytes each, which point to the address of the associated routine minus 1. The JMP INIT vector points directly to the routine that is executed upon initialization of the handler only, which can just be an RTS command.
It doesn't matter what IOCB channel is used, because whatever operation of your device handler is executed, all the associated bytes used in the command are loaded down to the Page-0 IOCB, ZIOCB at locations 32 - 47. Upon EXIT of your device operations, you should load the Y register with a value of 1 for good status, or the error code if not good status. Also, end all your routines with an RTS command.
The best explanation of handlers I've seen is in the Old Atari user magazine, Volume 3 number 2. Also see De Re Atari and the OS source-listing manual.
Try the following program:

```
10 DATA 14,6,14,6,16,6,14,6,14,6,16,6
20 DATA 76,14,6,160,1,96
30 FOR I=0 TO 17
40 READ D:POKE 1536+I,D:NEXT I
50 Y=794
60 IF PEEK(Y) THEN Y=Y+3:GOTO 60
70 POKE Y,ASC("N")
80 POKE Y+1,0:POKE Y+2,6
```

This is a 'Null' handler, it does absolutely nothing! It's useful for De-bugging routines, just set it up and replace all device calls in a program that you want debugged with the "N" device-character. If you wish to include your own routines, then just replace the addresses with the addresses of your own routines - 1. My routine simply performs: LDY #1 and RTS. The GET and XIO operations are directed to the RTS, skipping the LDY #1 instruction. This shows that they have been chosen as unused. You can change their addresses from 16,6 in the data line to 14,6.
You can change the addresses of the routines to your own if you like, try changing the two 16's in line-10 to 14's, change line-20 to read:
20 DATA 76,14,6,165,20,141,200,2,160,1,96
and change the FOR/NEXT loop in line-30 to go from 0 to 22. Press Reset and re-run the program. Type OPEN #1,4,0,"N:" and GET #1,B. You'll notice the new border colours are in the variable B.

829 - 831

Try the LIST "N:" command. You'll notice 2 border changes, this is because LIST uses 2 vectors; GET and PUT. You can use the XIO command vector to perform different tasks depending on the value if you like, ie; if you type XIO 3,#1,0,0,"N:". The command-value 3 will be passed to the ZIOCB location ICCOMZ at 34. Just read this location in your XIO vector routine and perform X-task for X-number. You can achieve many tasks by writing new handlers, perhaps even altering the existing handlers. One such task is creating new Basic commands. See back issues of Page-6 magazine.

829-831 33D-33F PUPBT1-3

Power-up and Reset validation registers 1 - 3, used on warmstart to verify the integrity of memory. The OS initializes these locations to 92, 147 and 37. When Reset is pressed, these bytes are checked and if they're the same as initialized a warmstart is done, otherwise a coldstart occurs.

Locations 832 - 959 are reserved for the 8 Input Ouput Control Blocks (IOCB's). IOCB's are channels for the transfer of information into and out of the Atari, and even from 1 area of memory to another.
Every time you use commands, such like, PRINT, SAVE, LOAD, LIST etc. you are using an IOCB. Some of the IOCB's are dedicated for special purposes, such as IOCB-0 which is used for the screen display. When you use the OPEN command, the parameters following it tell CIO which direction the data is to be transmitted. It is SIO and the device handlers that do the actual transfer of data.
You don't have to use Basic commands to access CIO, for example the OPEN #1,4,0,"D:" command can be implemented with several POKEs and a JSR to the CIO entry point at 58454 ($E456). It's useful to use CIO directly sometimes, because Basics INPUT command can only access 120 bytes at a time, where a single call to CIO can fill the whole RAM from the input device or vice versa. This transfer of bytes, ofcourse, is also at machine-language level which is much faster.
These blocks are used the same way as ZIOCB. The OS takes the information here, and moves it to the ZIOCB for use by CIO, it also returns the updated information back to the user area when the operation is complete.
Note that when Basic encounters the DOS command, all channels are closed except for channel-0 (IOCB-0).

832 - 847

832-847 340-34F IOCB0

IOCB-0. Normally used for the screen editor (E:). You can send all the screen output to the printer with POKE 838,202 and POKE 839,254. To send everything back to the screen, POKE 838,175 and POKE 839,242. You can use the program on the next page to toggle output back and fore with the SHIFT+HELP and CTRL+HELP keys. Note, that the program is written in the VBlank, so if you LIST a long program, pressing the console keys will react exactly on the bytes presently being displayed/printed etc.

```
10 FOR I=0 TO 47
20 READ D:POKE 1536+I,D:NEXT I
30 DATA 104,169,7,160,11,162,6,32,92,228,96
40 DATA 173,220,2,201,145,208,13
50 DATA 169,202,141,70,3,169,254,141,71,3,76,98,228
60 DATA 201,81,208,10
70 DATA 169,175,141,70,3,169,242,141,71,3,76,98,228
80 X=USR(1536)
90 POKE 54286,64
```

Another very useful application for these locations, is something called "Return-key mode". Try POKE 842,13. You'll notice the cursor is shuffling off down the screen! It's actually pressing the Return-key over anything it might come across on the screen. This is a very useful technique for adding or deleting lines to a program, from within the program! For example;

```
10 GRAPHICS 0
20 POSITION 2,5
30 FOR I=0 TO 6
40 ? 100+I;" REM ADDED LINE ";I
50 NEXT I
60 ? "CONT"
70 POSITION 2,0
80 POKE 842,13:STOP
90 POKE 842,12
96 GOTO 96
```

You'll notice that the program itself actually adds lines 100 - 106. The Return-key mode is activated on line-80 and the program is STOPped. The program is CONTinued when the cursor runs over the CONT command printed to the screen from line-60. Line-90 then turns the Return-key mode off and holds program execution at line-96.
If you LIST a file to cassette or disk, you can edit the file with a word processor and insert Basic commands in direct-mode (without line-numbers). When the file is loaded, the direct lines are executed straight away. This is very useful for protecting your Basic programs.

848 - 959

You can even automatically RUN your Basic programs from disk or cassette with the use of the Return-key mode and direct-mode command entry. This applies to both types of saved files: LISTed and SAVEd.

When you are in a Graphics mode other than 0, channel-0 is opened by the OS for the text-window. If the text-window is turned off and you OPEN channel-0, Graphics 0 will be called. The NEW and RUN commands close all channels except channel-0.

848-863 350-35F IOCB1

IOCB-1; unused.

864-879 360-36F IOCB2

IOCB-2; unused.

880-895 370-37F IOCB3

IOCB-3; unused.

896-911 380-38F IOCB4

IOCB-4; unused.

912-927 390-39F IOCB5

IOCB-5; unused.

928-943 3A0-3AF IOCB6

IOCB-6; The Graphics statement OPENs channel-6 for the screen display (S:), so once you are out of Graphics 0, you cannot use channel-6 unless you firstly issue a CLOSE #6 statement. If you do close this channel, however, you will not be able to use DRAWTO, PLOT or LOCATE until you reOPEN it. The LOAD command closes all channels, even #6, except for #0.

944-959 3B0-3BF IOCB7

IOCB-7; LPRINT automatically uses this channel. If the channel is already open when an LPRINT is issued, then an error will occur.

The LIST command also uses this channel and closes it after use. LOAD uses this channel to transfer programs between cassette and disk. LIST (except to the screen), LOAD and LPRINT also close the sound-voices. RUN and SAVE also use this channel.

Each byte in the IOCB's all have a particular meaning, explained in the chart below:

LABEL OFFSET BYTES

ICHID 0 1
Index into the device-name table for the currently open file. Set by the OS. If not in use, the value = 255 which is also initiation value.
ICDNO 1 1
Device number; 1 for D1:, 2 for D2: etc.. Set by the OS.
ICCOM 2 1
Device Command, set by the user. This is the 1st variable after the channel-number in an OPEN command. See the COMMAND chart overleaf for a full summary of these codes.
ICSTA 3 1
Device status, returned by the OS. See the chart overleaf.
ICBAL/H 4,5 2
Buffer address for data transfer, also the address of the filename for the OPEN and STATUS commands etc..
ICPTL/H 6,7 2
Address of the devices "put-one-byte" routine -1. Set by the OS on OPEN, but only used by Basic. Points to CIO's "IOCB NOT OPEN" message at power-up.
ICBLL/H 8,9 2
Buffer length, set to the amount of bytes to transfer in PUT and GET operations. Decremented by 1 for each byte transferred; updated after each READ and WRITE operation. Records the number of bytes actually transferred in and out of the buffer after each operation.
ICAX1 10 1
Auxiliary byte 1 (AUX1). Used with the OPEN statement to specify the type of file access. See the table on Page-96 for a full list of codes you can use with what devices.
ICAX2 11 1
Aux byte 2. Special use by each device driver; some serial port functions may use this byte.
ICAX3/4 12,13 2
Aux bytes 3 and 4, used to keep a record of the disk sector number with NOTE and POINT.
ICAX5 14 1
Pointer to the byte within a sector NOTEd/POINTed to. Value ranges 0 - 124. The last 3 bytes have special use by DOS, see location 1792.
ICAX6 15 1
Spare Aux byte.

Here's the SIO Status byte values, below them, the ICCOM command byte values:

STATUS:	EXPLANATION:
1	Operation complete; Status OK.
138	Device timeout (no response).
139	Device NAK
140	Serial-bus input framing error.
142	Serial-bus data-frame over-run error.
143	Serial-bus data-frame checksum error.
144	Device done error.

COMMAND:	DEC:	HEX:
OPEN channel	3	3
GET text-record line (INPUT)	5	5
GET Binary record (buffer)	7	7
PUT text-record line	9	9
PUT Binary record (buffer)	11	B
CLOSE channel	12	C
Dynamic (channel) status	13	D

Basic uses an IOCB "put-byte" vector for the PRINT #n,A$ command.
Disk-File Management (FMS) commands (Basic XIO commands) use:

Rename	32	20
Erase	33	21
Protect/lock	35	23
Unprotect/unlock	36	24
POINT	37	25
NOTE	38	26
Format single	253	FD
Format double	254	FE

In addition, XIO also supports:

GET character	7	7
PUT character	11	B
Draw line	17	11
Fill area; XIO 18,#1,0,0,"S:"	18	12

Fill requires the PLOT and POSITION commands, also its colour at location 765.

For the RS232 (R:), XIO supports:

Output partial block	32	20
Control RTS, XMT, DTR	34	22
BAUD, Stop-Bits, Word-size	36	24
Translation mode	38	26
Concurrent mode	40	28

CIO treats any command byte value greater than 13 ($0D) as a special case (XIO case), and transfers control over to the device handler for processing.

Here's a list of the ICAX1 bytes, associated also with the 1st parameter given in the OPEN statement:

DEVICE:	TASK#	DESCRIPTION:		
Cassette	4	Read		
	8	Write		
Disk	4	Read		
	6	Directory (S/dens)		
	7	Directory (D/dens). This shows up all files that use the sharp brackets.		
	8	Write new file. Any file OPENed in this mode will be deleted, and the 1st byte next written is at the start of the file.		
	9	Write - Append. In this mode, the file is left intact, but all data written to the file will start at the end of the existing data		
	12	Read and Write - update. Bytes read or written will start at the beginning of the file		
Screen-Editor E:	8	Screen output		
	12	Keyboard input and screen ouput		
	13	Screen input and output. Which is also known as "Return-key mode"		
Keyboard	4	Read		
Printer	8	Write		
RS232	5	Concurrent read		
	8	Block write		
	9	Concurrent write		
	13	Concurrent read and write		
		Clear Screen on GR.	Text Window also	Read operation
Screen-Display S:	8	yes	no	no
	12	yes	no	yes
	24	yes	yes	no
	28	yes	yes	yes
	40	no	no	no
	44	no	no	yes
	56	no	yes	no
	60	no	yes	yes

Note, that with S:, the screen is always cleared in Graphics 0 and there is no text-window unless you specifically POKE it there, by POKE 703,4.

Without the screen clear, the previous material will remain on-screen between Graphics mode changes, but will not necessarily be legible in other modes, or even within display memory view. The values with S: are placed in the 1st auxiliary byte of the IOCB. Also, all of the screen values back overleaf are a write operation.
The 2nd parameter in an OPEN statement (placed in the AUX2 byte) is far more restricted in its use. Usually set to 0. If set to 128 for the cassette, it changes from normal to short Inter-Record-Gaps (IRG). With the Old Atari 820 printer, a value of 83 means sideways characters. Other printer variabes are: 70 for normal 40-column printing, and 87 for wide printing mode. You can also call a Graphics mode with OPEN and other relevant codes, for example:

```
0100            *=$600
0110 ;
0120 CIO     =    58454
0130 COMMAND =    834
0140 BUFFER  =    836
0150 AUX     =    842
0160 AUX2    =    843
0170 ;
0180           LDX #32
0190           LDA #3            ;OPEN
0200           STA COMMAND,X
0210           LDA #24           ;CLRSCRN
0220           STA AUX,X
0230           LDA #8            ;MODE
0240           STA AUX2,X
0250           LDA #NAME&255
0260           STA BUFFER,X
0270           LDA #NAME/256
0280           STA BUFFER+1,X
0290           JSR CIO
0300           BRK
0305 ;
0310 NAME      .BYTE "S:"
```

You can select the Graphics mode by changing the number loaded into the Accumulator on line-230. Also set the mode to clear or not, have a text-window etc. with the value in line-210, which is taken from the table pre-leafed.

If you want to know how to draw a line in machine-language, then you can add the routine overleaf to the program above:

960 - 1000

```
0300          JMP DRAW
0320 ;
0330 DRAW     LDA #17          ;DRAW
0340          STA COMMAND,X
0350          LDA #8           ;WRITE
0360          STA AUX,X
0370 ;
0380          LDA #1           ;COLOUR
0390          STA 763
0400          LDA #50          ;PLOT
0410          STA 84           ;ROW
0415          LDA #90          ;
0420          STA 85           ;COLUMN
0423          LDA #0           ;
0426          STA 86           ;LO/HI
0430 ;
0440          LDA #90          ;DRAWTO
0450          STA 96           ;ROW
0460          LDA #50          ;
0470          STA 97           ;COLUMN
0480          LDA #0           ;
0490          STA 98           ;LO/HI
0500 ;
0510          JSR CIO
0520 ;
0530          LDA #12          ;CLOSE
0540          STA COMMAND,X    ;CHANNEL
0550          JSR CIO
0560 ;
0570          BRK
```

The whole program is fairly straightforward and comes into 3 parts. The 1st part is the Graphics call, the 2nd part is the actual draw-line routine and the final part is to CLOSE the OPEN channel. As you can see, it is quite large compared to its equivalent in Basic!
For more information on IOCB's, you could get hold of De Re Atari, the OS users manual or take a look at back issues of Old Atari user and Page-6 magazine.

960-999   3C0-3E7   PRNBUF

Printer buffer. The printer handler collects output from the LPRINT statement here, sending it all to the printer when an EOL occurs, or when the buffer is full. The old bug is now gone.

1000   3E8   SUPERF

Screen editor register; cleared on entry to the "put-byte" routine, the editor changes key-codes 142 - 145 to codes 28 - 31 and sets SUPERF to nonzero. See locations 121,122.

1001 - 1017

1001 3E9 CKEY

Cassette boot request flag on coldstart. Checks to see if the START key is pressed, and if so, CKEY is set. Autoboot cassettes are loaded by pressing the START key, pressing Play on the tape and pressing return. You can disable Basic by holding OPTION along with START.

1002 3EA CASSBT

Cassette boot flag. The Atari attempts both cassette and disk boots simultaneously. 0 here means no cassette-boot was successful. See location 9.

1003 3EB CARTCK

Cartridge checksum. A checksum of page-1 of the cartridge. The checksum is recalculated each VBlank and checked against the register. If not the same, the OS assumes the cartridge isn't there any more (was pulled out) and does a coldstart; 1200XL only. Unused in other XL/XE's.

1004 3EC DERRF

Screen OPEN error flag; if 0, then there is no error. If nonzero, then the OS can't initialize the screen editor.

1005-1015 3ED-3F7 ACMVAR

Reserved for OS variables; on power-up, all variables between 1005 - 1023 are 0'd, but unchanged on warmstart.

1016 3F8 BASICF

Shadow for the current status of Basic. 0 means that the ROM Basic is enabled, while nonzero means it is disabled. Must be in sync with disabling of ROM Basic. To disable Basic, set BASICF to nonzero and press Reset. DOS will tell you there is no Basic when you try to return to it.

1017 3F9 MINTLK

Although labelled, mapping states this to be unused.

1018 3FA GINTLK

Cartridge interlock register; the complement of BASICF. It reads 1 when an external cartridge is installed, and 0 when not (or Basic is in use). The value of TRIG3 at 53267 ($D103) is loaded here by the OS initialization routine. If at any time, the external cartridge is pulled, the system crashes.

1019,1020 3FB,3FC CHLINK

Relocatable handler chain use; allows chaining of portions of handler routines.

1021-1151 3FD,47F CASBUF

Cassette buffer. These locations are used by the cassette handler to read data from and write data to the tape recorder. The 128 data bytes for each cassette record are stored here beginning at 1024 ($400 - Page-4). The current buffer size is found at BLIM in 650. Location 61 points to the current byte being read or written.
CASBUF is also used to store the 1st sector in a disk-boot (beginning at 1024) before being transferred to its correct address, given by bytes 3 and 4. See the BOOT appendice.

A cassette record consists of 132 bytes: 2 control bytes set to 85 ($55; alternating 0's and 1's) for speed measurement in the baud-rate correction routine; 1 control byte which is explained on the next leaf; 128 data-bytes, and a checksum byte. Note, that only the data-bytes are stored in this cassette buffer.

CONTROL-BYTE Values:

VALUE:	EXPLANATION:
250 ($FA)	Partial record; the actual number of bytes is stored in the last byte (127) of the record,
252	Record full; 128 bytes follow,
254	End-Of-File (EOF) record; followed by 128 zero bytes.

1152-1405 480-57D STACK

Basic uses these 254-bytes as a syntax checking stack; $480 is a Basic input index; $481 an output index and $482 is a program counter.

1406 - 1791

If you are not using Basic, then you have these 253 bytes free for use. If you don't use the FP package, then you also have a further 129 bytes from 1406 - 1535. Should you use the FP package, then it is as follows:

1406 57E LBPR1

LBUFF prefix 1;

1407 57F LBPR2

LBUFF prefix 2;

1408-1535 580-5FF LBUFF

Basic line-buffer; 128 bytes. Used as an output result buffer for the FP to ASCII conversion routine. The input buffer is pointed to by locations 243 and 244.

1504 5E0 PLYARG

Polynomial arguments (FP use).

1510-1515 5E6-5EB FPSCR

FP scratch-pad use.

1516-1535 5EC-5FF FPSCR1

Ditto. The end of the buffer is named LBFEND.

1536-1791 600-6FF PAGE6

Page-6 is a very useful 256 bytes of free memory, specifically protected so that programmers can use this area safely in Basic/Assembly or machine-language. Besides being used to name a very good magazine, which now goes under the name 'New Atari-user', Page-6 can be used to store quick machine-language subroutines for use by Basic programs. You'll notice that all my programs in this book use this page in this way.

1792 - 7548 +

There is, however, one snag. If you use the Basic INPUT statement when inputting data, then you should ensure that the data you are INPUTing has an EOL flag (RETURN character - ATASCII 155) at a maximum of 128 bytes apart. If an EOL flag doesn't exist, then Basic will continue loading the data, past the Basic line buffer LBUFF at 1408 and on into Page-6, overwriting any 'thought-of' protected data presently residing there. It will keep going until either it reaches the 126th byte into page-6 (location 1662) where it places an EOL character, or until it reaches an End-Of-File (EOF) character (ATASCII 136, or CTRL+3).

Free RAM begins in all XL/XE's at location 1792, pointed to by MEMLO at 743 and 744. When DOS 2.0 is loaded, MEMLO is updated to point to location 7420. For DOS 2.5 see the relevent appendix. DOS is organized in the following manner:

1792-5377 700-1501

FMS provides the interface between Basic, DUP and the Disk-Drive. It is a sophisticated device driver for all I/O operations involving the D: device-name. It allows disk users to use the Basic' special XIO disk commands (see the IOCB area at 832 - 959). It also resides below Basic RAM and provides entry to DUP when called with the DOS command.

5440-13062 1540-3306

DUP.SYS area. The top will vary with the amount of buffer storage space allocated to the drive and sector buffers.

6780-7547 1A7C-1D7B

Drive buffers and sector-data buffers. The amount of memory will vary according to the amount of buffers allocated etc..

7548-MEMLO 1D7C-3306 (maximum)

Non-resident portion of DUP.SYS, DOS utility routines. DUP provides the utilities chosen from the DOS menu page, not from Basic. It is not resident in RAM when you are using Basic or another cartridge, rather it is loaded when DOS is called from Basic or on an autoboot powerup with the option-key depressed, thus, disabling Basic. When DUP is loaded, it overwrites the lower portion of memory. If you wish to save your program from destruction, you must have created a MEM.SAV file on the disk before you called DOS, or even simpler, just SAVEd it to the disk.

1792 - 1812

When software is booted, the MEMLO pointer points to the 1st free memory location above that software; otherwise, it's not affected and remains pointing to location 1792. The DUP portion of DOS is partly resident here, starting at 5440 and running up to 13062. DOS 2.5 takes up the 1st 78 sectors of a disk; the 1st sector is the boot sector, sectors 2 - 40 are the FMS portion and the remaining sectors 41 - 78 are the DUP.SYS portion of DOS. For full information on DOS, see the DOS and OS source listings including Inside Atari DOS.

FMS, DOS.SYS and DUP.SYS

Disk boot records (sector 1 of a DOS disk) are read into 1792, starting from this address the format of bytes is explained overleaf. Note, that the 1st 6 bytes of any disk are special-informatory bytes to the OS, explained fully in the BOOT-appendix, they tell the computer how much data to load, where to put it and where to execute within it.

