# CHAPTER 3: SOFTWARE

## 1. INTRODUCTION

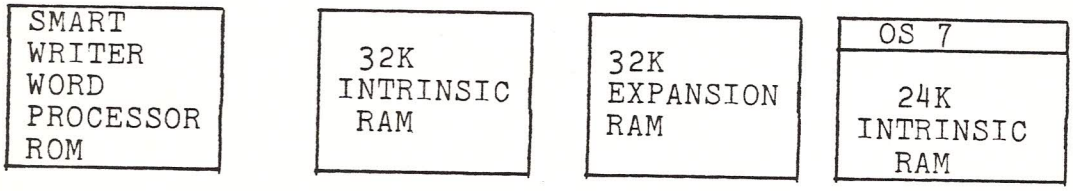
This chapter presents technical information on each of ADAM's software components. Emphasis is placed on AdamNet and the available operating systems. Descriptions of application software are provided as examples to software developers.

Source code listings for EOS and OS_7 can be requested with the form on the last page of this manual.

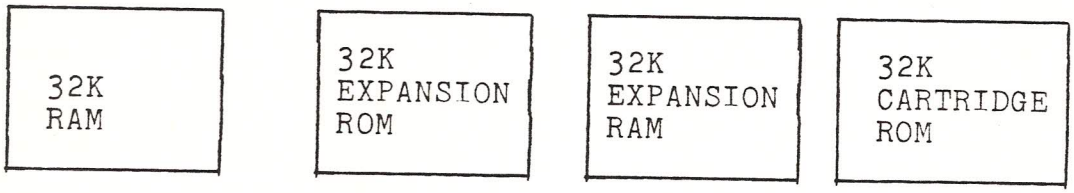
## 2. MEMORY MAP AND POWER UP/RESET PROCEDURE

Adam's Z80 microprocessor can address 64K bytes at any one time. The 64K addressable memory space is divided into two 32K sections. Each section may contain one of four memory options. Any one option for lower memory and any one option for upper memory can be selected for the full 64K memory space via port 7FH.

Memory Options for 0 - 7FFFH (lower memory)




Memory Options for 8000H - FFFFH (upper memory)




### 2.1 Lower Memory Options

SmartWRITER Word Processor ROM - This memory option consists of 32K of SmartWRITER ROM code. A small part of this code, EOS_BOOT, is responsible for system initialization during power up and reset. EOS ROM can also be accessed when this option is selected. See Subsection 4.1, EOS, for further details.

32K Intrinsic RAM - This option is the lower half of the 64K RAM included with every ADAM. DMA transfers to AdamNet can take place only in intrinsic RAM. SmartBASIC and most programs stored on data pack reside in this memory.

32K Expansion RAM - This option is the lower half of the 64K Memory Expander, an optional feature not included with the ADAM system. The 64K Memory Expander increases ADAM's memory to 144K of read/write memory (64K intrinsic, 64K expansion, 16K VRAM).

OS_7 and 24K Intrinsic RAM - This option contains OS_7 and 24K of ADAM's intrinsic RAM. OS_7 is the 8K ROM installed in ColecoVision and ADAM. In Expansion Module #3, this ROM is in the ColecoVision. The description of the 32K Intrinsic RAM also applies to this 24K intrinsic RAM.

## 2.2 Upper Memory Options

32K Intrinsic RAM - This option is the upper half of the 64K intrinsic RAM included with ADAM. DMA transfers to AdamNet can take place only intrinsic RAM. SmartBASIC and most programs stored on data pack reside in this memory.

Expansion ROM - This memory is provided by an expansion ROM, an optional feature not included in the ADAM system. The expansion ROM is installed in Connector #2 on the Memory and I/O Board. EOS_BOOT checks this connector for valid data before initializing EOS. If valid data is found, the EOS_BOOT code jumps to this ROM.

32K Expansion RAM - This is the upper half of the optional Expansion RAM described for lower memory.

32K Cartridge ROM - This memory option is the cartridge slot on ADAM or ColecoVision, used to execute game cartridges or other cartridge-based software.

## 2.3 Power Up/Computer Reset Procedure

When Adam powers up or when the computer reset switch is pressed, the MIOC selects SmartWRITER in lower memory and Intrinsic RAM in upper memory. EOS_BOOT executes this procedure:

Check for Expansion ROM. If Expansion ROM exists, jump to Expansion ROM.

Else, initialize EOS and jump to EOS_START

Check for the presence of devices in this order:

Disk Drive 1
Disk Drive 2
Data Pack Drive 1
Data Pack Drive 2

If a device exists, check for valid data in the device.

Boot and execute code from the first valid device found by loading block 0 to address 0C800H then jumping to 0C800H.

If no valid data is found on any device, execute SmartWRITER.

## 2.4 Z80 I/O Port Assignments

| Port | Description |
|---|---|
| 00H through 1DH | Reserved |
| 1EH | Optional Auto Dialer |
| 1FH | Reserved |
| 20H through 3EH | Reserved |
| 3FH* | Network reset; EOS enable |
| 40H through 4EH | Reserved |
| 4FH | Expansion connector #2 |
| 50H through 5DH | Reserved |
| 5EH | Optional Modem Data I/O |
| 5FH | Optional Modem Control Status |
| 60H through 7EH | Reserved |
| 7FH | Memory Map Control |
| 80H through FFH | Reserved for ColecoVision use |

*Net reset - The net reset function is performed by setting bit 0 and then resetting bit 0.

*EOS enable - Setting bit 1 enables the EOS ROM. Resetting bit 1 disables EOS ROM. The EOS enable function only affects the SmartWRITER ROMs. To access the EOS ROM, the SmartWRITER ROMs must be selected.

For further details on port assignments, see PORT_COLLECTION in the EOS Source Code Listing.

## 2.5 Memory Map Control

Software can select the memory configuration by writing to Port 7FH. Data bits D0 and D1 select the lower (0 - 7FFFH) memory option. D2 and D3 select the upper ( 8000H - FFFFH) memory option. D4 through D7 are reserved for future expansion, and should remain as 0. The value to be written to Port 7FH is obtained from the following tables.

Lower Memory Option Selection

| | | |
|---|---|---|
| 0 | 0 | SmartWRITER ROM and EOS ROM |
| 0 | 1 | 32K Intrinsic RAM |
| 1 | 0 | 32K Expansion RAM |
| 1 | 1 | OS7 + 24K Intrinsic RAM |

Upper Memory Option Selection

| | | |
|---|---|---|
| 0 | 0 | 32K Intrinsic RAM |
| 0 | 1 | Expansion ROM |
| 1 | 0 | Expansion RAM |
| 1 | 1 | Cartridge ROM |

## 2.6 Reset Procedures

Adam can be reset in either computer mode or in game mode. When the computer reset switch is pressed, Adam resets to computer mode, according to the power up procedure described in Subsection 2.3.