BYTE	HEX	LABEL and USE:
0	700	BFLAG: Boot-flag equals 0 (unused),
1	701	BRCNT: Number of consecutive sectors to read, Set to 3 by DOS 2.X,
2,3	702,3	BLDADR: Boot sector load address, DOS points to 1792 ($700),
4,5	704,5	BIWTARR: Initialization address,
6	706	JMP XBCONT: Boot continuatin vector, JMP ($4C); JMP command to the address in bytes 7 and 8,
7,8	707,8	Boot read continuation address,
9	709	SABYTE: Maximum number of concurrently open files, defaulted to 3,
10	70A	DRVBYT: Drive bits, the maximum number of drives attatched to your system. Default is 2,
11	70B	Unused: Buffer allocation direction, set to 0.
12,3	70C,D	SASA: Buffer allocation start address at 1995,
14	70E	DSFLG: DOS flag equals nonzero. It must be nonzero for the 2nd phase of boot process. It indicates that the DOS.SYS has been written on the disk, 0 means no DOS.SYS, 1 = 128-byte sector and 2 = 256-bytes,
15,6	70F,10	DFLINK: Pointer to DOS.SYS' 1st sector on disk,

17	711	BLDISP: Displacement to the sector-link bytes (last 3). The sector link bytes point to the next disk-sector to be read. If 0, then EOF has been reached.
18,9	712,3	DFLADR Address for the start of the DOS.SYS file.
20+	714+	Continuation of the boot-load file, see the OS users manual for more info.

Data from the boot sector is placed in locations 1792 - 1916. Data from the rest of DOS.SYS is located starting from 1917 ($77D). All binary file-loads start with 255 twice, the next 4 bytes are the start and end addresses, see locations 736 and 737 for a full breakdown of this.

Here's a further explanation of locations 1801 and 1802:

1801 709 SABYTE

This records the limit for the number of files that can be OPEN simultaneously. Usually set to 3, the maximum is 7 (1 for each IOCB). Each available file takes 128 bytes for a buffer, so if you increase the number of buffers, you decrease your RAM space accordingly. If you make any changes to this register or any of the other registers following, then to keep the changes permanent, you should go to DOS and re-write the DOS files to a new blank formatted disk.

1802 70A DRVBYT

The maximum number of disk-drives in your system, default being 2. The least 4-bits are used to record which drives are available, so if you have drives 1, 2 and 4, the location would read:

00001011; decimal = 11.

Each drive has a seperate buffer of 128 bytes reserved for it, thus, including more drives in your system, decreases your RAM availability.

1900 76C BSIO

Entry point to the FMS disk sector I/O routines.

1906 772 BSIOR

Entry point to the FMS disk handler routines.

1913 - 2773

1913 779 ...

Write verify flag for disk I/O operations. POKE with 80 to turn off the verify function, 87 to turn it back on. Disk-write is much faster without verify.

1923 783 ...

Stores the drive number for the DUP.SYS file. If you POKE here with the ASCII equivalent of the drive number (ie. POKE 1923,50 for drive-2), when you call DOS from Basic, DUP.SYS will be loaded from the drive specified rather than the default D1:. Remember, permanent changes can be made by saving an altered DOS file to a new blank disk.

1995 7CB DFMSDH

Entry point of a 21-byte FMS disk handler. The address of this handler is placed in HATABS by the FMS initialization routine. When CIO needs to call an FMS function, it will locate the address of that function via the handler address table. See chapters 8-11 of Inside Atari DOS. Note, the data stored here is different with DOS 2.0 and DOS 2.5.

2016 7E0 DINT

FMS initialization routine. The entry point is 1995. DUP calls FMS at this point. K-DOS uses the same location for its initialization routine.

2219 8AB DFMOPN

OPEN routines, including open for append, update and output.

2508 900 DFMPUT

PUT byte routines.

2591 A1F WTBUR

Burst I/O routines.

2592-2773 A20-AD5 ....

In DOS 2.0, there is a burst I/O occurrence bug which takes place when a file is OPENed for update. This bug can be exterminated by:

POKE 2592,130
POKE 2593,19
POKE 2594,73
POKE 2595,12
POKE 2596,240
POKE 2597,36
POKE 2598,106
POKE 2599,234
POKE 2625,16
POKE 2773,31

You can completely disable burst I/O with a POKE 2606,0. This makes LOAD and SAVE operations considerably slower, though, so I wouldn't recommend saving it as a permanent change.

2751 ABF DFMGET

GET byte routines, including GET file routines.

2817 B01 DFMSTA

Disk STATUS routines.

2837 B15 DFMCLS

IOCB CLOSE routines.

2983 BA7 DFMDDC

Start of the device-dependent command routines, including the Basic XIO special commands.

3033 BD9 XRENAME

Rename file routine.

3118 C2E ....

POKE with 0 to force the rename routine to change only the 1st occurrence of files bearing the same name. POKE with 184 to revert to normal.

3122 C32 XDELETE

Delete file routine.

3196 C7C XLOCK,XUNLOCK

Lock file routine. Unlock file routine begins at 3203 ($C83).

3258 CBA XPOINT

Basic POINT command routine.

3331 D03 XNOTE

Basic NOTE command routine.

3352 D18 XFORMAT

Format disk routine.

3460 D84 ....

De-allocation bytes of the VTOC and directory; see 4226, 4229, 4264, and 4266.

3501 DAD LISTDIR

List directory routine.

3742 E9E FNDCODE

Filename decode, including wildcard validity test. The current filename is pointed to by ZBUFP at 67 and 68 ($43 and $44).

3783 EC7 ....

By POKEing the desired ATASCII value here, you can change the "*" wildcard character used by DOS. Don't forget that changes can be made permanent by re-writing DOS. Either goto the DOS menu and use option H, or OPEN #1,8,0,"D:DOS.SYS" and CLOSE #1 from Basic.

3818 - 4206

3818,3822 EEA,EEE ....

By POKEing 3818 with 33 and 3822 with 123, you can modify DOS to accept filenames with punctuation, numbers and lowercase as valid. 33 is the low range code and 123 for the high range. Of course, you could change the range of accepting characters from 0 - 255, but you will have problems with spaces and the wildcards. Be sure that the wildcard character is not in this range.

3850 F0A FDSCHAR

Store the file name characters that result from the filename decode routine.

3873 F21 SFDIR

Directory search routine; search for the user-specified filename.

3889 F31 DOS3

If you PEEK here and get 76 ($4C), you have an early version of DOS 3, the later version will read 78. To correct some errors in the earlier version, type:

```
10 FOR I=1 TO 9
20 READ A,D:POKE A,D:NEXT I
30 DATA 3889,78,3923,78,3943,78,3929,76,3895,76
40 DATA 3901,77,3935,77,3955,77,2117,240
```

Better yet, to eradicate such a stupid move, chuck DOS 3 in the bin and get hold of DOS 2.5. DOS 3 is a serious space-waster!

3988 F94 WRTNXS

Write data sector routine.

4111 100F RDNXTS

Read data sector routine.

4206 106E RDDIR

Read and write directory sector routines.

4226 1082 ....

LSB of the current directory sector. The directory is normally located in sectors 361 - 368. Default here is 105.

4229 1085 ....

MSB of the current directory sector. To change the location of the directory, copy the 8 directory sectors from 361 - 368 into your desired area on the disk and POKE the address of the 1st sector into 4226 and 4229. Finally, write the value of the new sector number (sector/8+10) into 3460.
The FORMAT of a directory entry is comprised of 16 bytes. The bytes are as explained:

BYTE: USE:
0 Flag
- $00 Entry new (never used)
- $01 File currently OPEN
- $02 File created by DOS 2
- $20 File locked
- $40 File normal status
- $80 File deleted

1-2 Number of sectors in the file
3-4 Starting sector of the file
5-12 Filename (space or $20 if blank)
13-15 Extension

If you've deleted a file, but later you regret it, you can usually undelete it (bring it back to life) by using a sector editor. When a file is deleted, the actual data and filename remains on the disk, if you write something else on the disk, then the deleted file data will be overwritten, but if you have not written over the disk, then you should be able to reinstate your file by clearing bit-7 ($80) in byte 0 of the directory entry in the directory sectors. If you want to undelete any files that have been deleted on your DOS 2.X disk, then use this program:

```
10 X=0
12 DATA 104,32,83,228,96
14 FOR I=0 TO 4
18 READ D:POKE 1536+I,D:NEXT I
22 POKE 769,1
26 POKE 772,253:POKE 773,3
30 FOR K=361 TO 368
34 SHI=INT(K/256):SLO=K-(SHI*256)
38 POKE 778,SLO:POKE 779,SHI
42 POKE 770,82
46 X=USR(1536)
50 FOR I=0 TO 127 STEP 16
54 BYT=PEEK(1021+I+X)
58 IF BYT-127 THEN BYT=BYT-128
62 POKE 1021+I+X,BYT
66 NEXT I
70 POKE 770,80
74 X=USR(1536)
78 NEXT K
```

4235 - 4266

Sometimes, your files can be accidentally left OPEN and, thus, are unretainable. I've lost a lot of my files in the past through drive problems. Usually, the drive writes a file to the disk, but doesn't close it properly. If this happens, then you can use the program re-leafed to bring them back. Just alter line 58 to read:

```
58 IF  NOT BYT/2=INT(BYT/2) THEN BYT=BYT-1
```

If you still get problems, then the last effort to regain as much of your file is to use a sector editor, and alter the 2nd and 3rd bytes of the appropriate directory entry to $FF's. This way, as much as possible of the existing file will be regained when loaded

4235 108B RDVTOC

Read or write the volume table of contents (VTOC) sectors.

4264 10A8 ....

LSB of the current VTOC sector.

4266 10AA ....

MSB of the current VTOC sector, which is normally sector 360. The VTOC sector is a bitmap of the disk contents; after the initial status bytes, each of the following bits represents 1 sector on the disk in sequential order. There are 720 sectors on the single-density disk. The 1st 4 are reserved 'BOOT' sectors on DOS, sectors 360 - 368 are reserved for VTOC and the directory, leaving 707 free for use. You can move VTOC in the same way as the directory.
If you change the directory location, ensuring the destination for the new directory uses unused sectors, you should also alter the VTOC sector to de-allocate the original directory sectors (by setting these bits), and clear the bits of the new directory area to protect it from being overwritten.

You can also use this technique to lock out particular sectors on a disk for miscellaneous use.

The FORMAT of the VTOC sector is as follows:

4293 - 4618

BYTE:	USE:
0	DOS code (0 = DOS 2.0) (2 = DOS 2.5)
1-2	Total number of sectors; 707 single density 1010 dual density
3-4	Number of currently unused sectors
5-9	Unused
10-99	Bitmap: 1 bit for each sector: 0=in use/locked, 1=unused/free The leftmost bit of byte 10 is sector 0 (unaccessable), the next bit is sector 1 and so on, until the rightmost bit of byte 99, which is sector 719. Sector 720 is unused on any DOS 2.X disk
100+	Bytes 100 - 127 are unused

Within the bitmap area of used and unused sectors, the VTOC is the leftmost bit of byte 55, and the directory sectors are the remainder of the same byte and the 1st bit of byte 56. The leftmost 4-bits of byte 10 are the boot sectors, and the remainder of the bytes up to and including the leftmost 7-bits of byte 24 is taken by the DOS and DUP files. Disk directories and the VTOC are discussed in Inside Atari DOS.

4293 1OC5 FRESECT

Free sectors routine; returns the amount of free sectors available on a disk.

4358 1106 GETSECTOR

Get sector routine; retrieves the lowest unused sector for use off the disk.

4452 1164 SETUP

SETUP - initialization of the FMS parameters. Prepares FMS to deal with the operation to be performed and to access a particular file. See Inside Atari DOS, chapter 7.

4618 120A WRTDOS

Write new DOS.SYS file to disk routine, including new FMS file to DUP.SYS file.

4789 - 5377

4789 12B5 ERRNO

Start of the FMS error number table.

4856-4978 12F8-1372 ....

Miscellaneous FMS storage area; sector length, drive type, stack level, file-number etc..

4993-5120 1381-1400 FCB

Start of the FMS file Control Blocks (FCB's). FCB's are used to store information about files currently being processed. The 8 FCB's are 16-bytes each in length and correspond to a one-on-one manner with the IOCB's. Each FCB takes the following format:

LABEL:	BYT:	USE:
FCBFNO	1	Current file-number being processed
FCBOTC	1	File OPEN mode: 1=append, 2=directory, 4=input, 8=output and 12=update
SPARE	1	Unused
FCBSLT	1	Sector length type flag: 128 or 256 bytes,
FCBFLG	1	Work flag: 128= file OPEN for output and 64= buffer sector should be output,
FCBMLN	1	Max. sector data length: 125 or 253,
FCBDLN	1	Current byte for read/edit in the sector,
FCBBUF	1	Tells FMS which buffer is used by the file,
FCBCSN	2	Sector number in the buffer of the file in use,
FCBLSN	2	Next sector number in the chain-link,
FCBSSN	2	Start sector for file appending data,
FCBCNT	2	Sector count for the current file.

DUP doesn't use these FCB's; it writes to the IOCB's directly. CIO transfers the control to FMS as the operation demands, then onto SIO.

5121 1401 FILDIR

File directory, a 256-byte sequential buffer for entries to the disk directory.

5377 1501 ENDFMS

Disk directory (VTOC) buffer. 64-bytes are reserved, 1-byte for each possible file. It also marks the end of FMS.

5440 -

The VTOC (sector 360; $168) is a sequential bitmap of each of the 720 sectors on a DOS 2.0 disk. It starts at byte 10 and continues to byte 99. See 4264 and 4266.

5440 1540 DOS

DUP.SYS initialization address. Beginning of mini-DOS; the RAM-resident portion of DUP. Used for the same purpose in K-DOS.

5446,5450 1546,154A ....

Contains the address stored in DOSVEC at locations 10 and 11. This points to the address Basic jumps to upon execution of the DOS command.

5533 159D DUPFLG

Flag to test if DUP is already resident in memory. 0 means it's not.

5534 159E OPT

Used to store the value of the disk menu option chosen by the user.

5535 159F LOADFLG

If this location reads 128, then a memory file (MEM.SAV) doesn't have to be loaded.

5540 15A4 SFLOAD

Routine to load a MEM.SAV file if it is present on the disk.

5576 15C8 ....

You can run some machine-language files from Basic with OPEN #1,4,0,"D:FILENAME.EXT" and then doing a USR to this address.

5888 1700 USRDOS

Listed in the DUP.SYS equates file but not explained in the listings.

5899 - 6518

5899 170B MEMLDD

Flags that the MEM.SAV file has been loaded. 0 means it hasn't been loaded.

5947 173B ....

The MEM.SAV file creation routine begins here. It starts with the filename "MEM.SAV" stored in ATASCII format. The write routines begin at MWRITE in 5958. The DOS utility MEMSAVE copies the lower 6000 bytes of memory to disk to save your Basic program from being destroyed when you call DOS, which then loads DUP.SYS into that area of memory afterwards.

6044-6045 179C-179D INISAV

DOSINI vector save location, transferred down to locations 12 and 13. Entry point to DOS called from Basic.

6046 179E MEMFLG

Flag to show if memory has been written to disk using a MEM.SAV file.

6418 1912 CLMJMP

Test to see if DOS must load MEM.SAV from the disk before it does a run at cartridge address, then jumps to the cartridge afterwards.

6432 1920 LMTR

Test to see if DOS must load MEM.SAV before it performs a run at address command from the DOS menu.

6457 1939 LDMEM

MEMSAVE load routines, for the MEM.SAV file.

6518 1979 INITIO

DUP.SYS warmstart entry. An apparently excellent program to eliminate the need for DUP.SYS and MEM.SAV was presented in COMPUTE!, July 1982 called MicroDOS. See also "The Atari Wedge", COMPUTE! December 1982.

6630 - 7668

6630 19E6 ISRODN

Start of the serial interrupt service routine to 'output-data needed' routines in DUP.SYS.

6691 1A23 ISRSIR

Start of the serial interrupt ready service routines in DUP.SYS.

6781 1A7D ....

Start of the drive and data buffers. Drive buffers are numbered sequentially 1 - 4, data buffers are 1 - 8, assuming that many are allocated for each. Normally, the 1st 2 buffers are allocated for drives and the next 3 for data. Buffers are 128 bytes long each and begin at 6908, 7036, 7162 and 7292 ($1AFC, $1B7C, $1BFA and $1C7C). See locations 1801 and 1802.

7420 1CFC ....

MEMLO at 743,744 points here when DOS is resident unless the buffer allocation has been altered. MEMLO will point to 7164 for a 1-drive, 2 data buffer setup, a saving of 256 bytes. Loading the RS-232 handler from the 850 interface will raise MEMLO an extra 1728 bytes. The RS-232 handler in the 850 interface will only boot (load into memory) if you first boot the AUTORUN.SYS file on the original DOS masterdiskette. The RS-232 handler will boot-up into memory if you don't have a disk-drive attached assuming you have turned it on prior to the computer. Whether the RS-232 handler is booted or not, you can still use the printer parallel port on the 850.

7548 1D7C ....

Beginning of the non-resident portion of DUP; 40-byte parameter buffer.

7588 1DA4 LINE

80-byte line buffer.

7668 1DF4 DBUF

256-byte data buffer for the COPY routines. Copy routines work in 125-byte passes, equal to the number of data-bytes in each DOS sector on the disk.

7924 - 8990

There are 256-bytes because Atari accounted for the now existing double-density which gives 253 data-bytes per DOS sector. The US-Doubler is such an example modification to your disk-drive well worth making giving you the true-double density and accelerated speed.

7924 1EF4 ....

Miscellaneous variable storage area and data buffers.

7951-8278 1F0F-2056 DMENU

Disk-menu screen display data is stored here.

8191 1FFF ....

This is the top of minimum RAM required for operation (8K), to use DOS you must have a minimum of 16K.

DUP.SYS ROUTINES:
Locations 8192 - 32767 ($2000 - $7FFF) are the largest part of the RAM expansion area; this space is generally for your own use. If you have DOS.SYS or DUP.SYS loaded in, they also use a portion of this area to 13062 ($3306) below:

8309 2075 DOSOS

Start of the DOS utility monitor, including the utilities called when a menu selection fuction is completed and the display of the "Select item" prompt.

8505 2139 DIRLST

Directory listing.

8649 21C9 DELFIL

Delete a file.

8990 231E ....

Copy a file. This area starts with the copy messages. The copy routines themselves begin at PYFIL in 9080 ($2378).

9783 - 10690

9783 2637 RENFIL

Rename a disk-file routine.

9856 2680 FMTDSK

Format the entire disk. There is no way to format specific sectors in the standard 810s or 1050s. The Archiver chip allows you to do this, however, if you have one fitted.

9966 26EE STCAR

Execute a cartridge.

10060 274C BRUN

Run a binary-file at the user specified address.

10111 277F ....

Start of the write MEM.SAV file to disk routine. The entry point is at MEMSAV in 10138 ($279A).

10201 27D9 WBOOT

Write DOS/DUP files to the disk.

10483 28F3 TESTVER2

Test for version 2 DOS.

10522 291A LDFIL

Load a binary file into memory. If it has a run-address specified in the file, it will autoboot, unless you append "/A" to the binary load option L from the DOS menu.

10608 2970 LKFIL, ULFIL

Lock and unlock files on disk.

10690 29C2 DDMG

Duplicate a disk.

11528 - 49151

11528 2D08 DFFM

Duplicate a file.

11841 2E41 ....

Miscellaneous routines.

13062 3306 ....

End of DUP.SYS

20480-22527 5000-57FF SELFTEST

Self-test ROM when enabled. The Self-test ROM is switched into these addresses when you clear bit-7 in PORTB at location 54017 ($D301), thus, losing 2K of RAM in the process.
It's normally located under the Hardware memory at 53248 - 55295 ($D200 - $D7FF), and re-addressed, as above, when you type BYE in Basic, or turn the computer on with OPTION pressed without a disk-drive attached.

Location 13063 is the 1st free RAM location with DOS installed. The eternally free RAM memory expands up to 32767 ($7FFF) within Basic. Without Basic, you can safely use up to 40959 ($9FFF). Free RAM depends on what cartridge you are using; Basic or Assembly etc.. It also depends on the Graphics mode in use.

32768-40959 8000-9FFF CARTRIDGE-B

In the old Atari 800, this used to contain the right cartridge when present, and RAM otherwise. In the XL/XE's, this can now be considered as the lower of the 2 8K banks at the top end of RAM. When Basic is enabled, this area contains the Display List (DL) and Display Memory (DM). But, when Basic is disabled, this extra 8K is free RAM and the DL and DM occupy the higher of these 2 8K banks. This applies to the Assembler/Editor cartridge as well, or any other cartridge for that matter.

40960-49151 A000-BFFF CARTRIDGE-A

This was the left-cartridge slot in the old Atari 800, but can now be considered as the higher of the 2 8K banks at the top end of RAM.

40960 - 43631

When Basic is disabled, this area contains the DL and DM, but when Basic is enabled, the 8K RAM is switched-out and the 8K Basic-ROM is switched-in. You can convert the ROM Basic to a RAM Basic alike the OS, see location 54017 and create your own Basic commands. Another method of achieving this is to trap the keystrokes before they get passed to the Basic editor. You can find further information about this in COMPUTE!'s 3rd book of Atari.
A USR call here will coldstart the Basic cartridge when enabled, or any other cartridge inserted for that matter.
Listed below are the Basic routines and their addresses:

40960-41036 A000-A04C Coldstart

41037-41055 A04D-A05F Warmstart

41056-42081 A060-A461 Syntax checking routines

42082-42158 A462-A4AE Search routines

42159-42508 A4AF-A60C Statement name table

The statement TOKEN list begins at 42161 ($A4B1) and can be listed with this program:

```
10 XDRS=42161:TOK=0
20 IF NOT PEEK(XDRS) THEN ? :END
30 ? TOK,
40 BYT=PEEK(XDRS):XDRS=XDRS+1
50 IF NOT BYT-127 THEN ? CHR$(BYT);:GOTO 40
60 ? CHR$(BYT-128)
70 XDRS=XDRS+2:TOK=TOK+1
80 GOTO 20
```

42509-43134 A60D-A87E Syntax tables

The OPERATOR token list begins at 42979 ($A7E3) and can be listed with the previous program if you change TOK in line-10 to TOK=16, XDRS to 42979 and line-70 should only read TOK=TOK+1.
See the Basic TOKEN appendix for further information.

43135-43358 A87F-A95E Memory manager

If you PEEK location 43234 ($A8E2) and get back 96, you have Revision B ROM, B stands for BUGS, so you should try to get hold of Revision C. In all my experiences, the B ROM tends to come with the flatter (older) XL keyboards.

43359-43519 A95F-A9FF Execute CONT statement

43520-43631 AA00-AA6F Statement table

43632-43743	AA70-AADF	Operator table
43744-44094	AAE0-AC3E	Execute Expression routine
44095-44163	AC3F-AC83	Operator precedence routine
44164-45001	AC84-AFC9	Execute operator routine
45002-45320	AFCA-B108	Execute function routine
45321-47127	B109-B817	Execute statement routine
47128-47381	B818-B915	CONT statement subroutines
47382-47542	B916-B9B6	Error handling routines
47543-47732	B9B7-BA74	Graphics handling routines
47733-48548	BA75-BDA4	I/O routines
48549-49145	BDA5-BFF9	Floating-point routines:
48551	BDA7	SIN

Calculate SIN(FREO). Checks DEGFLG at 251 to see if trigonometric calculations are in radians or degrees.

48561	BDB1	COS

Calculate COSine(FR0) with carry. FR0 is Floating-Point register 0, locations 212 - 217. See FP entry points from 55296 onward.

48759	BE77	ATAN

Calculate Atangent using FR0, with carry.

48869	BEE5	SQR

Calculate square root (FR0) with carry. Note, that there is some conflict of addresses for the above routines. The addresses given are from De Re Atari. The OS Source-code listing gives the following entry-point addresses for these FP routines:

SIN	48513	($BD81)
COS	48499	($BD73)
ATAN	48707	($BE43)
SQR	48817	($BEB1)

These are the ones to ignore! Because they are WRONG!

49146,7 BFFA,B ...

Cartridge start address.

49148 BFFC ...

A nonzero value here tells the OS there is no cartridge installed (?).

49149 BFFD ...

Option byte. A cartridge which does not specify a disk-boot may use all the memory from 1152 ($480) to MEMTOP any way possible.

49150,1 BFFE,F ...

Cartridge initialization address.

When a Basic program is SAVEd, only 14 of the more than 50 Page-0 locations Basic uses are written to the disk/cassette along with the program. The rest are all re-calculated with a NEW or SAVE command, sometimes with RUN and GOTO. These 14 locations are:

128,129	80,81	LOMEM
130,131	82,83	VNTP
132,133	84,85	VNTD
134,135	86,87	VVTP
136,137	88,89	STMTAB
138,139	8A,8B	STMCUR
140,141	8C,8D	STARP

The string/array space is not loaded; STARP is included only to point to the end of the Basic program.
The 2 other critical Basic Page-0 pointers, which are not SAVEd, are:

142,143	8E,8F	RUNSTK
144,145	90,91	MEMTOP

For more information concerning Atari Basic, get hold of a 2nd hand copy of a good book such like: The Atari XL Handbook by Lupton & Robinson, Your Atari computer by Lon Poole or any of the fine COMPUTE! books such as 2nd book of Atari Graphics or 1st and 2nd books of Atari. You should also browse through the BASIC appendix given in this book.

49152 - 49808

49152-53247   C000-CFFF   OSROM

This 4K block was unused and unuseable in the old Atari' (very sad), but, thanks to Atari, this Pain-up-the-rear is now sorted! You can use any of the Translator disks to revert back to the old OS, in doing so, this area becomes 4K of user accessible RAM, Great EH!
Anyway, the C-Block now contains various interrupt handlers (vectored here from Page-2) and other routines:

49164-52223   C00C-CBFF   Interrupt handlers

A lot of interrupt vectors are set to jump to 49357 ($C0CD) or 49358 ($C0CE). The former contains a PLA and an RTI. The net result is a simple return back into the program without any other activity taking place.

Bytes 49152 - 49163 ($C000 - $C00B) are used to identify the computer and the ROM in the $C000 - $DFFF block.

BYTE:	USE:
49152,3/C000,1	Checksum of all the bytes in ROM except the actual checksum bytes.
49154/C002	Revision date, stored in the form DDMMYY. This is DD, day.
49155/C003	Revision date, month.
49156/C004	Revision date, year.
49157/C005	Reserved option byte, reads 0 for 1200XL, 800XL and 130XE.
49158/C006	Part number, in the form AANNNNNN AA = Ascii character, and the NNNNNN = 4-bit BCD digit; byte-A1.
49159,62/C007,A	Part number, bytes A2, N1-N6 (each byte has 2 N values of 4-bits).
49163/C00B	Revision number.
49164/C00C	Interrupt handler initialization.
49176/C018	NMI initialization.

Interrupt handlers and other routines in the C-block:

ENTRY:	HANDLER/USE:
49196/C02C	IRQ Processor
49298/C092	BREAK key IRQ
49312/C0A0	Continue IRQ processing
49359/C0CF	Table of IRQ types and offsets (16-bytes)
49378/C0E2	Immediate VBLANK NMI processing
49743/C24F	Process countdown timer-1 expiration
49746/C252	Process countdown timer-2 expiration
49749/C255	Decrement countdown timer
49778/C272	Set VBLANK parameters
49802/C28A	Process deferred VBLANK NMI
49808/C290	Perform Warmstart

49834 - 52069

ENTRY:	HANDLER/USE:
49834/C2AA	Process RESET
49864/C2C8	Perform Coldstart
49866/C2CA	Preset memory; cold/warm start continuation
50217/C429	Initialize cartridge software
50220/C42C	Process ACMI interrupt
50237/C43D	BOOT-ERROR message
50248/C448	Screen-editor specification; E:
50251/C44B	Table of interrupt handlers in the same order as RAM vectors at 512 - 549; $200 - $225
50289/C471	Miscellaneous initialization routines: OPTION-key checked at 50330/$C49A Basic enabled at 50337/$C4A1
50394/C4DA	Hardware initialization
50485/C535	Software and RAM variable initialization
50571/C58B	Attempt disk-boot
50619/C5BB	Boot and initialize disk
50633/C5C9	Complete boot and initialize
50729/C629	Execute boot loader
50747/C63B	Initialize booted software
50750/C63E	Display BOOT-ERROR message
50777/C659	Get next sector routine
50798/C66E	Attempt cassette boot
50851/C6A3	Initialize DIO; Disk I/O
50867/C6B3	DIO; Disk I/O
51002/C73A	Set buffer address
51013/C745	Relocate relocatable routine to new address
51093/C795	Handle end record type
51151/C7CF	Get byte
51154/C7D2	Execute Run-at-address
51157/C7D5	Handle text record
51281/C851	Relocate text into memory
51309/C86D	Handle word reference record type
51346/C892	Handle low-byte and 1-byte record type
51452/C8FC	Select and execute Self-test
51468/C90C	Initialize generic parallel device
51507/C933	PIO-Parallel device I/O; PIO vector tables (see 58368; $E400) begin at 51601; $C991
51631/C9AF	Select next parallel device
51658/C9CA	Invoke parallel device handler
51753/CA29	Load and initialize peripheral handler
51799/CA57	Start of the Self-test offsets and text (uses hardware values for character display)
52054/CB56	Checksum linkage table
52069/CB65	Empty/zeroed

52224 - 53505

52224-53247 CC00-CFFF CHARSET2

International character-set, assembled in the same manner as the standard character-set at 57344 ($E000). There are 2 character-sets in the XL/XE, and you can change between them with POKE 756,224 for the standard one and POKE 756,204 for the international one. The only difference is the CTRL-key characters. Standard gives you graphics characters, while the international one gives you the phonetic symbols for writing in other languages.

Locations 53248 - 55295 ($D000 - $D7FF) are the ROM special I/O Large-Scale Integration (LSI) chips that give the XL/XE it's power. There is the GTIA, POKEY, PIA and ANTIC. GTIA uses 53248 - 53503 ($D000 - $D0FF), POKEY uses 53760 - 54015 ($D200 - $D2FF), PIA uses 54016 - 54271 ($D300 - $D3FF) and ANTIC uses 54272 - 54783 ($D400 - $D5FF). For the most extensive description of these chips, see the Atari Hardware manual, or checkout my HARDWARE-CHIPS appendix.

Many of the following registers can't be read directly, since they are hardware registers. Writing to them can often be difficult because in most cases the registers are updated every stage-1 or stage-2 VBlank. The values in these locations are copied up from their shadow registers in RAM. To affect any permanent change, you'll need to POKE the shadow registers themselves. This way, the hardware register/s will be updated at the next stage-1 or stage-2 VBlank. Defaults are returned on RESET by transferring the appropriate values from the actual ROM in higher memory.
Another feature of the hardware memory is the dual purpose of registers. Some registers are PEEKed for one purpose, but POKEd to for a completely different purpose. For this reason, you should avoid performing Basic expressions such like: POKE 53248,PEEK(53248)+1. This will not consecutively increment this memory location. Where a register is used for 2 different purposes, it is indicated with a (R) and a (W) for READ and WRITE, respectively. Where (R) or (W) is on its own, then this is all you can do; Read from it OR Write to it.

53248-53505 D000-D0FF GTIA

GTIA is a special television interface chip designed exclusively for the Atari to process the video signal. ANTIC controls most of the GTIA chip functions. The GTIA chip shifts the display 1/2 a colour-clock so that players and playfields can overlap perfectly. This, however, results in a very slight colour difference from the older CTIA chip (wow).

GTIA modes don't normally offer a text-window, but there are ways of obtaining one. For convenience, you can call your GTIA mode and POKE 703,4. The text isn't readable like this, but as I say, it gives the convenience of stopping program execution without returning to a Graphics 0 screen. You should also be able to get a full screen in any mode, by adding 16 to the mode number prior to POKEing 703 with 4. The Display memory for the window is 1-byte above the main screen memory.
On the other hand, if you would like a readable text-window in a GTIA mode, then you can achieve this with a DLI. See the DLI appendix about this.
By the way, Mapping states that GTIA stands for "George' Television Interface Adapter".

In the following list of hardware registers, the shadow registers are enclosed in parentheses; you can see these locations for additional information or programs in some cases.

53248 D000 (W) HPOSP0
(R) M0PF

(W) Horizontal position of Player #0. Values from 0 - 227 are possible here, but depending on the playfield size, visible areas change. In the standard width playfield (see location 559), the left-edge to the right is 48 - 208. Other positions are off-screen. POKEing the players to a 0-position is a way of affectively turning the players off when not using PMBASE. See this location at 54279 for further details.
The players are usually tall and thin. They are only 8-bits wide, although, each bit can be echoed between 1 and 3 colour-clocks, see the SIZE registers. They stretch from the very top of the screen to the very bottom, in single line resolution the range is 32 - 224, in double line resolution the range is 16 - 112. See the PMG appendix for full details on Player/Missile Graphics.
As soon as you POKE this register with the horizontal position for the player, this value is 'no longer'. You cannot perform: POKE 53248,PEEK(53248)+1 to move the player, you must keep a recorded position in RAM or in a variable. Try:

```
10 POKE 53261,255:POKE 53256,1
20 XCO=50
30 S=STICK(0)
40 POKE 53248,XCO
50 V=(S=7 AND (NOT XCO-227))-(S=11 AND (NOT XCO))
60 XCO=XCO+V
70 GOTO 30
```

For vertical movement of players/missiles, see the PMG appendix.

(R) The PEEK purpose of this register is to detect Missile #0 to playfield collision. This tells you which playfield is in collision with missile #0:

BIT:	DEC:	USE:
7-4		unused...
3	8	Playfield #3
2	4	" #2
1	2	" #1
0	1	" #0

All the 4 HPOSP/M#PL registers take the same format as described above. Also, see HITCLR at 53278 about collisions.

53249 D001 (W) HPOSP1
(R) M1PF

(W) Horizontal position of player #1.
(R) Missile #1 to playfield collisions.

53250 D002 (W) HPOSP2
(R) M2PF

(W) Horizontal position of player #2.
(R) Missile #2 to playfield collisions.

53251 D003 (W) HPOSP3
(R) M3PF

(W) Horizontal position of player #3.
(R) Missile #3 to playfield collisions.

53252 D004 (W) HPOSM0
(R) P0PF

(W) Horizontal position of missile #0. Missiles are alike players, although are only made of 2-bits in width.

(R) Player #0 to playfield collisions. There can be some confusion and problems using collision detection and prioritizing in GTIA Graphics modes because the collision playfields only apply to registers 53270 - 53273 ($D016 - $D019). In Graphics 10, playfield colours are set by PCOLR0 - 3 (704 - 707) and they behave like players where priorities are concerned. The background register also changes from shadow register 712 to register 704.
In some cases, a player to playfield collision also shows up in the P#PL register, because the registers in use are the same.

53253 - 53257

The bit use is exactly the same format as with the MOPF collision register at 53248 except for Player #0 to playfields.

53253 D005 (W) HPOSM1
(R) P1PF

(W) Horizontal position of missile #1.
(R) Player #1 to playfield collisions.

53254 D006 (W) HPOSM2
(R) P2PF

(W) Horizontal position of missile #2.
(R) Player #2 to playfield collisions.

53255 D007 (W) HPOSM3
(R) P3PF

(W) Horizontal position of missile #3.
(R) Player #3 to playfield collisions.

53256 D008 (W) SIZEP0
(R) MOPL

(W) Size of player #0. POKE with 0 or 2 for normal size, 1 for double width and 3 for quadruple width. Each player can have its own width, bit use is:

BIT:	DEC:	WIDTH:	
7-2		unused...	
1-0	0	0 0	Normal; 8 colour-clocks
	1	0 1	Double; 16 " "
	2	1 0	Normal
	3	1 1	Quadruple; 32 " "

(R) Missile #0 to player collisions. Again, the same format as 53248 except for missiles to players.

53257 D009 (W) SIZEP1
(R) M1PL

(W) Size of player #1.
(R) Missile #1 to player collisions.

53258 D00A (W) SIZEP2
(R) M2PL

(W) Size of player #2.
(R) Missile #2 to player collisions.

53259 D00B (W) SIZEP3
(R) M3PL

(W) Size of player #3.
(R) Missile #3 to player collisions.

53260 D00C (W) SIZEM
(R) P0PL

(W) Size of all 4 missiles; each missile only requires 2-bits each, so all these are set in just the 1-byte:

```
BIT:  7 6  5 4  3 2  1 0
M#    -3-  -2-  -1-  -0-
DEC:  1
      2 6  3 1
      8 4  2 6  8 4  2 1
```

The size selection works the same way as in SIZEP0 at 53256, except for the particular bit-pair, which denote the missile#.
If you wanted to select double width in missiles #1 and #3, then you would set bits 3 and 7, thus, give decimal values 8 + 128 = 136.

```
BIT-pair:
0 and 0: normal size -    2 colour-clocks wide
0 and 1: double size -    4      "        "
1 and 0: normal
1 and 1: Quadruple size - 8      "        "
```

(R) Player #0 to player collisions. Again, the bit use is alike all other collision registers, except for player #0 to player collisions.

53261 D00D (W) GRAFP0
(R) P1PL

(W) Graphics shape for player #0 written directly to the player graphics register. In using these registers, you bypass ANTIC. You only use the GRAFP# registers when you are not using Direct Memory Access (DMA) (see GRACTL at 53277 for DMA).

53261 cont.

If DMA is enabled, then the graphics registers will be loaded automatically each single or double scan-line with the users given data, pointed to by PMBASE at 54279. Without PMBASE, the GRAFP# registers can only echo the same 'bit-shape' value throughout the graphic (top to bottom). For example:

```
10 POKE 53248,160:POKE 704,245
20 POKE 53256,3
30 POKE 53261,PEEK(20):GOTO 30
```

To remove the data from the screen, but retain the present horizontal position of the graphic, just POKE 53261 with 0. Each bit set in this register runs the entire height of the screen as you'll see with the example program. The handy thing with using the GRAFP# registers is that you can use a PMG for screen boundaries. You don't have to use PMBASE to change the shape of a graphic either, if you just want to create several blocks (with the same graphic) at different positions on the screen, then you can use the following program. This can also be very handy for selecting the coloured bars to choose a menu option on the screen:

```
10 DATA 72,138,72
12 DATA 166,203,189,64,6,141,0,208
14 DATA 41,240,5,204,141,18,208
16 DATA 198,203,16,4,169,24,133,203
18 DATA 104,170,104,64
20 FOR I=0 TO 29
22 READ D:POKE 1536+I,D:NEXT I
30 FOR K=2 TO 28:IF K=4 THEN K=6
32 POKE DL+K,PEEK(DL+K)+128:NEXT K
40 POKE 704,130:POKE 203,24:POKE 204,0
42 POKE 53256,3:POKE 53261,255
50 POKE 512,0:POKE 513,6
52 POKE 54286,192
60 FOR COL=0 TO 24
62 POKE 1624-COL,48+COL*4:NEXT COL
```

The program is fairly straightforward; the DLI is POKEd into memory and the DL has the DLI-bit set on every line. The DLI uses location 203 as a line-count, so don't use this location. Location 204 is a luminance control and colour-shifter for the graphic. The line and column is achieved on lines 60 - 62 of the program. The column takes the formula: 48+COLUMN*4, just substitute the column number in the expression. The line that your on depends on the memory location you POKE the column into. Locations 1600 - 1624 are used for the 24 on-screen lines and the position of the graphic on the border. Note, that the lines are actually reversed; hence, the top-line is at location 1624 and the bottom-line is at 1601. Location 1600 is the border position of the graphic.

53262 - 53265

(R) Player #1 to player collisions.

```
53262          D00E         (W) GRAFP1
                            (R) P2PL
```

(W) Graphic for player #1.
(R) Player #2 to player collisions.

```
53263          D00F         (W) GRAFP2
                            (R) P3PL
```

(W) Graphic for player #2.
(R) Player 3 to player collisions.

```
53264          D010         (W) GRAFP3
                            (R) TRIG0
```

(W) Graphic for player #3.

(R) Joystick trigger 0 (location 644). Controller jack 1, pin-6. For all the triggers, 0 means trigger is pressed and 1 means released. If Bit-2 of GRACTL at 53277 is set to 1, then all TRIG bit-0's are latched (set to 0) when any trigger button is pressed, and are only reset to 1 (not pressed) when the latch bit is cleared at GRACTL. This affect of latching triggers is to return a 'constant button pressed' value until the latch-bit is cleared.

```
53265          D011         (W) GRAFM
                            (R) TRIG1
```

(W) Graphics for all missiles, not used with DMA (same as players). GRAFM works in the same way as GRAFP0 described earlier. Each pair of bits represents one missile as missiles are only 2-bits wide:

```
BIT:   7 6   5 4   3 2   1 0
M#     -3-   -2-   -1-   -0-
       1
       2 6   3 1
       8 4   2 6   8 4   2 1
```

Each bit will create a vertical line down the TV screen. To turn off any missiles, just disable (clear) the bit-pair for the missile you wish to disable. If you wished to make missile #3 2-bits wide and missile #1 just 1-bit wide, you would set bits: 7, 6 and 3 (or bit-2 instead of 3); thus, 128 + 64 + 8 = 200. POKE 53265,200.

(R) Joystick trigger 1 (645). Controller jack 2, pin-6.

53266 D012 (W) COLPM0
(R) TRIG2

(W) Colour and luminance of player and missile #0 (704). Missiles share the same colours as their associated players, except when joined together to make the 5th player with bit-4 of GPRIOR (623), in which case, the 4-missiles then assume the colour stored at location 53733 (711).

(R) TRIG2; No longer used.

53267 D013 (W) COLPM1
(R) TRIG3

(W) Colour and luminance of player and missile #1 (705).
(R) TRIG3; No longer used.

53268 D014 (W) COLPM2
(R) PAL

(W) Colour and luminance of player and missile #2 (706).

(R) Denotes whether your Atari is PAL (European and Israeli TV compatible when value here is 0) or NTSC (North American compatible when value here is 13). PAL Atari' TV frames are refreshed every 50th of a second (12% slower than NTSC), where NTSC refreshes its frames every 60th of a second. For this reason, the 6502 microprocessor in PAL Atari' works at 2.217 MHz, which is 19% faster than the 1.79MHz NTSC 6502. Also, their $E000 and $F000 ROMS are different, so there may be some incompatibilities in the cassette handling routines. There is a 3rd TV standard called SECAM, used in France, USSR and parts of Africa. If Atari supports SECAM, I don't know. See the PAL/NTSC appendix.

53269 D015 COLPM3

(W) Colour and luminance of player and missile #3 (707).

53270 D016 COLPF0

(W) Colour and luminance of playfield #0 (708).

53271 D017 COLPF1

(W) Colour and luminance of playfield #1 (709).

53272 D018 COLPF2

(W) Colour and luminance of playfield #2 (710).

53273 D019 COLPF3

(W) Colour and luminance of playfield #3 (711). This is also the 5th player colour register COLPM5.

53274 D01A COLBK

(W) Colour and luminance of playfield #4/border (712).

53275 D01B (W) PRIOR

(W) Priority selection register. PRIOR establishes which objects on the screen (players, missiles and playfields) will be in-front of other objects. Values here are also described at 623; $26F, the shadow register. If you set multiple bits, then conflicting priorities at the same level turn black in overlapping regions:

BIT:	3	2	1	0
DEC:	8	4	2	1
PRIORITY:				
HIGH	PF0	PF0	PM0	PM0
	PF1	PF1	PM1	PM1
	PM0	PF2	PF0	PM2
	PM1	P5/PF3	PF1	PM3
	PM2	PM0	PF2	PF0
	PM3	PM1	P5/PF3	PF1
	PF2	PM2	PM2	PF2
	P5/PF3	PM3	PM3	P5/PF3
LOW	BAK/G	BAK/G	BAK/G	BAK/G

For example; if you set bits 3 and 1, then PM0 and 1 will blackout with PF0 and 1 of the same level. This is what you could call a power cut.

(R) Reset to 15.

53276 D01C VDELAY

(W) Vertical delay register. Used to give 1 line resolution movement capability in the vertical positioning of an object when the 2 line resolution display is enabled. Setting a bit in VDELAY to 1 moves the corresponding object down 1 TV scan-line.

53277 - 53278

If DMA is enabled, then moving an object by more than 1 line is accomplished by moving bits in the memory map instead, see the PMG appendix.

BIT:	7	6	5	4	3	2	1	0
DEC:	128	64	32	16	8	4	2	1
PM#:	P3	P2	P1	P0	M3	M2	M1	M0

(R) Reset to 15.

53277 D01D GRACTL

(W) Used with DMACTL at 54272; $D400, to latch all stick and paddle triggers. Also used to turn on players and missiles. Bit use is:

BIT:	DEC:	USE:
0	1	Turn missiles on
1	2	Turn players on
2	4	Latch all trigger inputs

To revoke P/M authorization and turn off both players and missiles, POKE 53277 with 0. Once latched, triggers will give a continuous 'button pressed' status until this latch bit is cleared (set to 0).
If you've ever pressed BREAK during a game using player/missile graphics, then you'll have noticed that the players/missiles are still left on screen, and in some cases, they turn into flickering blocks. You can get rid of this junk by POKEing 0 into this location or by POKEing 559 with 34. If for some reason, it does not dissappear, then there is an active interrupt working; POKE 580,0 and hit RESET should do the trick. You can also use POKE 623,4 to prioritise playfields over players and missiles.

(R) Reset to 15.

53278 D01E HITCLR

(W) POKE with any value to clear all player/missile collision registers. It is important to clear this register often in a program which frequently checks for collisions, otherwise, old collision values may remain and confuse the program. A simple way to accomplish this is to clear the collision registers prior to every joystick check, this way, if a collision is detected, then it is due to the most recent joystick input.

(R) Reset to 15.

53279 D01F CONSOL

(W/R) Used to see if any of the silver consol keys have been pressed, although, not RESET and HELP. For Reset, see locations 10 and 11, see location 732 for the HELP key. Depending on which key you press from OPTION, SELECT or START, a value is returned to this register as shown in the table:

KEY/S PRESSED:			DEC:
OPTION	SELECT	START	
yes	yes	yes	0
yes	yes	no	1
yes	no	yes	2
yes	no	no	3
no	yes	yes	4
no	yes	no	5
no	no	yes	6
no	no	no	7; Default

CONSOL is normally 7 (no keys pressed) and is updated every stage-2 VBlank. The OPTION key is also used to disable Basic by holding it down while turning-on your Atari XL/XE. You should normally only need to hold OPTION down until the blue-screen appears, or just a couple of seconds.
It is possible to use the consol speaker to generate different sounds, try the following program:

```
10 DATA 104,162,255,169,255,141,31,208
20 DATA 169,0,160,240,136,208,253
30 DATA 141,31,208,160,240,136,208,253
40 DATA 202,208,233,96
50 FOR I=0 TO 26
60 READ D:POKE 1536+I,D:NEXT I
70 X=USR(1536)
```

To change the tone, you POKE 1547 and 1555 with a higher or lower value (both are presently 240). To change the tone duration, you POKE 1538 with a lower value (it's set to 255). Apart from changing the tone, you can also create some wicked sways in the notes, for example, try POKE 1546,164 - POKE 1547,20 and POKE 1555,80. You could also put the program in an endless loop with a GOTO 70 statement for a fuller sounding affect.

53280-53503 D020-D0FF REPEAT MEMORY

These locations are repeats of locations 53248 - 53279 ($D000 - $D01F). Mapping states that you cannot use these locations, but in fact, you can. Whether or not, they hold any other secrets, I don't know for sure. They appear to be exactly 'timed' repetitions of the earlier locations.

53504 - 54015

53504-53759 D100-D1FF ....

Unused by the OS, this area is switched out when an external device connected to the expansion bus is selected and the device memory is switched in. The situation is reversed when the device I/O is completed:

53504-53758	D100-D1FE	Device registers
53504	D100	Hardware get and put register (HWGET/HWPUT) data from the device on the bus is stored here
53505	D101	Hardware RESET and status register (HWRSET = write; this resets the get/put register HWSTAT for read)
53759	D1FF	Hardware select register, shadow byte is 583 ($247). Bit-0 is device-0, Bit-1 is device-1 etc. Writing to this byte deselects the FP ROM and selects device ROM (try looking at it and subsequent locations with MAC/65's DDT or a similar tool while altering $D1FF)

This area is normally $FF'd (completely comprised of 255's) and is not alterable at all.

53760-54015 D200-D2FF POKEY

POKEY is a digital I/O chip that controls the audio frequency and control registers; frequency dividers, poly noise counters, pot (paddle) controllers, the random number generator, keyboard scan, serial port I/O and the IRQ interrupts.
The AUDF# (audio frequency) locations are used for the pitch for the corresponding sound channels, while the AUDC# (audio control) locations are the volume and distortion values for those same channels.

Frequency values range from 0 - 255, although the value is increased by 1 by the computer to range from 1 to 256. Note, that the sum of the volumes should not exceed 32, since volume is controlled by the least 4-bits, volumes also distort if the sum of all the channels output volumes to the speaker is greater than 32 because POKEY only controls the speaker cone at 16 different positions from resting position (inclusive). The range is set from 0 - 15. You can POKE it with 16 (Volume only, Bit-4) and a sound will be forced out (the speaker cone gets pushed out to its furthest position causing a slight 'pop' sound). The highest 3-bits are used for distortion; 192 gives pure-digital tone, other values range from 32 - 192 in steps of 16.

53760 - 53761

The AUDF# registers are also used as the POKEY hardware timers. These are generally used for counting intervals less than 1 VBlank (see the explanation in VTIMR4 at 532,533). For longer intervals, use the software timers.
VBI's and DLI's occasionally have painful results if they conflict with the hardware interrupts. These results can occur if your DLI's are too long, the 6502 interrupt flag is not set or a STA WSYNC occurs at an awkward time.

POT values are for paddles, ranging from 0 - 228, increasing as the knob is turned counterclockwise, but values less than 40 and greater than 200 represent an area on either edge of the TV screen that may not be visible on all TV sets or monitors.

53760 D200 (W) AUDF1
(R) POT0

(W) Audio channel-1 frequency. This is actually a number (N) used in a "divide by N circuit"; which, for every N pulses coming in (as set by the POKEY clock), 1 pulse goes out. As N gets larger, output pulses will decrease and the sound produced will be of lower tone. As N gets lower, the reverse happens.
Try POKE 53761,168 and POKE 53760,200. This is the same as the Basic SOUND statement: SOUND 0,200,10,8.

(R) POT (paddle) 0 (624); POT is short for potentiometer (variable resister). Turning the paddle knob clockwise results in decreasing pot values. When reading paddles in machine-language, the POT values are only valid 228 scan lines after the POTGO command, or after ALLPOT changes (see 53768 and 53771).
POT registers continually count down to 0, decrementing every scan line. They are reset to 228 when they reach 0 or by the values read from the shadow registers. This makes them useful as system timers too. COMPUTE!, February 1982 shows this use.
The POTGO sequence (see 53771) resets the POT registers to 0, then reads them 228 scan lines later. For the fast pot-scan, Bit-2 of SKCTL at 53775 must be set.

53761 D201 (W) AUDC1
(R) POT1

(W) Audio channel-1 control. Each AUDF register has an associated control register which sets volume and distortion levels. The bit use is:

BIT:	7	6	5	4	3	2	1	0
DEC:	128	64	32	16	8	4	2	1

53762 - 53763

Distortion (noise)	Volume only	Volume level	
0 0 0	0	0 0 0 0	Lowest
0 0 1		0 0 0 1	
etc.to:		etc.to:	
1 1 1	1 Forced output	1 1 1 1	Highest

The values for the distortion bits are as follows. The 1st process is to divide the clock value by the frequency, then mask the output using the polys in the order below, and finally, the result is divided by 2. The various sound affects are also given, they vary depending on low or high frequencies:

BIT: 7 6 5	BIT-POLYS:	DISTORTION Low freq. to high:
0 0 0	5 then 17	geiger-counter to steam
0 0 1	5	machine-gun to power-transformer
0 1 0	5 then 4	calm-fire to car-engine
0 1 1	5	machine-gun to power-transformer
1 0 0	17	crashing-building to waterfall
1 0 1	none	pure tones
1 1 0	4	airplane to electric-razor
1 1 1	none	pure tones

In general, the tones become more regular (a recognizable droning becomes apparent) with fewer and lower polys masking the output. This is all the more obvious at low frequencies. POKE with 160 or 224 plus the volume for pure tones. De Re Atari gives a good explanation of sound.

(R) POT-1 register (625).

53762 D202 (W) AUDF2
(R) POT2

(W) Audio channel-2 frequency. Also used with AUDF3 to store the 19200 baud rate for SIO.

(R) Pot-2 (626).

53763 D203 (W) AUDC2
(R) POT3

(W) Audio channel-2 control.
(R) Pot-3 (627).

53764 - 53768

53764	D204	(W) AUDF3 (R) POT4

(W) Audio channel-3 frequency. Used with AUDF2 and AUDF4 to store the 600 baud rate for SIO.

(R) Pot-4. Since there are no more than 4 Paddles on the XL/XE series, POT's 4 - 7 are repeats of POT's 0 - 3.

53765	D205	(W) AUDC3 (R) POT5

(W) Audio channel-3 control.
(R) POT5; repeat of POT-1.

AUD#2 and 3 can be altered in a program to point to AUD#6 and 7 to have stereo output if you have made the stereo upgrade in your Atari. See the STEREO appendix for full information. It's well worth the modification.

53766	D206	(W) AUDF4 (R) POT6

(W) Audio channel-4 frequency.
(R) POT6; repeat of POT-2.

53767	D207	(W) AUDC4 (R) POT7

(W) Audio channel-4 control.
(R) POT7; repeat of POT-3.

53768	D208	(W) AUDCTL (R) ALLPOT

(W) Audio control. To properly initialize the POKEY sound capabilities, POKE AUDCTL with 0 and POKE 53775,3. These 2 POKEs are the equivalent of Basics SOUND 0,0,0,0. AUDCTL is the option byte which affects all sound channels. This bit asignment is:

BIT:	DEC:	DESCRIPTION:
7	128	Makes the 17-Bit poly into a 9-Bit poly (see below)
6	64	Clock channel-1 with 1.79MHz
5	32	Clock channel-3 with 1.79MHz
4	16	Join channels 1 and 2 (16-Bit)
3	8	Join channels 3 and 4 (16-Bit)
2	4	Insert High-pass filter into channel-1, clocked by channel-3
1	2	Insert High-pass filter into channel-2, clocked by channel-4
0	1	Switch main clock-base from 64KHz to 15KHz

53768 cont.

Poly (polynomial) counters are used as a source of random pulses for noise generation. There are 3 polys; 4-Bits, 5-Bits and 17-Bits long. The shorter polys create more repetitive sound patterns, while the longer poly has no apparent repetition. Therefore, setting Bit-7 above, making the 17-Bit poly into a 9-Bit poly will make the pattern in the distortion more evident. You select which polys you wish by setting the high 3-Bits in the AUDC# registers. The 17-Bit poly is also used to generate the random-number at location 53770; $D20A.

The clock-bits allow you to speed-up or slow-down the clock-timers, respectively, making higher or lower frequency ranges possible. Setting the channels to 1.79MHz will produce a very much higher sound, the 64KHz clock is far lower, while the 15KHz clock is the lowest. The main-clock is also used when setting the frequency for the hardware-timers.
Bits 3 and 4 (decimal 8 and 16) allow you to combine channels 1 and 2, or 3 and 4 to obtain much higher or lower frequencies within a 9-octave range, instead of the usual 5. Try the following example:

```
10 POKE 53768,80
20 POKE 53761,160:POKE 53763,168
30 POKE 20,0:POKE 19,0
40 POKE 53760,PEEK(20):POKE 53762,PEEK(19):GOTO 40
```

If you have a set of paddles, then you can use them to alter the frequency, just substitute line 40 for:

```
40 POKE 53760,PADDLE(0):POKE 53762,PADDLE(1):GOTO 40
```

Or, if you only have a joystick:

```
40 S=STICK(0)
50 F=(S=7 AND (NOT X-255))-(S=11 AND X)
60 C=(S=13 AND(NOT Y-255))-(S=14 AND Y)
70 X=X+F:Y=Y+C
80 POKE 53760,X:POKE 53762,Y:GOTO 40
```

Where the left paddle (stick left and right) is for fine adjustment and the right one (stick up and down) is for coarse adjustment.
High-pass filters only allow frequencies higher than the clock value to pass through, which is very handy for creating dynamic sounds with no updates. This method is also handy for making special affects:

```
10 POKE 53768,4
20 POKE 53761,168:POKE 53765,168
30 POKE 53760,254:POKE 53764,127
40 GOTO 40
```

Break the program and do a POKE 53768,5. Now, try POKE 53764,255. The possibilities are wide and varied.
There is a very good sound article in De Re Atari, and the Hardware manual is worth seeing.

The actual frequencies described pre-leaf are all rounded off; 64KHz is actually 63.921 KHz, 15KHz is really 15.6999 KHz and 1.79MHz is 1.78979MHz. You can correctly calculate the POKEY interrupt frequency with:

INTFREQ = clock-freq / (2 * (1 + AUDF# value))

See COMPUTE!'s 3rd book of Atari, or the VOLUME-BIT appendix in this book.

(R) ALLPOT; 8-line POT port state; reads all 8 POTs together. The lower 4 bits represent the paddles of the same number, the higher 4-bits are repeats of the lower 4. Bits are set to 1 if valid (paddle in use). ALLPOT is used with the POTGO command at 53771; $D20B.

53769	D209	(W) STIMER (R) KBCODE

(W) Start the POKEY timers (the AUDF registers). You POKE any non-zero value here to load and start the timers; the value itself isn't used in the calculations. This resets all of the audio frequency dividers to their AUDF values. If enabled by IRQEN below, these AUDF registers generate timer interrupts when they count down from the number you POKEd there to 0. The vectors for the AUDF1, AUDF2 and AUDF4 timer interrupts are located between 528 and 533; $210 - $215.

(R) KBCODE holds the keyboard code which is then loaded into the shadow register 764; $2FC when a key is hit. Usually read in response to the keyboard interrupt. Compares the value with that in CH1 at 754, and if both the values are the same, then the new code is accepted only if a suitable key debounce delay has transpired. The routines which test to see if the keycode will be accepted start at 64537; $FC19.

53770	D20A	(W) SKREST (R) RANDOM

(W) Reset bits 5 - 7 of the serial port status register at 53775 to 1.

53771 - 53773

(R) RANDOM; When this location is read, it acts as a random number generator. It reads the high order 8-Bits of the 17-Bit polynomial counter (9-Bit if Bit-7 of AUDCTL is set) for the value of the number. You can PEEK this register in a program to generate a random integer between 0 - 255. If you want a random number between 0 - 65535, then you can use: ? PEEK(53786)*256+PEEK(53770). The Basic equivalent uses the INT and RND statements as: ? INT(RND(0)*65536).

53771 D20B POTGO

(W) Start the POT scan sequence. You must read your POT values 1st and then start the scan sequence, since POTGO resets the POT registers to 0. Written by the stage-2 VBlank.

53772 D20C ....

Unused and unalterable; set to 255. Most of the hardwares unused memory is set to 255.

53773 D20D (W) SEROUT
(R) SERIN

(W) Serial port data output. Usually written to in the event of a serial data out interrupt. Writes to the 8-Bit (1-byte) parallel holding register that is transferred to the serial shift register when a full byte of data has been transmitted. This 'holding' register is used to contain the bits to be transmitted 1 at a time (serially) as a 1-byte unit before transmission.

(R) Serial port input. Reads the 1-byte parallel holding register that is loaded when a full byte of serial input data has been received. As above, this holding register is used to hold the bits as they are coming in 1 at a time until a full byte has passed. This byte is then taken by the computer for processing. Also used to verify the checksum value at location 49; $31.

The serial bus is the port on the Atari into which you plug the cassette or disk cable. For the pin descriptions, see the PINOUTS appendice.

53774 D20E (W) IRQEN
(R) IRQST

(W) Interrupt request enable. POKE with 0 to turn off all interrupts, or with the appropriate values to enable the desired interrupt. Bit use is:

BIT:	DEC:	INTERRUPT:	VECTOR:
0	1	Timer-1 enable	VTIMR1 528; $210
1	2	Timer-2 "	VTIMR2 530; $212
2	4	Timer-4 "	VTIMR4 532; $214
3	8	Serial O/P transmitted	VSEROC 526; $20E
4	16	Serial O/P data needed	VSEROR 524; $20C
5	32	Serial I/P data ready	VSERIN 522; $20A
6	64	Other-key enable	VKEYBD 520; $208
7	128	Break-key "	BRKKY 566; $236

When a bit is set or cleared, that interrupt is enabled or disabled. For example, if you enable the break key interrupt, the vector BRKKY is only taken when the break key is pressed. When you set the timer interrupts, then their associated timers are decremented, and when they reach 0, the Atari vectors through its associated interrupt vector. These timer bits are not set on power-up, so should be set by the user before enabling the processor IRQ.
There is 1 other interrupt, processed by PIA, generated over the serial bus proceed and interrupt lines, set by PACTL at 54018; $D302. See this register for further details.

(R) IRQST; Interrupt request status. Bit functions are the same as IRQEN except that they register the interrupt request status; ie. timers are read as 1 when they count down and reach 0, rather than the enable bit when it is set. IRQST is used to determine the cause of the interrupt request with IRQEN and PACTL described above.
All IRQ interrupts are normally vectored through 65534; $FFFE to the IRQ service routine at 49196; $C02C which determines the cause of the interrupt. The IRQ global RAM vector VIMIRQ at 534; $216 ordinarily points to the IRQ processor at 49200; $C030. This processor routine then examines 53774; $D20E and the PIA register 54018 to determine the cause of the interrupt. Once determined, the routine vectors through 1 of the IRQ RAM vectors on Page-3.

53775 D20F (W) SKCTL
(R) SKSTAT

(W) Serial port control. Holds the value 255 if no key is pressed, 251 for most other keys, 247 for the shift key. This also stores the help key detection, the help key, when read here, also has the auto-repeat feature. POKE with 3 to stop the occasional noise from the cassette unit after I/O to bring POKEY out of 2-tone mode. Shadow register is 562. See SKSTAT also, for bit use.

(R) SKSTAT; reads the serial port status. It also returns values governed by a signal on the digital track of the cassette tape. You can generate certain values using the SOUND command and a PEEK to SKSTAT: SOUND 0,5,10,15 returns a value of 255 here, but 127 on occasion. SOUND 0,8,10,3 returns a value of 239. This is handy for adding a voice track to your Atari tapes. You use the left channel for your voice track and the right channel for the tones you want to use as cueing marks. You can use your TV speaker to generate the tones by placing a microphone directly in front of it. The computer will register these tones in this register when it encounters them in a later cassette load. See COMPUTE!, July 1981 for some other ways of doing this. Remember, you can turn the cassette off by POKE 54018,60 and back on with a value of 52.
SKCTL bits are normally 0 and perform the functions below when set. The status when used as SKSTAT (R) are also listed here, below the (W) function:

BIT:	DEC:	MODE/FUNCTION:
0	1	(W) Keyboard debounce circuit enable
1	2	(W) Keyboard scanning circuit "
		(R) Serial I/P shift register busy
2	4	(W) Fast pot-scan enable; the pot-scan counter completes its sequence in 2-TV scan-lines instead of 1-frame time (228 scan-lines) not as accurate as the normal pot-scan
		(R) Last key is still pressed
3	8	(W) Serial O/P is transmitted as a 2-tone signal rather than logic on/off. POKEY 2-tone mode
		(R) Shift key is pressed
4,5,6		(W) Serial port mode control, see next page
4	16	(R) Audio I/P; data can be read here ignoring the shift register
5	32	(R) Serial data I/P over-run, see next page
6	64	(R) Keyboard over-run, see next page
7	128	(W) Force break; serial O/P to 0
		(R) Serial data I/P frame error caused by missing or extra bits, see next page

Bit-2 is 1st set to 0 to reset POT registers to 0 (dumping the capacitors to change the POT registers). Then Bit-2 is set to 1 to enable the fast scan. This is not as accurate as the normal scan. This Bit must be reset to 0 to enable normal scan-mode; otherwise, the capacitors will never dump. This Bit has also been used in a small machine-language routine in the Atari' Graphics demo-disk. With a few other locations, it can be used to achieve full colour GTIA photograph displays as it is done on this demo-disk. You should be able to get hold of the disk at various PD-libraries.

Write (W) bits 4, 5 and 6 are used to set the bi-directional clock-lines so that you can either receive external clock-data or provide clock-data to external devices; see Hardware manual p.II.27. There are 2 pins on the serial port for Clock-IN and Clock-OUT. See the OS Users manual p.146. The whole of section-9 describes this area. Bits 5 and 6 are listed the other way round in Mapping and most other manuals, and in fact they are wrong. The bit assignment in this book is the correct one; see Page-6 mailbag, issue 60. Bits 5 - 7 (latches) can also be reset to 1 by using SKRES at 53770; $D20A.

53776-54015 D210-D2FF REPEAT-MEMORY

These locations are repeats of locations 53760 - 53775, although, you will find that many of them have different default values when PEEKing them. Enter Basic and try this program:

```
10 DL=PEEK(560)+256*PEEK(561)
20 POKE DL+4,0:POKE DL+5,210
```

You should see all the (R) locations. The Random number is different for all its repeated locations, and if you press a key and hold it down, particular groups of locations flicker. These groups are the same locations, only repeats, but they have different default values.

53776-53791 D210-D21F POKEY2

If you've got the stereo sound upgrade in your Atari, then these locations are the new AUD# registers etc., see the STEREO appendix for complete details.

54016-54271 D300-D3FF PIA: 6520

The peripheral interface adapter (PIA) integrated circuit is a special microprocessor used to control the Atari ports, controller jacks 1 and 2. Ports can be used for both input and output simultaneously or alternately. The ports can be used, and are used for a wide variety of purposes on the XL/XE series; from a thermostat control to a video-camera input or speech/music digitizing. These ports are a major resource for external and internal control and expansion.
PIA also processes ROM configurations at 54017; $D301, and 2 of the IRQ interrupts: VPRCED and VINTER, vectored at locations 514 - 517; $202 - $205. These interrupts are not used by the OS, but do provide greater control over external devices.

54016 D300 PORTA

(W/R) Reads or writes data from controller jacks 1 or 2 if Bit-2 of PACTL is set to 1. Writes to 'direction-control' if Bit-2 of PACTL is 0.
This register also controls the direction of data-flow to the port, if the controller register PACTL has bits 4 and 5 set (POKEd with 48), then, if the bits here read 0, it is in input (R) mode; if they read 1, then it is in output (W) mode. A 0 POKEd here makes all bits input, a 255 makes all bits output. Bits 0 - 3 address pins 1 - 4 on jack 1, while bits 4 - 7 address pins 1 - 4 on jack 2. POKE 54018 with 52 to make this register into a data register again. Shadow registers are 632; $278 for STICK(0), 633; $279 for STICK(1) and 636 - 639; $27C - $27F for PTRIG0-3.

```
Bits used as a data-register:
7  6  5  4   3  2  1  0
--Jack-0--   --Jack-1--
-STICK(0)-   -STICK(1)-

Forward:    Bits 0 and 4 = 1
Backward:      "  1  "  5 = 1
Left:          "  2  "  6 = 1
Right:         "  3  "  7 = 1
Neutral:   All 4 jack bits =1
```

PORTA is also used to test if the paddle 0 - 3 triggers (PTRIG) have been pressed, using these bits:

BIT:	7	6	5	4	3	2	1	0
PTRIG:	3	2	-	-	1	0	-	-

54017

The PORTA register is also used in the keyboard controller (used with a keypad) operation where:

BIT:	7	6	5	4	3	2	1	0
ROW:	4	3	2	TOP	4	3	2	TOP
JACK:	......2......				......1......			

Columns for the keyboard operation are read through the POT (PADDL) and TRIG registers. See micro, May 1982 and the Hardware manual for more information on jacks and ports.

54017 D301 PORTB

(W/R) Since the XL/XE series no longer have a PORTB (on the old Atari', this was for ports 3 and 4, giving 4 joysticks! Why ever did Atari drop the 4 ports?), this register is used for 1200XL LED control, 130XE bank switching as well as XL/XE memory management in particular.
You can disable the ROM between 49152 - 53247; $C000 - $CFFF and 55296 - 65535; $D800 - $FFFF by clearing Bit-0 to 0. These 2 ROM areas are switched-out and RAM is switched-in. Note, that the Hardware memory between $D000 - $D7FF remains intact and is not switchable. When you do switch the ROM-out, unless another OS has been provided, the system will crash at the next interrupt (a maximum of 1/50th second later), for this reason, if you do switch this ROM out for another ROM, you should disable all NMI and IRQ interrupts. Do this with:

```
LDA #$0
STA  $D40E      ;disable NMI's
SEI             ;disable IRQ's
```

Bit-1 controls Basic; If 0, Basic is enabled, if 1, then it is disabled and the 8K region between $A000 - $BFFF is available as RAM. If you disable Basic from within a Basic program using any Basic keyword, then the system will lock-up.

Bits 2 and 3 control the 1200Xl LED'; 0 means on and 1 means off. LED-1 is the keyboard enable/disable; LED-2 is the character-set selected. In the 130XE, these bits are used for bank switching 16K blocks of RAM. You can use this extra memory as video memory or program/data memory. See the 130XE BANK-SWITCHING appendix.

Bits 4 - 6 are unused in the XL' and 65XE. In the 130XE, bits 4 and 5 are used to enable bank switching.

Bit 7 controls the RAM region 20480 - 22527; $5000 - $57FF which is normally enabled (set to 1). When this bit is cleared to 0, the OS-ROM in this area is enabled and access provided to the Self-test code moved from 53248 - 55295; $D000 - $D7FF (under the Hardware memory).

Try this: POKE 54017,PEEK(54017)-128 to enable the Self-test ROM. Now type X=USR(20480). The Self-test screen appears. The RAM in this area is restored on Reset/warmstart or cold-start. Of course, you really only have to type BYE in Basic to access the Self-test routine, but when you enter the Self-test this way, the system also sets the COLDSTart flag at location 580 to 255, so pressing Reset actually coldstarts the system.
Here's a program from Joe Miller of Koala technologies which copies the OS-ROM in 2 portions (skipping the $D000 - $D7FF block) into RAM, disables the ROM, and then moves the OS back to its original address area, but giving a RAM-OS:

```
100 REM RAMROM - Install RAM-based
110 REM OS in an XL/XE computer
120 REM by Joe Miller
130 REM March 23rd, 1985
180 REM
190 ? CHR$(125)
200 ? "Moving OS-ROM into RAM...";
205 RESTORE 300
210 FOR I=1536 TO 1635
220 READ B:POKE I,B:NEXT I
230 X=USR(1536)
240 ? "OS moved back to original"
250 ? "area, but is now RAM-OS."
260 ? "Press RETURN for a proof-test";
270 ? :?
280 ? "POKE 57344,1
290 POSITION 2,5
300 DATA 169,0,133,203,133,205,169,192
310 DATA 133,204,169,64,,133,206,160,0
320 DATA 177,203,145,205,200,208,249
330 DATA 230,206,230,204,240,12,165,204
340 DATA 165,204,201,208,208,237
350 DATA 169,216,133,204,208,231,8,120
360 DATA 173,14,212,72,169,0,141,14,212
370 DATA 173,1,211,41,254,141,1,211
380 DATA 169,192,133,206,169,64,133,204
390 DATA 177,203,145,205,200,208,249
400 DATA 230,204,230,206,240,12,165,206
410 DATA 201,208,208,237,169,216,133,206
420 DATA 208,231,104,141,14,212,40,104,96
```

You can make this program into an AUTORUN.SYS file by changing the loop at line 1610 to: FOR I=1536 to 1634, removing the last occurrence of the number 104 in line 420 and deleting the USR call at line 230. Re-run the program, and then goto DOS and use the Binary-save option K, and type:

Filename.ext,0600,0662,0600

This way, every time you boot this disk up, the ROM-OS will become a RAM-OS occupying the same area of memory it usually does. Here's the Source listing:

```
;Move XL OS ROM into RAM
;
;RAMROM-Installs the XL ROM-based OS
;in RAM at the same address space. This
;is useful for making small patches to
;the OS or for experimenting with new-
;design concepts such as; multi-tasking
;or window management etc..
;
;by Joe Miller
;
;This version is configured as an
;AUTORUN.SYS file
;
SOURCE EQU $CB           ;Page-0 useage
DEST   EQU SOURCE+2
START  EQU $0600         ;start addr
OSROM  EQU $C000         ;OS-ROM start
OSRAM  EQU $4000         ;ROM dest addr
NMEIN  EQU $D40E         ;NMI register
PORTB  EQU $D301         ;Memory CTL
;
       ORG START
       LDA #low OSROM
       STA SOURCE
       STA DEST          ;init copy addr
       LDA #high OSROM
       STA DEST+1
       LDY #0
                         ;repeat
PASS1  LDA (SOURCE),Y    ;copy ROM - RAM
       STA (DEST),Y
       INY
       BNE PASS1
       INC DEST+1
       INC SOURCE+1
       BEQ SWAP          ;if done
```

```
        LDA SOURCE+1
        CMP #$D0
        BNE PASS1          ;skip D-block
        LDA #$D8
        STA SOURCE+1
        BNE PASS1          ;until SOURCE=$0000
SWAP    PHP                ;save proc.stat
        SEI                ;disable IRQ'
        LDA NMEIN
        PHA                ;save NMEIN
        LDA #$0
        STA NMEIN          ;disable NMI'
        LDA PORTB
        AND #$FE           ;disable ROM'
        STA PORTB          ;BASIC unchanged
        LDA #high OSROM
        STA DEST+1         ;setup block copy
        LDA #high OSRAM
        STA SOURCE+1
                           ;repeat
PASS2   LDA (SOURCE),Y     ;return OS
        STA (DEST),Y
        INY
        BNE PASS2
        INC SOURCE+1       ;next page
        INC DEST+1
        BEQ ENABLE         ;when complete
        LDA DEST+1
        CMP #$D0
        BNE PASS2          ;skip D-block
        LDA #$D8
        STA DEST+1
        BNE PASS2          ;until DEST=$0000
ENABLE  PLA
        STA NMEIN          ;enable NMI'
        PLP                ;enable IRQ'
        RTS
        END START
```

Altering the ROM-OS into a RAM-OS can be a REAL bonus, because now that the OS is RAM, you can alter anything you like; you can alter any of the 2 character-sets in the original locations, thus, saving 2K of memory, you could re-write the handlers, interrupts or any other routine you desire. For a Reset-key trap, see the PROGRAMS appendix.

As well as turning the ROM-OS into a RAM-OS, you can also switch the ROM Basic and Self-test to their RAM equivalents residing in their original locations of course. The program on the next page will perform these tasks:

54018

The program in its present form will enable the Self-test ROM, transfer it into RAM, switch the Self-test ROM into RAM and copy the Self-test package from lower RAM, back up into its original locations:

```
10 DATA 173,1,211,41,127,141,1,211
12 DATA 169,80,133,204,169,40,133,206
14 DATA 169,0,133,203,133,205,160,0
16 DATA 177,203,145,205,200,208,249
18 DATA 230,204,230,206,165,204,201,88,208,239
20 DATA 173,1,211,9,128,141,1,211
22 DATA 169,40,133,204,169,80,133,206
24 DATA 169,0,133,203,133,205,160,0
26 DATA 177,203,145,205,200,208,249
28 DATA 230,204,230,206,165,206,201,88,208,239
30 DATA 104,96,-1
40 I=0
50 READ D:IF D+1 THEN POKE 1536+I,D:I=I+1:GOTO 50
60 X=USR(1536)
```

If you want to do the same thing with the Basic ROM, then make the following changes:

1. Change 41,127 in line 10, to 41,253
2. " 80 in line 12, to 160
3. " 88 in line 18, to 192
4. " 9,128 in line 20, to 9,2
5. " 80 in line 22, to 160
6. Lastly, change 88 in line 28, to 192

With the Basic turned to RAM, try the following POKEs:

```
POKE 42223,ASC("H")
POKE 42224,ASC("E")
```

What you've actually done with these 2 POKEs, is to have altered a Basic keyword. The keyword was TRAP, but it is now called HEAP. If you don't believe me, type: TRAP 40000. You'll get an error because Basic doesn't understand the word TRAP anymore, it now thinks it's called HEAP. Type: HEAP 40000. All is taken fine.
Instead of altering the keyword names themselves, you can alter the tasks performed by the keyword. See the ALTERING BASIC appendix.

54018 D302 PACTL

(W/R) PORTA controller. POKE with 52 to turn the cassette motor on, and with 60 to turn it back off. You can play a music tape through the TV speaker using this method, handy when programming in the early hours of the morning without waking the whole house up with your getto-blaster!

PACTL can be used for other external applications by the user, Bit use is:

BIT:	DEC:	FUNCTION:
7	128	Peripheral-A interrupt (IRQ) status; only read
6	64	Zero forced; unalterable
5	32	Set to 1
4	16	Set to 1
3	8	Peripheral motor control line; write only
2	4	Set to 1 for PORTA addressing, direction control register when 0; write only
1	2	Set to 0; this is alterable
0	1	Peripheral-A interrupt (IRQ) enable. 1 = enable Set by the OS, but available for use; write only

54019 D303 PBCTL

(W/R) Originally for the PORTB controller, but since there is no PORTB anymore, this register is unused, however, it still has Bit-6 forced to 0. You can use this as an extra RAM register, so long as whatever value you place here does not require Bit-6 to be set. Hence, you cannot store decimal values: 64 - 127, and 192 - 255 here.
Get hold of COMPUTE! February 1981 for an article showing you how to use the joystick ports as a printer port.

There is 1 other point to note about this register. All sources say that this is now unused, but in fact, SIO actually stores a value here. See Appendix E6, address $E9CB.

54020-54271 D304-D3FF REPEAT-MEMORY

These locations are repeats of 54016 - 54019; $D300 - $D303.

54272-54783 D400-D5FF ANTIC

ANTIC is a special, seperate micro-processor in the Atari to control GTIA, the screen-display and other screen related functions including the NMI interrupts. It uses its own 'instruction-set', called the 'Display-List' (DL), which tells ANTIC where to find the screen data in RAM and how to display it. ANTIC also uses an internal 4-bit counter called the Delta-counter (DCTR) to control the vertical dimensions of each block.

54272 D400 DMACTL

(W) Direct Memory Access (DMA) control. This is used to define 1 or 2 line resolution for PMG's as well as to turn them onto the screen. Values should normally be POKEd into the shadow register 559; $22F, and the bits are fully described there (Page-45).
For the experienced machine-language user, you might be interested to know that you can cause some queer affects by successively altering this register whilst retaining the normally enabled status of the VBlanks; for instance:

```
10 DATA 169,5,141,0,212,76,0,6
20 FOR I=0 TO 7
30 READ D:POKE 1536+I,D:NEXT I
40 X=USR(1536)
```

If nothing happens at 1st, just press Reset and re-run the program, or alter the value 5 loaded into the Accumulator until something does happen. You'll notice the screen turns into chaos, but there are several important things that you should note: 1stly, the 2 very-small borders at the very roof and the very floor of the TV tube do not exist. Another, more important point is that the frame is twisted. This, I hope, will give you some insight as to bending screen images without italicising them in the memory. You can have a lot of fun with this technique.

54273 D401 CHACTL

(W) Character mode control. See its shadow register 755 for values. Only the least 3-bits are active in this register, higher bits simply duplicate the lower bits. With this register, you can affect any text when inversed, or turn all text upside down. Inverse alterations also affect the cursor, because the cursor is only an inversed 'space' character anyway.

54274,5 D402,3 DLISTL/H

Display list pointer. Tells the OS the start address of the Display List (DL), which distinguishes the screen mode(s) and RAM to display. See SDLST at 560 and 561 for full details.

54276 D404 HSCROL

(W) Horizontal fine-scroll offset. HSCROL is the Hardwares horizontal fine-scroll register, which can offset the DM up to a maximum of 16 colour-clocks (4 Graphics 0 bytes) from its LMS origin. See SDLST at 560 and 561 for information on LMS. Controlled by Bit-4 of the DL pointed to by SDLST. When you scroll memory horizontally, you must re-calculate it, similar to the way shown in the following program:

```
10 FOR I=0 TO 33
20 READ D:POKE 1536+I,D:NEXT I
30 DATA 112,66,0,255,112,66,64,156,2,2
40 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
50 DATA 2,2,2,2,2,41,0,6
60 POKE 560,0:POKE 561,6
70 FOR K=0 TO 255 STEP .02
80 POKE 1538,K:NEXT K
```

You can see that each byte of the extra line at the top of the screen moves in chunks (coarsely). To enable the fine-scroll register, then add a value of 16 to the mode-line, in this case, change the 1st 66 in line-30 to 82. Now try taking out lines 70 and 80, and adding the follwowing lines:

```
70 FOR I=0 TO 15 STEP .5
80 POKE 54276,I:NEXT I
85 FOR I=15 TO 0 STEP -.5
90 POKE 54276,I:NEXT I
95 GOTO 70
```

You'll see the fine-scrolling in action. If you want to successively move through the memory, then you should fine-scroll a whole byte, and then, simultaneously, restore the HSCROL value and update the LMS display byte/s. You could do this in Basic, but it is flicky. This is because HSCROL is not reset in the same frame that the LMS DM byte is restored; you may think if you executed the Basic line:

```
XX POKE 1538,PEEK(1538)+1:POKE 54276,3
```

It is simultaneous, but, you should realise that each 1 of these POKEs takes approx. half of a frame to decode and process. For this reason, you'll see that Basic is too slow, ie; take out lines 70 - 95 of the previous program, and add:

```
70 FOR J=0 TO 255
75 FOR I=3 TO 0 STEP -.2
80 POKE 54276,I:NEXT I
85 POKE 54276,3:POKE 1538,J
90 NEXT J
```

To remove the flicker, the 2 POKEs on line 85 need to execute in good timing, so you really need to execute a machine-code routine for this, add the following to the program:

```
62 FOR Q=0 TO 11
64 READ D:POKE 1600+Q,D:NEXT Q
66 DATA 169,3,141,4,212,165,203
68 DATA 141,2,6,104,96
85 POKE 203,J:X=USR(1600)
```

Now, try the program again.

If you didn't organize the memory in the manner of this program, a wraparound affect would occur, this is where the memory from the adjacent line would wrap onto the scrolling line. You can, of course, use HSCROL to scroll the entire display horizontally, or perhaps add a DLI, and scroll the 2 halves of the screen in different directions.
The real power of any of the Atari' capabilities are normally only accessible from machine-language, or from using machine-language routines from Basic, but if you don't understand 6502 machine-code, there are quite a lot of sources that can help you: Page-6 magazine, machine-code tutor by Paul Bunn (program) which is excellent, machine-language for beginners by COMPUTE! books, also any of the assembler packages, such like Atari Assembler/Editor or MAC65. Also see my relating appendix.

54277 D405 VSCROL

(W) Vertical scroll offset. VSCROL is the Hardwares vertical fine-scroll register, which can offset the DM of up to 16 scan-lines from the LMS origin. Controlled by Bit-5 of the DL pointed to by SDLST. For an example of vertical fine-scroll of a Graphics 0 screen, try this program:

```
10 GRAPHICS 0
15 LIST
20 DL=PEEK(560)+256*PEEK(561)
30 FOR I=3 TO 28
40 IF I=4 THEN I=6
50 POKE DL+I,PEEK(DL+I)+32:NEXT I
60 FOR I=0 TO 7 STEP 0.2
70 POKE 54277,I:NEXT I
80 FOR I=7 TO 0 STEP -0.2
90 POKE 54277,I:NEXT I
95 GOTO 60
```

Should you wish to move the memory forward, then you should add 40 bytes to the origin LMS whilst restoring VSCROL. Similar to HSCROL, these functions must be achieved simultaneously. You can use the following program for this:

```
10 GRAPHICS 0:LIST
15 FOR I=0 TO 16
20 READ D:POKE 1536+I,D:NEXT I
25 DATA 104,169,0,141,5,212,165,203
30 DATA 141,0,0,165,204,141,0,0,96
35 DL=PEEK(560)+256*PEEK(561)
40 DM=PEEK(DL+4)+256*PEEK(DL+5)
45 FOR I=3 TO 27:IF I=4 THEN I=6
50 POKE DL+I,PEEK(DL+I)+32:NEXT I
55 POKE 1545,PEEK(560)+4
60 POKE 1550,PEEK(560)+5
65 POKE 1546,PEEK(561)
70 POKE 1551,PEEK(561)
75 POKE DL+28,2
80 FOR J=DM TO 255*256 STEP 40
85 JHI=INT(J/256):JLO=J-JHI*256
90 FOR Q=0 TO 7 STEP 0.2
92 POKE 54277,Q:NEXT Q
94 POKE 203,JLO:POKE 204,JHI
96 X=USR(1536)
98 NEXT J
```

There is also another fine feature that you can use the VSCROL register for, try the following program for example:

```
10 GRAPHICS 0
20 DL=PEEK(560)+256*PEEK(561)
30 POKE DL+3,2+32+64:POKE DL+7,2+32
35 POKE DL+9,2+32
40 POKE 54277,4
45 ?
50 ? "Y+fa+/2*E*F3SHMSSkYYA011QK4/H"
60 ? "sX[OTD/xC|EOsLWDOLZ7A#1T&044P"
```

You'll notice that the screen on mode lines 2 and 3 have been somewhat merged together. You can use any of the 15 Graphics modes in combination with any other, you are not limited to just Graphics 0 as I've used, this is merely to show you what you can achieve with text. Try changing the value in 54277 also. It is also possible to horizontally shift the top or the bottom part of the displaying line so that the text appears italicised.
This technique, as shown with the program, is very powerful. It not only gives you infinite character possibilities, but also saves you memory, whereas normally you'd have to reserve 1K for each additional character-set.

54278

The HSCROL and VSCROL registers offer another type of scrolling when used together. This is diagonal scrolling, for an example, just try this program:

```
10 GRAPHICS 0:POKE 752,1:S=0.2:?
15 DL=PEEK(560)+256*PEEK(561)
20 POKE DL+7,PEEK(DL+7)+32+16
25 ? "GOBBLEDEGOOK....GOBBLEDEGOOK"
30 POSITION 5,2
35 ? "DIAGONAL SCROLLING EXAMPLE";
40 POSITION 13,3
45 ? "BRING OUT THE BRANSTON"
50 FOR I=0 TO 7 STEP S
52 POKE 54276,I+8:POKE 54277,I
54 NEXT I
56 FOR I=7 TO 0 STEP -S
58 POKE 54276,I+8:POKE 54277,I
60 NEXT I
62 FOR I=7 TO 0 STEP -S
64 POKE 54276,I:POKE 54277,7-I
66 NEXT I
68 FOR I=0 TO 7 STEP S
70 POKE 54276,I:POKE 54277,7-I
72 NEXT I
74 GOTO 50
```

In affect, you can achieve scrolling in all 360 degrees, but to achieve this, you need to give HSCROL and VSCROL different step ratios.
There is one other type of scrolling that can be achieved with the XL/XE's, this is known as 3D scrolling, where the screen appears to come toward you, or away. There isn't any registers that will control the DM in this manner, so to achieve this, you need to display the 'on-screen' memory in a particular fashion. If you take as an example, Atari' Pole Position. The track appears to come toward your car. In fact, what is happening, is that the data (racing track) in the lower part of the screen is displayed larger, both vertically and horizontally, than the data in the higher part of the screen, hence, giving the affect that the higher part of the screen is more distant than the lower part. There are other ways of getting around this, and a good example would be seen in: The Great American Cross Country Road Race. This program, which I find highly addictive, actually uses PMG's for oncoming objects whilst keeps the track static!

54278 D406 ....

Unused; (R) set to 255.

54279 D407 PMBASE

(W) MSB of the Player/missile base address used to locate the graphics for your players and missiles, where the address is PMBASE*256. Player missile graphics can be quite difficult when trying to manipulate various images, as in animation or just vertical movement, because there are no Basic commands to support their use, which I find very dissapointing (slap slap, Atari).
PMG' must reside on either a 1K or 2K boundary, depending whether they use double or single line resolution, respectively. So when you set the page number for PMBASE residence, use this formula: POKE PMBASE,PAGE*4 for double, and POKE PMBASE,PAGE*8 for single line resolution.
Horizontal position, colouring and size particulars are all very simple to process, but shaping, vertical movement and/or animation is far more difficult.
See the PMG appendix, or TWAUG newsletter issue #2 for a full PMG discussion.

54280 D408 ....

Unused; (R) set to 255.

54281 D409 CHBASE

(W) Character base address; the location for the start of the current character-set, which is either the standard-set (224; $E0) at 57344; $E000, the international-set (204; $CC) at 52224; $CC00 or a user defined one which can begin at any 1K boundary within the computer, ie. the correct formula for CHBASE is POKE CHBASE,PAGE*4. Shadow location is 756. See the ROM at 57344 and 52224 also.

54282 D40A WSYNC

(W) Wait for horizontal synchronization. Allows the OS to synchronize the vertical TV display by causing the 6502 CPU to halt and restart 7 machine-cycles before the beginning of the next TV scan-line. It is also used to synchronize VBI' or DLI' with the screen display.
Here's a direct machine-language routine to show you the affect:

```
10 DATA 104,173,11,212,201,50,208,249
12 DATA 165,20,141,10,212,141,24,208
13 DATA 141,10,212,173,198,2
14 DATA 141,24,208,76,1,6,-1
20 READ D:IF D+1 THEN POKE 1536+I,D:I=I+1:GOTO 20
30 X=USR(1536)
```

54283 - 54285

You don't necessarily have to use the WSYNC register in order to achieve the timing of colour changes etc., in fact you can use the NOP command (code 234; $EA) which wastes 2 machine-cycles of time, or just JSR around the bush (6 cycles wasted, and an additional 6 cycles for RTS) until the desired amount of time is up.
Note, that the keyboard handler sets WSYNC repeatedly while generating the keyboard click on the console speaker at 53279; $D01F. To bypass this, examine the VCOUNT register on the next page and delay your interrupt processing by 1 line when no WSYNC delay has occurred. You could also, only enable the keyboard in a lower part of the screen, below the area where any WSYNC problems may occur, and ensure the keyboard is disabled above this area.

54283 D40B VCOUNT

(R) Vertical TV scan-line counter. Used to keep track of which line is currently being generated on the screen. This can be used during DLI' to change colours or Graphics modes. PEEKing here returns the line count divided by 2, ranging from 0 - 156 on PAL systems, and 0 - 131 on NTSC systems. The following program gives a colourful demonstration, which uses the VCOUNT register position as a colour:

```
10 DATA 104,173,11,212,141,24,208,76,1,6
20 FOR I=0 TO 9
30 READ D:POKE 1536+I,D:NEXT I
40 X=USR(1536)
```

The colour you see at each vertical position on the screen is the actual scan-line value where VCOUNT currently is. Since, the TV frame is refreshed every 50th second rate, it's not surprising that all the colour appears simultaneously!

54284 D40C PENH

(R) Light-Pen horizontal position (564). Holds the horizontal colour clock count when the trigger is pressed.

54285 D40D PENV

(R) Light-Pen vertical position (565). Holds the VCOUNT value when the pen trigger is being pressed. See the Hardware manual p.II-32 for a description of the light-pen operation.

54286 - 55295

54286 D40E NMIEN

(W) Non-Maskable Interrupt (NMI) enable. POKE with 192 to enable both the VBI and DLI'. When Bit-7 is set to 1, it means the DL instruction interrupt, and any DL instruction that has Bit-7 set will cause this interrupt to execute at the start of the last scan-line of the relative mode-line. When Bit-6 is 1, the Vertical Blank interrupt is enabled. Bit-5 is enable forced and unalterable, it it used for the RESET interrupt. NMIEN is set to 64; $40 by the OS IRQ code on power-up, enabling just the VBI. All NMI' are vectored through 65530; $FFFA to the NMI service routine at 49176; $C018 to determine their cause.

54287 D40F (W) NMIRES
(R) NMIST

(W) Reset for NMIST; clears the interrupt request register, resetting all of the NMI status' together.

(R) NMIST; NMI status. Holds the cause for the NMI interrupt in Bits 5, 6 and 7, corresponding to the same bits in NMIEN on the previous page. If a DLI is pending, then a JMP is made through the global RAM vector VDSLST at 512 and 513. The OS doesn't use DLI', so 512 and 513 point to an RTI instruction.
If the interrupt is not due to a DLI, then a test is made to see if the interrupt was caused by pressing the RESET key, and if so, a jump is made to 58484; $E474. If not a RESET interrupt, then the system assumes the interrupt was a VBLANK interrupt, and a jump is made through VVBLKI at 546, 547 which normally points to the stage-1 VBLANK processor. From there, it checks the CRITIC flag at 66 and, if not from a critical section, jumps through VVBLKD at 548, 549 to the VBLANK exit routine. See the VBLANKS appendix for further information on these. For IRQ', see location 53744; $D20E.

54288-54527 D410-D4FF REPEAT-MEMORY

These locations are repeats of locations 54272 - 54287; $D400 - $D40F.

54528-54783 D500-D5FF ....

Although unused memory, mapping states that if you read or write from/to any of these addresses, the cartridge control line (CCNTL) is enabled. (R) Normally cleared with 255's.

54784-55295 D600-D7FF ....

This memory appears to be unused, which like above is cleared with 255's. Not user alterable.

55296 - 65535

OPERATING SYSTEM ROM:

55296-65535 D600-FFFF OS-ROM

This 10K of memory is the OS-ROM, containing the Floating Point (FP) package, the 2 in-built character-sets, device handlers, CIO, SIO, NMI', IRQ' etc.. It differs from the older Atari's OS', so some older programs will not load on XL/XE'. In these cases, you can use the translator disks such as XL FIX, which is all Public-Domain (PD) software.

55296-57343 D800-DFFF FP-PACKAGE

This is the Floating Point mathematics package. There are also other areas used by FP in page-0 at 212 - 254, and in page-5 at 1406 - 1535. There are also trigonometric functions in the Basic ROM located from 48549 - 49145 which the the FP routines. See De Re Atari for additional information.
Here are the entry points to some of the subroutines; unless otherwise noted, they use the FP register 0 (FR0 at 212 - 217):

55296 D800 AFP

ASCII to FP conversion.

55526 D8E6 FASC

FP value to ASCII conversion.

55722 D9AA IFP

Integer to FP conversion.

55762 D9D2 FPI

FP to integer conversion.

55876 DA44 ZFR0

Clear FR0 at 212 - 217 by setting all bytes to 0.

55878 DA46 ZF1

Clear the FP number from FR1, locations 224 - 229 by setting all bytes to 0. Also called AF1 by De Re Atari.

55904 DA60 FSUB

FP subtract routine; the value in FR0 minus the value in FR1.

55910 DA66 FADD

FP addition routine; FR0 plus FR1.

56027 DADB FMUL

FP multiplication routine; FR0 timez FR1.

56104 DB28 FDIV

FP division routine; FR0 divided by FR1.

56640 DD40 PLYEVL

FP polynomial evaluation.

56713 DD89 FLD0R

Load the FP number into FR0 from the 6502 X and Y registers.

56717 DD8D FLD0P

Load the FP number into FR0 from the user routine, using FLPTR at 252.

56728 DD98 FLD1R

Load the FP number into FR1 from the 6502 X and Y registers.

56732 DD9C FLD1P

Load the FP number into FR1 from the user routine, using FLPTR at 252.

56743 - 57262

56743 DDA7 FSTOR

Store the FP number into the 6502 X and Y registers from FRO.

56747 DDAB FSTOP

Store the FP number from FRO, using FLPTR.

56758 DDB6 FMOVE

Move the FP number from FRO to FR1.

56768 DDC0 EXP

FP base e exponentiation.

56780 DDCC EXP10

FP base 10 exponentiation.

56909 DE4D P10COET

Power of 10 coefficients table.

57037 DECD LOG

FP natural logarithm.

57041 DED1 LOG10

FP base 10 logarithm.

57202 DF72 LOGCOET

Logarithm coefficients table.

57262 DFAE ARCOET

Arctangent coefficients table.

This FP area also has another purpose. It is addressable by the device when the OS switches out ROM to perform I/O on a device connected to the expansion slot (Parallel Bus Interface; PBI), whilst switching it back when finished. This means an external device cannot use FP, or any software which does (such as Basic).

On a coldstart, the OS polls for parallel devices, and if it finds 1, JMPs through 55321; $D819 to the INIT routine at 55322/55323; $D81A/$D81B which places the address of the generic parallel device handler into the handler tables with the device name.

55296 - 58367

The 1st 26 bytes of the hardware ROM vector area when the OS ROM is deselected are as follows:

BYTE:	HEX:	USE:
55296/55297	D800/D801	ROM checksum LSB/MSB; optional
55298	D802	ROM revision number; optional
55299	D803	ID number; 128 $80
55300	D804	Device type; optional
55301	D805	JMP instruction; 76 $4C
55302/55303	D806/D807	I/O vector LSB/MSB
55304	D808	JMP instruction
55305/55306	D809/D80A	Interrupt vector LSB/MSB
55307	D80B	ID number; 145 $91
55308	D80C	Device name in ASCII; optional
55309/55310	D80D/D80E	OPEN vector LSB-1/MSB
55311/55312	D80F/D810	CLOSE vector LSB-1/MSB
55313/55314	D811/D812	GET byte vector LSB-1/MSB
55315/55316	D813/D814	PUT byte vector LSB-1/MSB
55317/55318	D815/D816	STATUS vector LSB-1/MSB
55319/55320	D817/D818	XIO special vector LSB-1/MSB
55321	D819	JMP instruction
55322/55323	D81A/D81B	INIT vector LSB/MSB
55324	D81C	unused..

57344-58367 E000-E3FF CHARSET1

Standard (domestic) character-set. See location 756 for a full description of making your own character sets. The character-set here is the default upon power-up and Reset, it holds the special characters, punctuation and numbers at $E000, the capital letters begin at 57600; $E100, the special graphics characters at 57856; $E200 and the lowercase letters at 58112; $E300.

There are 1024 bytes here, each character requires 8 bytes, giving 128 characters. Inverse characters are obtained by inverting the bits of the standard character, or EORing with 128; $80, which is the value found at 694; $2B6. In Graphics modes 1 and 2, only the 1st 64 characters are accessible, so to obtain the 2nd half of this character-set in these modes, then POKE 756 with 226, 2 pages more than the default 224. This trick also applies with the international character-set found at 52224; $CC00.

Besides redesigning the character-set for use in text modes, you can use the data to POKE into any of the 'MAP' modes. Graphics 8 would be ideal. Try the programs on the next page (the 2nd program is put to good use in the STEREO appendix).

```
10 GRAPHICS 8:POKE 709,12:POKE 710,4
15 POKE 756,224
20 DL=PEEK(560)+256*PEEK(561)
25 DM=PEEK(DL+4)+256*PEEK(DL+5)
30 COLOR 1
35 SET=PEEK(756)*256
40 X=INT(RND(0)*40)
45 Y=INT(RND(0)*152)
50 CH=INT(RND(0)*128)
55 FOR I=0 TO 7
60 AREA=DM+X+Y*40+I*40
65 POKE AREA,PEEK(SET+CH*8+I):NEXT I
70 GOTO 35
```

This shows you how to use the character-set as a good means to placing text on Graphics 8. Try changing the POKE value 224 in line 15 with 204, or even other values! X and Y are the screen co-ordinates, while CH is the randomly chosen character.

Here's a better example of Graphics 8 text, which allows text to print in any of 360 degrees:

```
10 GRAPHICS 8:POKE 709,12:POKE 710,4
12 POKE 756,224
14 DIM A$(20)
16 DL=PEEK(560)+256*PEEK(561)
18 DM=PEEK(DL+4)+256*PEEK(DL+5)
20 COLOR 1
22 SET=PEEK(756)*256
24 A$="RED RED WINE"
26 DIR=41:X=5:Y=50
28 FOR J=1 TO LEN(A$)
30 C=ASC(A$(J,J))
32 NC=C
34 IF SGN(C-96)=-1 THEN NC=C-32
36 IF SGN(C-32)=-1 THEN NC=C+64
38 CH=SET+NC*8
40 FOR I=0 TO 7
42 AREA=DM+J*DIR+I*40+X+Y*40
44 POKE AREA,PEEK(CH+I)
46 NEXT I
48 NEXT J
```

Just put any comment you wish in A$ and RUN the program up. DIR is the direction of print, which can also go in steps greater than 1 if required. Try using different values in this variable such like: 1, 2, 161 and 320. Lines 32 - 36 merely convert the characters in A$, from their ASCII codes to their equivalent INTERNAL codes. To do this, Ascii codes 0 to 31 have 64 added, codes 32 - 95 have 32 subtracted and codes 96 to 127 remain the same.

58368 - 58454

58368-58447 E400-E44F HANDLER VECTORS

These are the vector tables for all the resident handlers in ROM. Each handler consists of a 15-byte table; 2 bytes each for OPEN, CLOSE, GET byte, PUT byte, STATUS and XIO special routine addresses. Following those LSB/MSB vectors, there is a JMP instruction (76; $4C) and the address of the initialization routine for that handler. The 16th byte of each handler is zeroed and unused. Below, is a table showing all the handlers vector addresses. You should also note that all the vectors, except JMP, all point to the address of the routine minus 1:

Device & Loc:	OPEN	CLOSE	GET	PUT	STATUS	XIO	JMP
E: 58368/E400	EF93	F22D	F249	F2AF	F21D	F22C	EF6E
S: 58384/E410	EF8D	F22D	F17F	F1A3	F21D	F9AE	EF6E
K: 58400/E420	F21D	F21D	F2FC	F22C	F21D	F22C	EF6E
P: 58416/E430	FEC1	FF06	FEC0	FECA	FEA2	FEC0	FE99
C: 58432/E440	FCE5	FDCE	FD79	FDB3	FDCB	FCE4	FCDB

58448-58511 E450-E48F VECTORS

Here's some more vectors, the address of these vectors remain at the same address as the old OS, but point to different locations:

58488 E450 DISKIV

Disk handler initialization vector, initialized to 50851; $C6A3.

58451 E453 DISKINV

Disk handler (interface) entry which basically checks the disk status, you can JuMP here in your own routine to perform other functions, but you'll need to reset the data direction bits in location 771; $303 before every call. Points to 50867; $C6B3.

58454 E456 CIOV

Central Input/Output (CIO) utility entry point. Initialized to 58591; $E4DF. CIO is responsible for all I/O operations and data transfers. To use CIO, you should set up your IOCB and JuMP here. Note, however, that the X register should contain the IOCB number multiplied by 16, so IOCB #0 would be 0, IOCB #1 would be 16, #2 is 32 and so on...

58457 - 58460

Once CIO is initiated, the appropriate IOCB information is passed to the Device Control Block (DCB), this then calls up SIO (below) to control the actual peripherals. CIO treats all I/O in this same manner, device independant.

You jump here to use the handler routines in the OS ROM. Basic itself doesn't support these routines (buffer I/O), that's why its device I/O operations are slower, however, with a short machine-code routine you can use this I/O method in your Basic programs. All you'll need to do is OPEN your device/file on your selected channel, set the appropriate values in the OPENed IOCB channel (locations 832 - 959) and then execute the following machine-code routine:

```
PLA, PLA, PLA, TAX, JMP $E456
104, 104, 104, 170,  76,  86, 228
$68, $68, $68, $AA, $4C, $56, $E4
```

or even X=USR(ADR("hhh*LVd")). Note; the "*" and "d" characters should be inversed.

58457 E459 SIOV

Serial Input/Output (SIO) utility entry point. Initialized to 51507; $C933. SIO drives the serial bus and the peripherals connected to it. When a request is placed in the Device Control Block (DCB), either by a device handler or by the user, SIO takes control and uses the data in the DCB to perform the operation desired. CIO is reponsible for the packaging of the data transfers before the actual transision, which is accomplished by SIO. When CIO utilizes SIO, it does so many times to accomplish the task asked of it. The DCB is locations 768 - 779; $300 - $30B.
The SIO routines peripheral poll is achieved by firstly sending a command frame which is consisted of 5 bytes (locations 570 - 574); the device ID code, command byte, 2 aux bytes for device-specific information and a checksum byte which is the sum of the 1st 4 bytes. If the device polled acknowledges and responds to the command frame, it is followed by, if necessary, a data frame of fixed length, depending upon the device; cassette record, disk sector etc..

58460 E45C SETVBV

Set system timers during Vertical Blank routine. Initialized to 49778; $C272. When you set up your own Vertical Blank, it's address should be loaded into the Immediate or Deferred vector in page-3, however, you must load both low and high bytes before the next VBlank executes.

58463 - 58472

If only 1 of the 2 address bytes were loaded when the VBlank routine was executed, the actual jump address will be incorrect and the system will probably crash. Of course, one method would be to wait for the flyscan to be in a safe place on the screen, however, this is the other way to go about it:

```
LDA #ID                 A9 ID
LDX #HI-byte (MSB)      A2 HI
LDY #LO-byte (LSB)      A0 LO
JSR $E45C               20 5C E4
```

or for the USR routine from Basic, use the data: 104, 169, ID, 162, HI, 160, LO, 32, 92, 228, 96

where HI is the MSB, LO is the LSB and ID is either 6 or 7 depending on whether you want to set the Immediate or Deferred VBlank vector, respectively. Using this method, the appropriate vector will be set during the next Vertical Blank. Also see page-3 of memory and the relating VBlank appendices.

58463 E45F SYSVBV

Stage-one VBlank entry point. It performs the processing of a VBlank interrupt. The 2nd and 3rd bytes is the same as the address found in VVBLKI, locations 546 and 547. It is initialized to 49378; $C0E2.

58466 E462 XITVBV

Exit from the VBlank routine, entry point. Used to restore the system to its pre-interrupt state and resume normal processing. The 2nd and 3rd bytes is the same as the deferred interrupt address at VVBLKD, locations 548 and 549. It is initialized to 49802 $C28A.

58469 E465 SIOINV

SIO utility initialization entry point. Initialized to 59740 $E95C. OS use only.

58472 E468 SENDEV

Send enable routine. Initialized to 60439; $EC17. OS use only.

58475 E46B INTINV

Interrupt handler initialization. Initialized to 49164; $C00C. OS use only.

58478 E46E CIOINV

CIO utility initialization. Initialized to 58561; $E4C1.

58481 E471 SELFSV

Self-test mode entry. Initialized to 61987; $F223. The self-test mode can be executed with a JMP here, a USR here, typing BYE in Basic, typing DOS in Basic when DOS has not been loaded and turning the computer on with the option key depressed with the disk drive turned off.
This area used to be what was known as the "Blackboard" mode, which no longer exists in the XL/XE's, however, you can simulate it! In Turbo-Basic it is easily simulated by typing ENTER "E:" (I think?), but in normal Atari Basic the situation differs. You can use LOAD "E:", but an error occurs after about 12 bytes have been inputted with the use of the return key. You can overcome this with: 0 TRAP 0:LOAD "E:", this way, whenever an error occurs the screen clears. Not brilliant, but affective. Perhaps someone knows of a solution. I'm pretty sure a few simple POKEs could rectify it. Anyhow, here's a more suitable simulation of the mode:

```
0 OPEN #1,4,0,"E:"
1 GET #1,K:? CHR$(K);:GOTO 1
```

This will work fine, for an almost perfect simulation, add a TRAP and disable the BREAK key.

58484 E474 WARMSV

Warmstart entry point routine (Reset button vector). Initialized to 49808; $C290 which initializes the OS RAM region. The Reset key causes an NMI interrupt and a chip-reset (CR). This interrupt seems to be at hardware level only so it appears that you cannot disable the action of the Reset-key. I have often wondered what would happen if you switch the ROM OS into a RAM OS, then when you press Reset, will the system execute your RAM OS Reset routine OR will the original ROM OS be switched back in before the Reset routine is executed? I've never tried it so I'm not sure, but I would probably expect the ROM OS to be switched back in first. You can also USR here to simulate the press of the Reset key.

58487 E477 COLDSV

Coldstart (power-up) entry point. Initialized to 49864; $C2C8 which initializes the OS and user RAM regions wiping out any programs etc.. You can perform power-up by USR'ing here.

58490 E47A RBLOKV

Cassette read block routine entry point. Initialized to 64909; $FD8D. OS use only.

58493 E47D CSOPIV

Cassette OPEN for input vector. Initialized to 64759; $FCF7. OS use only.

58496 E480 PUPDIV

Entry to power-on display (Self-Test mode in all XL/XEs except the 1200XL; Atari logo screen in 1200XL. Initialized to 61987; $F223. Try USR'ing to this address.

58499 E483 SELFTSV

Entry to Self-Test mode once switched into low memory at 20480; $5000.

58502 E486 PENTV

Entry point to the handler uploaded from the disk-drive or a peripheral. Initialized to 61116; $EEBC.

58505 E489 PHUNLV

Entry point to the uploaded handler unlink routine. Initialized to 59669; $E915.

58508 E48C PHINIV

Entry point to the uploaded handler initialization routine. Points to 59544; $ E898.

58511 E48F GPDVV

Generic parallel device handler general purpose vector. This can be used to interact with any device connected to the expansion port, simply copy this address into HATABS, locations 794 - 828 along with an appropriate device name character such as V:, G: or T:.

58526 - 59192

For more information on the expansion bus then see the relating appendix. Note that there are 7 vectors here, corresponding to the vector tables residing at 58368; $E400.

58526-58559 E49E-E4BF ...

Unused, zero forced. This area is available if your ROM OS is used as a RAM OS, and indeed so are any other relating areas above this address.

58560 E4C0 ...

Seems to be unused, just a $60 code.

58561 E4C1 ICIO

Initialize CIO.

58588 E4DC IIN

IOCB not OPEN error routine.

58591 E4DF CIO

This is the CIO, it includes the following routines:

Address:		Routine:
58640	$E510	Nonexistant device error
58645	$E515	Load peripheral handler for OPEN
58650	$E51A	Perform CIO command
58687	$E53F	Execute OPEN command
58716	$E55C	Initialize IOCB for OPEN
58742	$E576	Poll peripheral for OPEN
58748	$E57C	Execute CLOSE command
58775	$E597	Execute STATUS and SPECIAL (XIO) commands
58802	$E5B2	Execute GET command
58910	$E61E	Execute PUT command
58992	$E670	Set status
58994	$E672	Complete CIO operation
59029	$E695	Compute handler entry point
59067	$E6BB	Decrement buffer length
59080	$E6C8	Decrement buffer pointer
59089	$E6D1	Increment buffer pointer
59096	$E6D8	Set final buffer length
59114	$E6EA	Execute handler command
59124	$E6F4	Invoke device handler
59135	$E6FF	Search handler table
59158	$E716	Find device handler

59193 - 60920

59193 E739 PHR

Peripheral handler loader routines are:

Address:		Routine:
59193	$E739	Initialization
59326	$E7BE	Perform poll
59358	$E7DE	Load handler
59414	$E816	Get byte routine
59443	$E833	Get next load block
59485	$E85D	Search handler chain
59540	$E894	Handler warm-start initialization
59544	$E898	Warm-start initialization with chaining
59550	$E89E	Cold-start initialization
59584	$E8C0	Initialize handler and update MEMLO
59648	$E900	Initialize handler
59669	$E915	Handler unlinking

59740 E95C SIO

The SIO routines include:

59740	$E95C	Initialization
59761	$E971	SIO main routine
59946	$EA2A	Complete SIO operation
59959	$EA37	Wait for completion or ACK
60040	$EA88	Send buffer to serial bus
60077	$EAAD	Process serial output ready IRQ
60140	$EAEC	Process serial output complete
60157	$EAFD	Receive
60199	$EB27	Indicate timeout
60204	$EB2C	Process serial input ready IRQ
60295	$EB87	Set buffer pointers
60317	$EB9D	Process cassette I/O
60433	$EC11	Timer expiration
60439	$EC17	Enable SIO send
60480	$EC40	Enable SIO receive
60502	$EC56	Set for send or receive
60548	$EC84	Disable send or receive
60570	$EC9A	Get device timeout
60585	$ECA9	Table of SIO interrupt handlers
60591	$ECAF	Send to intelligent device
60608	$ECC0	Set timer and wait
60616	$ECC8	Compute baud rate
60718	$ED2E	Adjust VCOUNT value
60733	$ED3D	Set initial baud rate
60871	$EDC7	Process BREAK key
60898	$EDE2	Set SIO VBLANK parameters

60921 EDF9 TPFV

Table of POKEY frequency values (24 bytes).

60945 EE11 NTSC/PAL

Table of constant values.

60957 EE1D TABLES

Screen memory and DL tables:

Address:	Routine:
60957 $EE1D	Screen memory allocation
60973 $EE2D	Display list entry counts
61005 $EE4D	ANTIC graphics modes
61021 $EE5D	Display list vulnerability
61037 $EE6D	Left shift columns
61053 $EE7D	Mode column counts
61069 $EE8D	Mode row counts
61085 $EE9D	Right shift counts
61101 $EEAD	Display masks

61116 EEBC PHE

Peripheral handler entry routines:

Address:	Routine:
61116 $EEBC	Peripheral handler entry
61177 $EEF9	Peripheral poll at OPEN
61222 $EF26	Put byte for provisionally OPEN IOCB

61294 EF6E SIN

Screen initialization routines, including other screen handler routines:

61294 - 62199

Address:		Routine:
61294	$EF6E	Initialization
61326	$EF8E	Perform screen OPEN
61332	$EF94	Perform editor OPEN
61340	$EF9C	Complete OPEN command
61824	$F180	Screen GET byte
61839	$F18F	Get data under cursor
61860	$F1A4	Screen PUT byte
61873	$F1B1	Check for end-of-line (EOL)
61898	$F1CA	Plot point
61929	$F1E9	Display
61960	$F208	Set exit conditions
61982	$F21E	Screen STATUS
61987	$F223	Self-Test entry point
61997	$F22D	Screen editor special (just an RTS)
61998	$F22E	Screen editor CLOSE
62026	$F24A	Editor GET byte (see GETCHAR below)
62128	$F2B0	Editor PUT byte (see OUTCHAR below)
62142	$F2BE	Process character

62026 F24A GETCHAR

JSR here to fetch a keypress from the keyboard, this acts just like the Basic GET #1,K operation where channel #1 has OPENed the keyboard for input, ie: OPEN #1,4,0,"K:". The Atascii value of the character pressed is returned in the Accumulator. Note that this is a very familiar incompatibility problem between old 4/800 software and the XL/XEs, since this routine used to reside at locations 63038; $F63E (EGETCH).

62128 F2B0 OUTCHAR

This is the PUT character routine which used to reside at 63140; $F6A4 (EOUTCH). Used to put the Atascii character in the Accumulator onto the screen in the next print location. As described above, this character output routine is also incompatible with some older sofware, since illegal calls are sometimes made directly to these routines, at their old addresses!

62200 - 63266

62200 F2F8 IGN

Exactly the same as the GETCHAR routine on the previous page, except that any keyboard character pressed prior to the call of this routine is not cleared. The routine knows if the character has not been cleared when the value in location 764 is not equal to 255.

62205 F2FD KGB

Keyboard GET byte routine. The keyboard handler includes these routines:

Address:		Routine:
62432	$F3E0	ESCape character handler
62438	$F3E6	Cursor up
62451	$F3F3	Cursor down
62464	$F400	Cursor left
62474	$F40A	Cursor to right margin
62476	$F40C	Set cursor column
62481	$F411	Move cursor point
62491	$F41B	Cursor to left margin
62496	$F420	Clear screen
62528	$F440	Cursor home (top-left corner)
62586	$F47A	TAB character handler
62613	$F495	Set TAB
62618	$F49A	Clear TAB
62623	$F49F	Insert character
62677	$F4D5	Delete character
62732	$F50C	Insert line
62752	$F520	Delete line
62806	$F556	Sound bell (CTRL-3)
62815	$F55F	Cursor to bottom
62821	$F565	Double-byte double decrement
62825	$F569	Store data for fine scrolling
62840	$F578	Double-byte single decrement
62880	$F5A0	Set scrolling display list entry
62892	$F5AC	Convert cursor row/column to address
62986	$F60A	Advance cursor
63073	$F661	RETURN with scrolling
63077	$F665	RETURN
63150	$F6AE	Subtract end point
63164	$F6BC	Check cursor range
63256	$F718	Restore old data under cursor

63267 - 64336

63267 F723 BMI

Bitmap routines for the editor and screen handler.

63479 F7F7 SCR

Screen scroll routines.

63665 F8B1 CBC

Buffer count computation routines; various keyboard, editor and screen routines follow also:

Address:		Routine:
63768	$F918	Delete line
63804	$F93C	Control character check
63820	$F94C	Save row/column values
63831	$F957	Restore row and column
63842	$F962	Swap cursor with regular cursor position
63875	$F983	Sound key click
63895	$F997	Set cursor at left edge
63910	$F9A6	Set memory scan counter address
63919	$F9AF	Perform screen special command

64260 FB04 TMSK

Various screen and keyboard tables:

Address:		Routine:
64260	$FB04	Bit masks
64264	$FB08	Default screen colours (708 - 712)
64269	$FB0D	Control character routines. Each entry is 3 bytes; the control character and the 2 byte routine address
64317	$FB3D	Shifted function keys (1200XL)
64329	$FB49	Atascii to internal conversion constants
64333	$FB4D	Internal to Atascii conversion constants
64337	$FB51	Keyboard definition table (see next page)
64529	$FC11	Function key definitions

64337 - 65394

64337 FB51 KDT

192-byte keyboard definition table. See 121 and 122.

64537 FC19 KIRQ

Keyboard IRQ processing routines (nothing to do with StarTrek); Character checking and processing, Control-1, HELP key, Control and function keys (1200XL). The 1200XL routines also remain in other XL and XE's OS's, although, they appear to be unused.

64708 FCC4 FDL

Process display list interrupt for fine scrolling.

64728 FCD8 CIN

Cassette initialization routine, including cassette I/O routines and NTSC/PAL constants for file leader length and beep duration.

65177 FE99 PIN

Printer initialization and I/O routines including:

Address:		Routine:
65218	$FEC2	Printer OPEN
65227	$FECB	Printer PUT byte
65259	$FEEB	Fill printer buffer
65270	$FEF6	Perform printer PUT
65287	$FF07	Printer CLOSE
65300	$FF14	Setup DCB for printer
65348	$FF44	Printer timeout from STATUS
65355	$FF4B	Process print mode

65395 - 65535

65395 FF73 VFR

ROM checksum verify routines for 1st 8K bank.

65426 FF92 VSR

Verify routines for ROM checksum, 2nd 8K bank, inclusive of routines to examine checksum region and table of addresses to verify.

65518-65529 FFEE-FFF9 ...

Checksum and identification for the ROM area 57344 - 65535; $E000 - $FFFF. See 49152; $C000 also.

Byte:	Use:
65518 $FFEE	Revision date D1 and D2 (4-bit BCD)
65519 $FFEF	Revision date M1 and M2
65520 $FFF0	Revision date Y1 and Y2
65521 $FFF1	Option byte: 1 = 1200XL 2 = 800XL
65522-26 $FFF2-6	Part number in the form AANNNNNN
65527 $FFF7	Revision number (my 800XL is 2)
65528-9 $FFF8-9	Checksum bytes (LSB/MSB)

65527 should read 1 for the 600XL and 2 for the 800XL. For the 1200XL, 64728 should not read 162.

65530-65535 FFFA-FFFF Machine Vectors

Contains NMI, RESET (power-up) and IRQ service vectors. Initialized to 49176; $C018, 49834; $C2AA and 49196; $C02C, respectively.

A small comment.

Well, there you have it fellow Atarians. The whole truth and nothing but... about the XL and XE 8-bit machines. If you're an amateur programmer, then you will find most of the information in this book very tedious so you'll need a lot of patience. It might be a good idea to send off for one of the various Atari newsletters or disk magazines. A very good disk magazine is called "THE GRIM REAPER" and the editor goes by the name JOHN E. This address and several others are in a supporting appendix. The more experienced programmers amoung you will probably be glad for the publication of this book. Even you advanced programmers might find some interesting information in this book. I'm no 'know-it all' by the way, there is quite a lot of stuff that I've never delved into, in fact I don't think I'll ever stop learning! One thing I would like to be is less lazy, so if you think this book is good, or bodatiously amazing (!), or perhaps just crap, why not let me know and tell me why you think so and what I could have extended on etc.. Who knows, I might even write another book! This is my 1st and it took me several months.

The master copy of this book has been re-arranged and printed by T.W.A.U.G., using a 24 pin printer the STAR/LC24-100.

INDEX BY LABEL.

OK then, here's the 1st index which gives the locations involved according to the alphabetically listed name, either of a single location or group.

NAME	LOCATION	NAME	LOCATION
ABUFPT	28-31	CARTB	32768-40959
ACMISR	727,728	CARTCK	1003
ACMVAR	1005-1015	CARTFLG	49148
ADDCOR	782	CARTINI	49150,49151
ADRESS	100,101	CARTINIT	50217
AFP	55296	CARTLD	49146,49147
ALLPOT	53768	CARTOPT	49149
ANTIC	54272-54783	CASBUF	1021-1151
APPMHI	14,15	CASETV	58432
ARCOET	57262	CASFLG	783
ATACHR	763	CASINI	02,3
ATAN	48759	CASSBOOT	50798
ATRACT	77	CASSBT	1002
AUDC1-4	53761-53767	CAUX1	572
AUDCTL	53768	CBAUDL/H	750,751
AUDF1-4	53760-53766	CBC	63665
BASICF	1016	CCNTLEN	54528-54783
BASICVER	43234	CCOMND	571
BFENLO/HI	52,53	CDEVIC	570
BFLAG	1792	CDTMA1	550,551
BITMSK	110	CDTMA2	552,553
BIWTARR	1796,1797	CDTMF3	554
BLDADR	1794,1795	CDTMF4	556
BLDISP	1809	CDTMF5	558
BLIM	650	CDTMV1	536,537
BMI	63267	CDTMV2	538,539
BOOT	09	CDTMV3	540,541
BOOTAD	578,579	CDTMV4	542,543
BOOTERROR	50237	CDTMV5	544,545
BPTR	61	CH	764
BRCNT	1793	CH1	754
BRKKEY	17	CHACT	755
BRKKEY	566,567	CHACTL	54273
BRUN	10060	CHAR	762
BSIO	1900	CHARSET1	57344-58367
BSIOR	1906	CHARSET2	52224-53247
BUFADR	21,22	CHBAS	756
BUFADRS	51002	CHBASE	54281
BUFCNT	107	CHKSNT	59
BUFRFL	56	CHKSUM	49
BUFRLO/HI	50,51	CHLINK	1019,1020
BUFSTRM	108,109	CHSALT	619
CARTA	40960-49151	CIN	64728
CIO	58591	DFLADR	1810,1811
CIOCHR	47	DFLAGS	576
CIOINT	58561	DFLINK	1807,1808
CIOINV	58478	DFMCLS	2837
CIOV	58454	DFMDDC	2983
CIX	242	DFMGET	2751
CKEY	1001	DFMOPN	2219
CLEARSCRN	62496	DFMPUT	2508

INDEX BY LABEL.

CLMJMP 6418
CMCMD 07
COLAC 114,115
COLBK 53274
COLCRS 85,86
COLDST 580
COLDSV 58487
COLINC 761
COLOR 200
COLOUR 200
COLOUR0 708
COLOUR1 709
COLOUR2 710
COLOUR3 711
COLOUR4 712
COLOURS 704-712
COLPF0-3 53270-53273
COLPM0-3 53266-53269
COLRSH 79
CONSOL 53279
COS 48561
COUNTR 126,127
CPYFIL 8990
CRETRY 668
CRITIC 66
CRSINH 752
CRSROW 108
CRVTSL/H 4264,4266
CSOPIV 58493
CURDSL/H 4226,4229
DATAD 182
DATALN 183,184
DAUX1/2 778,779
DBSECT 577
DBUF 7668
DBUFLO/HI 772,773
DBYTEL/H 776,777
DCB 768-779
DCOMND 770
DDCC 56780
DDEVIC 768
DDMG 10690
DECTIMR 49749
DEGFLG 251
DELFIL 8649
DELTAC 119,120
DELTAR 118
DERRF 1004
DEVMEM 53504-53759
DFFM 11528
ENDFMS 5377
ENDPT 116,117
EOUTCH 62128
ERRFLG 575
ERRNO 4789
ERRSAV 195
ESCFLG 674
DFMSDH 1995
DFMSTA 2817
DHEADR 576-579
DIGRT 241
DINDEX 87
DINIT 50851
DINT 2016
DIRLST 8505
DISKBOOT 50571
DISKINV 58451
DISKIV 58448
DLISTL/H 54274,54275
DLRAM 39967
DMACTL 54272
DMASAV 733
DMASK 672
DMENU 7951-8278
DMRAM 39967-40959
DOS 5440
DOS3 3889
DOSINI 12,13
DOSINIDL 6044,6045
DOSOS 8309
DOSUSE 1792-7419
DOSVEC 10,11
DOSVECDL 5446,5450
DRETRY 701
DRKMSK 78
DRVBUF 6780-7547
DRVBYT 1802
DSCTLN 725,726
DSFLG 1806
DSKFMS 24,25
DSKTIM 582
DSKUTL 26,27
DSPFLG 766
DSTAT 76
DSTATS 771
DTIMLO 774
DUNIT 769
DUNUSE 775
DUPDSK 10690
DUPEND 13062
DUPFIL 11528
DUPFLG 5533
DUPSYS 5440-13062
DVSTAT 746-749
EDITRV 58368
EEXP 237
EGETCH 62026
GPRIOR 623
GRACTL 53277
GRAFM0-3 53265
GRAFP0-3 53261-53264
GTIA 53248-53503
HARDI 50394
HARDWARE 53248-55295

ESIGN	239
EXP	56768
FADD	55910
FASC	55526
FCB	4993-5120
FCHRFLG	240
FDIV	56104
FDL	64708
FDSCHAR	3850
FEOF	63
FILDAT	765
FILDIR	5121
FILELD	5576
FILFLG	695
FINE	622
FKDEF	96,97
FLDOP	56717
FLDOR	56713
FLD1P	56732
FLD1R	56728
FLPTR	252,253
FMOVE	56758
FMSRAM	1792-5377
FMTDSK	9856
FMUL	56027
FMZSPG	67-73
FNDCODE	3742
FPI	55762
FPROM	55296-57343
FPSCR	1510-1515
FPSCR1	1516-1535
FPTR2	254,255
FRO	212-217
FR1	224-229
FR2	230-235
FRE	218-223
FREQ	64
FRESECT	4293
FRMADR	104,105
FRX	236
FSTOP	56747
FSTOR	56743
FSUB	55904
FTYPE	62
GAMCTL	624-647
GBYTEA	719,720
GETCHAR	62026
GETSECTOR	4358
GINTLK	1018
GLBABS	736-739
GPDVV	58511
IOCB7	944-959
IOCBS	832-959
IOCMD	192
IODVC	193
IRQEN	53774
IRQST	53774
HATABS	794-828
HELPFG	732
HIBYTE	648
HIMEM	741,742
HITCLR	53278
HIUSED	715,716
HNDLRVCTR	58368-58447
HOLD1	81
HOLD2	671
HOLD3	669
HOLD4	700
HOLDCH	124
HPOSM0-3	53252-53255
HPOSP0-3	53248-53251
HSCROL	54276
ICAX1Z	42
ICAX2Z	43
ICAX3Z/4Z	44,45
ICAX5Z	46
ICAX6Z	47
ICBAL/HZ	36,37
ICCOMT	23
ICCOMZ	34
ICDNOZ	33
ICHIDZ	32
ICIO	58561
ICSPRZ	44-47
ICSTAZ	35
IFP	55722
IGN	62200
IIN	58588
IMASK	651
INBUFF	243,244
INISAV	6044,6045
INITAD	738,739
INITIO	6518
INITPDEV	51468
INSDAT	125
INTEMP	557
INTHNDLRS	49152-52223
INTINV	58475
INTRRINI	49164
INTRVEC	522,523
INVFLG	694
IOCB0	832-847
IOCB1	848-863
IOCB2	864-879
IOCB3	880-895
IOCB4	896-911
IOCB5	912-927
IOCB6	928-943
NEWCOL	758,759
NEWROW	757
NGFLAG	01
NMIEN	54286
NMIINIT	49176
NMIRES	54287

ISRODN	6630	NMIST	54287
ISRSIR	6691	NOCKSM	60
JMPTBL	24,25	NOCLIK	731
JVECK	652	NSIGN	238
KBCODE	53769	NTSC/PAL	60945
KDEFTBL	64337	OLDADR	94,95
KEYBDV	58400	OLDCHR	93
KEYCLICK	63875	OLDCOL	91,92
KEYDEF	121,122	OLDROW	90
KEYDEL	753	OPNTMP	102,103
KEYDIS	621	OPT	5534
KEYREP	730	OSDBUFS	512-1151
KGB	62205	OSRAM	0-127
KIRQ	64537	OSROMHI	55296-65535
KRPDEL	729	OSROMLO	49152-53247
LBPR1	1406	OSTABLS	512-1151
LBPR2	1407	OSVARS	512-1151
LBUFF	1408-1535	OUTCHAR	62128
LDFIL	10522	P#PF	53252-53255
LDMEM	6457	P#PL	53260-53263
LEDCTL	54017	P1OCOET	56909
LINE	7588	PACTL	54018
LISTDIR	3501	PADDL0	624
LKFIL	10608	PADDL1	625
LMARGN	82	PADDL2	626
LMTR	6432	PADDL3	627
LNFLG	00	PAGE0	0-255
LOADAD	721,722	PAGE1	256-511
LOADFLG	202	PAGE6	1536-1791
LOADFLG	5535	PAL	53268
LOG	57037	PALNTS	98
LOG10	57041	PBCTL	54019
LOGCOET	57202	PBPNT	734
LOGCOL	99	PBUFSZ	735
LOGMAP	690-693	PCOLR0	704
LOMEM	128,129	PCOLR1	705
LPENH	564	PCOLR2	706
LPENV	565	PCOLR3	707
LSICHIPS	53248-55295	PDMSK	585
LTEMP	54,55	PDVMSK	583
M#PF	53248-53251	PENH	54284
M#PL	53256-53259	PENTV	58502
MEMFLG	6046	PENV	54285
MEMLDD	5899	PHE	61116
MEMLO	743,744	PHINIV	58508
MEMSFC	5947	PHR	59193
MEMTOP	144,145	PHUNLV	58505
MEMTOP	741,742	PIA	54016-54271
MEOLFLG	146	PIN	65177
MINTLK	1017	PIO	51507
MLTTMP	102,103	PIRQQ	64537
NEWADR	654,655	PLYARG	1504
PLYEVL	56640	SAVADR	104,105
PMBASE	54278	SAVCUR	190
POKADR	149,150	SAVIO	790
POKEY	53760-54015	SAVMSC	88,89
POKEY2	53760-54015	SBUSCOM	522-527

INDEX BY LABEL.

POKMSK	16
PORTA	54016
PORTB	54017
POTO-3	53760-53767
POTGO	53771
PPTMPA	588
PPTMPX	589
PRINTV	58416
PRIOR	53275
PRNBUF	960-999
PROMPT	194
PTABW	201
PTRIGO	636
PTRIG1	637
PTRIG2	638
PTRIG3	639
PUPBT1-3	829-831
PUPDIV	58496
PZASMV	176-207
PZBASV	146-202
PZERO	0-255
PZRAM	0-255
PZUNUSD	203-209
RADFLG	251
RAMLO	04,5
RAMSIZ	740
RAMSWTCH	54017
RAMTOP	106
RANDOM	53770
RBLOKV	58490
RDDIR	4206
RDNXTS	4111
RDVTOC	4235
RECLEN	581
RECVDN	57
RELADR	568,569
RELADR	586,587
RENFIL	9783
RESET	59544
REVDATE	65518-65520
REVISION	49163
RMARGN	83
ROMVCTRS	65530-65535
ROWAC	112,113
ROWCRS	84
ROWINC	760
RTCLOK	18,19,20
RUNAD	736,737
RUNADR	713,714
RUNADREN	51154
RUNSTK	142,143
SABYTE	1801
SASA	1804,1805
STOFFSETS	51799
STOPLN	186,187
STRIGO	644
STRIG1	645
SCR	63479
SCREENV	58384
SCRFLG	699
SCRNEDP	148,149
SCRNRAM	656-703
SCROLL	63479
SDLSTL/H	560,561
SDMCTL	559
SELFSV	58481
SELFTEST	20480-22527
SELFTSEL	51452
SENDEV	58472
SERIN	53773
SEROUT	53773
SETUP	4452
SETVBL	49778
SETVBV	58460
SFDIR	3873
SFLOAD	5540
SFTTIMR	536-558
SHFAMT	111
SHFLOK	702
SHPDVS	584
SIN	48551
SIN	61294
SIO	59740
SIOINV	58469
SIOORG	59740
SIOV	58457
SIZEMO-3	53260
SIZEPO-3	53256-53259
SKCTL	53775
SKREST	53770
SKSTAT	53775
SLFTSV	58499
SOUNDBELL	62806
SOUNDR	65
SPARE	563
SQR	48869
SRTIMR	555
SSFLAG	767
SSKCTL	562
STACK	256-511
STACK2	1152-1405
STACKP	792
STARP	140,141
STATUS	48
STCAR	9966
STICKO	632
STICK1	633
STIMER	53769
STMCUR	138,139
STMTAB	136,137
VSCROL	54277
VSERIN	522,523
VSEROC	526,527
VSEROR	524,525

INDEX BY LABEL.

SUBTMP	670
SUPERF	1000
SWPFLG	123
SYSVBL	49378
SYSVBV	58463
TABLES	60957
TABMAP	675-689
TEMP	574
TEMP	80
TEMP1	786,787
TEMP2	788
TEMP3	789
TEMPCHR	80
TESTVER2	10483
TIMER1	780,781
TIMER2	784,785
TIMFLG	791
TINDEX	659
TMPCOL	697,698
TMPLBT	673
TMPROW	696
TMSK	64260
TOADR	102,103
TOLDADR	&&&,667
TOLDCHR	665
TOLDCOL	663,664
TOLDROW	662
TPFV	60921
TRAMSZ	06
TRIGO-1	53264-53265
TSTAT	793
TXTCOL	657,658
TXTMSC	660,661
TXTOLD	662-667
TXTROW	656
ULFIL	10608
USRDOS	5888
VARNOLE	211
VARSTG	7924
VARTYPE	210
VAUX2	573
VBREAK	518,519
VCOUNT	54283
VDELAY	53276
VDSLST	512,513
VERIFY	1913
VFR	65395
VIMIRQ	534,535
VINTER	516,517
VKEYBD	520,521
VNTD	132,133
VNTP	130,131
VPIRQ	568,569
VPRCED	514,515
VSFLAG	620
VSR	65426
VTIMR1	528,529
VTIMR2	530,531
VTIMR4	532,533
VVBLKD	548,549
VVBLKI	546,547
VVTP	134,135
WARMST	08
WARMSV	58484
WBOOT	10201
WILD*	3783
WMODE	649
WRTDOS	4618
WRTMEMS	10111
WRTNXS	3988
WSYNC	54282
WTBUR	2591
XBCONT	1799,1800
XDELETE	3122
XFNME	3818,3822
XFORMAT	3352
XITVBV	58466
XLOCK	3196
XMTDON	58
XNOTE	3331
XPOINT	3258
XRENAME	3033
XUNLOCK	3203
ZCHAIN	74,75
ZF1	55878
ZFRO	55876
ZHIUSE	717,718
ZIOCB	32-47
ZIOVARS	48-75
ZLOADA	723,724
ZPFP	212-255
ZTEMP1	245,246
ZTEMP3	249,250
ZTEMP4	247,248

INDEX BY SUBJECT

This 2nd and final index is organized by subject. This is one thing Ian Chadwick should have included when he made the XL/XE version of Mapping the Atari: a FULL XL/XE subject index!

SUBJECT	LOCATIONS
ANTIC	
direct memory access (DMA)	559, 54272
instruction set pointer	560, 561
interrupts	512, 513
mode numbers	87
P/M graphics	559, 54272
ROM	54272-54783
BASIC	
array table	140, 141
blackboard mode	no longer exists
cartridge	40960-49151
disable	1016
error codes and lines	186, 187, 195
Floating Point routines (FP)	48549-49145
GOTO and GOSUB	142, 143
graphics modes	87
jump to DOS	10, 11, 6040
line numbers	136, 137
machine-code file load	5576
memory pointers	128, 129, 144, 145, 740-744
OPERATOR list	42509
page zero	128-209
program	14, 15, 136-139
program end	14, 15, 144, 145
runtime stack	142, 143
stack	256-511
statement pointer and table	136-139
stopped at line action	186, 187
string table	140, 141
TOKEN list	42159
variable name, value tables	130-135
BLACKBOARD MODE	no longer exists
BOOT	
cassette	9, 12, 1002
disk boot initialization	12, 13
disk boot routine	4, 5, 50571
DOS vector	10, 11, 5446, 5450
self-test package	20480, 58481
success flag indicator	9
system lockup	9

COMPLETE & ESSENTIAL MAP

BORDER	
colour registers	704, 712
disable/enable/enlarge	559
rainbow	712
BREAK KEY	
disable	16, 53774
enable	16, 53774
flag	17, 53774
forced	53775
interrupt	16, 53774
restored	16, 53774
shadow register	16, 53774
status	17, 48
vector	566, 567
BUFFERS	
cassette	1021-1151
command frame	570-573
data	50-53, 56
device (SIO data)	772, 773
disk	1024
line	735, 1408
printer	734, 960-999
ZIOCB	36, 37, 40, 41
CARTRIDGES	
A (left) cartridge	40960-49151
B (right) cartridge	32768-40959
Basic (see A cartridge)	40960-49151
DOS boot flag	49149
initialization vector	49150-49151
load address vector	49146-49147
test for presence	6, 50289
CASSETTE	
baud rate	750, 751
beep count	64, 65
boot	2, 3, 9, 1001, 1002
buffer	61, 1021-1151
buffer size	650
buzzer	65020
end of file	63
handler routines	64728-65176
handler vector	58432
initialization vector	2, 3
inter-record gap (IRG)	62
load	2, 3
mode	649, 783
motor control	54018
OPEN for input	64759
read block entry	64742
record size	1021
run address	10, 11, 12, 13
status register	?
voice track	53775

COMPLETE & ESSENTIAL MAP

CHARACTER
- ATASCII 763, 52224, 57344
- auto repeat logic 729, 730, 764
- bit mapping 52224, 57344
- blinking text 548, 549, 755
- character sets 756, 52224, 57344
- character set address 756, 54281
- colours 708-712, 756
- control codes 766
- control key 702, 764
- control register 674, 694, 755
- cursor inhibit 752, 755
- hardware code 764
- internal code 762, 764
- inverse 694
- invisible inverse 755
- last character read, written 763
- lowercase outside graphics 0 756
- logic processing 124
- mode 755, 54273
- move set into RAM 756
- printer output ?
- prior character code 754
- ROM routines 62205-63266
- screen location 87, 88, 89
- shadow 756
- shift key 702, 53775
- tests 64537
- translation of codes 52224, 57344
- under cursor 93
- uppercase outside graphics 0 756
- upside down 512, 513, 755, 54273

CHECKSUM 49, 59, 60

CIO
- command 23
- IOCBs 832-959
- utility initialization 58561
- variables 43
- vector 58454

CLOCK
- attract mode 77-79
- CPU APC D02
- realtime 18-20
- serial clock lines 53775
- sound use 53768, 53784

COLDSTART
- cassette boot 9, 1001
- disk boot 9
- entry point 58487
- flag 580
- power-up 49864

COLOUR
- artifacting 710
- attract mode 77-79
- default values 708-712, 64264-64268
- GTIA registers 704-712, 53266-53274
- player/missile shadows 704-707
- playfield shadows 708-712
- rotating 77, 703
- screen mode 87, 560, 561

CONSOLE KEYS
- basic disable 53279, 54017
- basic enable 53279, 54017
- cassette boot 53279
- detection 53279
- disk boot 53279
- self-test package 20480, 54017, 58481

CPU
- clock rate APC D2
- instruction set APC D4

CURSOR
- advance 85, 86
- character under 93, 125
- click 731,
- column 85, 86
- current position 84-86, 94, 95
- end of line (EOL) 125
- flash 755
- graphics 90-92
- home 121, 122, 62528
- inhibit (disable) 752
- LOCATE 85, 86
- logical line 99
- move to left margin 121, 122, 62491
- move to right margin 121, 122, 62474
- move to bottom-left corner 121, 122, 62815
- opaque, transparent 755, 54273
- out of range error 87
- pause toggle 767
- previous position 90-92
- row 84
- tab stops 675-689
- tab width using the comma 201
- text window 85, 86, 123

DEVICE
  buffer 772, 773
  byte transfer 776, 777
  command 770
  command frame retries 668
  Device Control Block (DCB) 768-779
  drivers (adding) 794-828
  error status 746-749
  HANDLER address table 794-828
    routines 58591-59192
    vectors 794-828, 58368-58447
  memory 55296-55323
  retries 701
  status registers 746-749
  timeout value 747
  vector tables 58368, 58447
  ZIOCB number 33

DIRECT MEMORY ACCESS (DMA)
  graphics control 53277
  ROM 54272
  shadow 559

DISK
  beep during I/O 65, 60504
  boot 9-13, 1001, 1002
  boot load address 578, 579
  boot continuation 4, 5
  boot initiation address 12, 13
  boot routine 50571, 50619
  buffer 21, 22, 1802
  flags 576, 577
  FMS page zero 67-73
  FMS pointer 24, 24
  handler commands 778
  handler routines 50851-51001
  handler vector 58448, 58451
  header bytes 576-579
  initialization address 12, 13
  records OPEN 1801
  retries 668
  run address 736-739
  start vector 10, 11
  timeout 582
  utilities 26, 27
  vector 10, 11
  verify routines 1913

DISPLAY
HANDLER
routines 61294
vector 58384
LIST
address 100, 101, 560, 561
enable 559, 54286
entries 81, 560, 561
instructions 559-561
interrupts 512, 513, 54286, 64708
location 560, 561, 65530, 65531
lowest address 14, 15, 106, 560, 561
pointer 560, 561
reserving memory 106, 560, 561
ROM tables 60957
screen mode 87, 559, 560, 561, 623
scrolling 560, 561, 54276, 54277
size 81, 560, 561
vertical line counter 54283
logical line map 690-693
memory 14, 15
pixel mask 672
RAM 656-703
registers 76, 80, 81, 99-105, 107-127
routines 61294
text window 656-667

DLI
address 512, 513
disable 54286
enable 54286
vector 512, 513

DOS
boot address 10, 11, 578, 579
boot record 1792
buffers 6780-7547, 7588-7923
burst I/O 2592-2773
drives in system 1802
DUP.SYS RAM 5440-13062
filename change 3818, 3822
files reserved 1801
FMS RAM 1792-5377
initialization 12, 13, 738, 739
run address 736, 737
start vector 9-11
wildcard character 3783

DRAW
colour of line 763
cursor 90-92
endpoint of line 84-86, 757-759
flag 695
graphics 0 87
ROM routines ?
screen mode 87

DUP.SYS
  load 10, 11

ERRORS
  BASIC 186, 187, 195
  device 746
  disk I/O 73
  SIO 575

ESCAPE KEY
  control codes (without ESC) 766
  flag 674
  forced 766

FILE
  header bytes 576-579
  initialization address 738, 739
  load address 578, 579
  run address 736, 737

FILL
  colour of fill area 765
  colour of line 763
  endpoint of line 84-86, 757-759
  flag 695

FLOATING POINT
  BASIC ROM 48549-49145
  degree/radians flag 251
  page zero 210-255
  pointers 252-255
  RAM page five 1406-1535
  registers 212-217, 224-229
  ROM (OS) 55296-57343
  trig functions 251

FMS
  page zero buffer 67-73
  pointer 24, 25
  RAM 1792-5377

GRAPHICS
  display mode 87, 659
  DRAW, DRAWTO and FILL 85, 86, 757-759
  IOCB 928-943
  line plotting 112-120, 760, 761
  memory use 88, 89, 106
  mode 8 text 57344
  pixel 672
  players/missiles 53261-53265
  row and column plotting 112-120, 760, 761
  screen memory 14, 15, 123, 126, 127
  scroll 622, 54276, 54277
  tab width 201
  XIO commands 757-759

GTIA	
collisions	53248-53263, 53278
console keys	53279
console speaker	53279
examples	623
mode selection	87, 659, 559, 623
ROM	53248-53503
stick triggers	53264-53265
test	623
text window	623, APC C11
trigger latching	53277
HANDLERS	
interrupt routines	49164
RESET	794
ROM routines	58561
HARDWARE	
memory	53248-55295
HELP KEY	
detection	732
INTERRUPTS	
BREAK key disabled	16
BREAK key vector	566, 567
display list (DLI)	512, 513
enabled	16, 53774, 54286
handler routines	49164
IRQ	16, 514-535, 53774, 49196
NMI	512, 513, 546-549, 54286 49176, 49378, 49802
PIA (periperal)	54018, 54019
POKEY	16, 53774
RAM	512-535, 546-549, 566, 567
serial	16
status request	53774
timer	16
Vertical Blank (VBI)	546-549, 54286, 58460-58468 49378, 49802
IOCB	
graphics screen	928-943
LIST, LOAD and LPRINT	944-959
move	58609
page zero	32-47
RAM	832-959
screen editor	832-847
IRQ	
BREAK key vector	566, 567
service routines	49196, 64537
vectors	514-535
JIFFY	
realtime clock	18-20
vertical scan-line counter	54283

KEYBOARD
- code — 764, 53769
- console keys — 9, 580, 732, 53279
- control key flag — 702, 53769
- controller — 54016
- delay flag — 555, 729, 730, 753
- debounce delay — 729, 730
- display flag — 766
- enable debounce and scanning — 562, 53775
- ESCape key flag — 674
- handler routines — 62205-63266
- handler vector — 58400
- interrupts — 16, 53774
- inverse toggle — 694
- option, select and stop keys — 53279
- repeat rate — 555, 729, 730
- SHIFT key flag — 702, 53769
- start, stop flag (CTRL-1) — 767
- status — 76
- synchronization — 54282
- timer delay — 555

LIGHT PEN
- horizontal value — 564, 54284
- vertical value — 565, 54285

LINE
- bitmap — 690-693
- buffer — 1408-1535
- cursor — 99
- logical line — 83
- margins — 82, 83
- plotting — 112-120, 126, 760-761
- screen editor — 107
- tabs — 201, 675-689

LUMINANCE
- attract mode — 77-79

MACHINE LANGUAGE
- illegal instruction set — APC D4
- instruction set — APC D4
- page six — 1536-1791

MARGINS
- border — 82, 83, 559
- bottom — 84, 560, 561
- editing — 82, 83
- initialization — 82, 83
- left — 82
- playfield size — 559
- right — 83
- scrolling — 82, 83, 54276, 54277
- top — 84, 560, 561

MEMORY
- bank switching — 54017
- hardware — 53248-55295
- RAM — 0-32767, 32768-40959, 40960-49151, 49152-53247, 55296-65535
- ROM — 49152-53247, 55296-65535

MONITOR
- handler routines — 49864

NMI
- DLI — 512, 513, 560, 561, 54286
- reset register — 54287
- service routines — 49176
- status — 54287
- VBI — 546-549, 54286
- vectors — 512, 513, 546-549

OPERATING SYSTEM
- character sets — 52224, 57344
- Floating Point (FP) — 55296
- handlers — 58591-
- ROM — 49152-53247, 55296-65535
- vectors — 58368-58533

PAGE ZERO
- BASIC use — 128-202
- buffer — 21,22
- Floating Point use — 210-255
- FMS registers — 67-73
- IOCB (ZIOCB) — 32-47
- RAM — 203-209, 0-255
- unused RAM (unconditionally) — 28-31, 147, 203-209

PERIPHERAL
- controllers — 54018, 54019
- interrupts — 53774
- ports — 54016

PIA
- ROM — 54016-54271
- stick — 54016
- paddle (pot) triggers — 54016
- ports — 54016, 54018-54019

PLAYER/MISSILE GRAPHICS
- character base — 54279
- collision clear — 53278
- collision detection — 53248-53263
- colour registers — 704-707
- disable — 559, 53277

DMA	559, 54272
enable	559, 53277
fifth player	623, 53275
graphic shape	53261-53265
horizontal movement	53248
horizontal position	53248-53255
location	54279
memory reservation	54279
movement	53248
multicoloured	623, 53275
overlap	623, 53275
priority	623, 53275
resolution (line)	559, 54272
screen boundaries	559, 53248
size and width	53256-53260
using DLIs with PMGs	53261
vertical delay	53276
vertical movement	APC C10
PLAYFIELD	
colours	704-712
DMA	559
enable	559
priority	623, 53275
scrolling	54276, 54277
size	559
POKEY	
enable	53768, 53775
interrupts	16, 514-535
pots	53760-53767
ROM	53760-54015
sound channels	53760-53767
sound control	53768, 53775
stereo upgrade	APC B3
POLYNOMIALS	
random numbers	53770
sound dividers	53761, 53768
POTS	
fast-scan enable	562, 53775
POKEY registers	53760-53767
port state read	53768
shadows	624-631
start read sequence	53771
trigger latch	53277
triggers	636-643, 54016
values	624
POWER-UP	
RAM size	6, 740
ROM stability test	50289, 50394, 65395, 65426
warmstart	8, 9, 580

PRINTER
  buffer 734, 735, 960-999
  character output ?
  handler routines 65177
  handler vector 58416
  IOCB use 944-959
  sideways printing 735
  status 788, 735
  timeout 788

PRIORITY
  ROM 53275
  shadow 623

RAM
  clear memory 88, 89, 106
  free memory 0-32767, 32768-40959
  40960-49151, 49152-53247
  55296-65535
  monitor 0, 1
  page zero 28-31, 147, 203-209, 0-255
  pointer, bottom 743, 744, 1792
  pointer, top 106, 741,742
  pointers, general 4, 5, 15, 128, 129
  protected area (Page-$6) 1536-1791
  RAMtop 106, 740-742
  reserving 106, 743, 744
  screen 88, 89
  size 106, 740
  stack 512-767
  test 4-7
  vector table 58496

RANDOM NUMBER GENERATION
  register 53770
  control 53768

RESET KEY
  coldstart 580, 58487
  DOS 10, 11
  flag 580
  handler routines 49834, 58484
  handler tables 794
  interrupt 54286
  lockup 9
  margins 82, 83
  vector 9, 65530
  warmstart 8, 580, 58484

SCREEN
  bit mapping 110
  boundaries 53248
  buffer 107
  clear memory 88, 89, 106
  clear screen 88
  colour clocks 672

control codes 766
display address 560, 561
display mode 87, 659
graphics modes 87-89, 106
handler vector 58368
IOCB use 832, 928
line buffer 1408-1535
logical line map 690-693
lowest address 14, 15, 88, 89
memory restrictions 741, 742
memory use 88
mode 87, 659
page zero RAM 80-120
PAL compatible 53268
pixel justification 111
pixel mask 672
rows 703
save routines 88, 89
screen modes 560, 561
scrolling 88, 89, 106, 622, 699, 767 54276, 54277
size 76, 88, 89, 559, 672
split screen 123
TAB map 675-689
text rows 703
vertical line counter 54283
wait synchronization 54282

SELF-TEST
address 20480
enable 54017
vector 58481

SERIAL PORT
control 562, 53775
data port 790
input/output 16, 53773
interrupts 16, 53774
reset status 53770
shadow 562
status 53775

SIO
checksum 49
command frame buffer 570-573
data buffer 50-53, 56
device control block 768-779
disk flags 576, 577
error flag 575
flags 56-60
interrupt handler 49164
interrupts 514-527
routines 59740
send enable 58472
stack pointer 792
status 48

timeouts	788
transmission flags	56-60, 701
utility initialization	58469
vector	58457
SOFTWARE	
initiation address	738, 739
run address	736, 737
timers	536-545
SOUND	
audio control	53761-53768
audio frequency	53760-53768
beeps	64, 65
buzzer (CTRL-2)	62806
cassette buzzer	?
clock frequency	53768
console register	53279
CTRL-2 buzzer	66, 62806
distortion	53761
filters	53768
I/O beeps	65, 60504-60547
keyboard speaker	53279
margins	83
octave range	53768
poly counters	53761
STACK	
page one	256-511
runtime	142, 143
STATUS	
device	747
display	76
printer timeout	788
SIO	48
ZIOCB	35
STICK	
attract mode	77
PIA register	54016
read routines	632
shadows	632-635
trigger latch	53277
triggers	644-647
values	632
TABS	
comma spacing	201
stop map	675-689
TEXT	
colour	708-711
size	87
status flag	752, 755
upside down	755

WINDOW 656-667, 703
- address 660, 661
- cursor 123, 656-658
- enabled on mode 0 703
- GTIA 87
- margins 82, 83
- mode 659
- plot 87
- rows available 703
- scrolling 622, 699
- tab width 201

TIMEOUT
- baudrate correction 791
- device 748
- disk 582
- printer 788
- storage 48
- value 788

TIMERS
- attract mode 77
- baud rate 780-782, 784-787
- critical code 66
- hardware (POKEY) 16, 528-533, 53768
- interrupt enable 16, 53774
- jump vectors 550-553
- realtime clock 18-20
- repeat 555
- software (system) 536-558
- start hardware 53769
- suspended 66
- VBLANK 66
- vectors 550-558

TRANSMISSION FLAGS 56-60

TRIGGERS
- GTIA registers 53264-53265
- latches 53277
- paddle (POT) 636-643
- PIA register 54016
- stick (joystick) 644-647

VARIABLE
- assigning values 134
- list 132
- name table (VNT) 130-133
- statement table 136, 137
- string and array table 140, 141
- value table 134, 135

VERTICAL BLANK	
attract mode	77-79
clock	18-20
critical section	66
entry point	58463
exit	58466
interrupts	546-549, 54286
interrupt status	54287
key delay	729, 730, 753
set timers	18, 58460
timer value	0, 1
VECTOR	
BREAK key interrupt	566, 567
break instruction (BRK)	518, 519
cassette handler	58432
CIO	58454
command	23
device handlers	794-828, 58368-58477
disk	10, 11
disk handler	58448, 58451
display handler	58384
DLI	512, 513
immediate IRQ	534, 535
keyboard handler	58400
RESET key interrupt	49834, 58484, 65532
serial interrupt	516, 517
serial proceed line	514, 515
serial receive data ready	522, 523
serial transmit complete	526, 527
serial transmit ready	524, 525
software timer-1	550, 551
software timer-2	552, 553
timer-1 interrupt	528, 529
timer-2 interrupt	530, 531
timer-4 interrupt	532, 533
VBI	546-549
WARMSTART	
entry point	58484, 49808
flag	8, 580
NMI check	8, 54287

COMPLETE & ESSENTIAL MAP

for the

XL / XE

BOOK CORRECTIONS

After reading through the book we have unfortunately found a few page references that do not correspond with the pages indicated in the book.

On page 15 in the paragraph under location 91,92 it indicates to refer to page 97, unfortunately it should read: "See page 85 of the map".

On page 140 in the first paragraph, under location 54272, it reads (Page-45) but it should read "Page-38".

In part two of the book on page 170 in the OPEN paragraph it reads: (See the table on page 96), this is another mistake, it should read "See the table on page 84".

These mistakes have occurred when the author's Master Copy was set up and re-printed as it is now. There were too many large gaps between the lines and some pages had only a few line on them, it would have pushed the cost up too high. Please notify TWAUG with any other errors found in the book, the page references above are the only ones I've found up to now.

The author wasn't able to print the "lesser than < and greater than > characters with his printer, in place he used the square brackets []. Again some of these characters were overlooked, you will find these square brackets in some of the BASIC program listings, mostly in the appendix pages. Please replace these square brackets [] with the lesser than and greater than <> characters, or the programmes wont run.

If we find further mistakes we will update this 'Book correction leaflet' and post it out to our customers. Please keep this leaflet clipped to your book.

TYNE&WEAR ATARI USER GROUP

T.W.A.U.G.

P.O.Box No.8, WALLSEND
Tyne & Wear NE28 6DQ

NEWS Publishers

TWAUG publications presents

THE

Atari XL/XE

Complete And Essential MAP

Including Probably The Most
Comprehensive Appendice
Selection Ever Produced

Written by
Andrew C. Thompson

This Book Contains Information
Never Released Anywhere Before
And Is Heavily Based And
Expanded On Mapping The Atari - Revised

TYNE & WEAR
ATARI
USER GROUP

T.W.A.U.G.
P.O.Box No.8
Wallsend
TYNE & WEAR
NE28 6DQ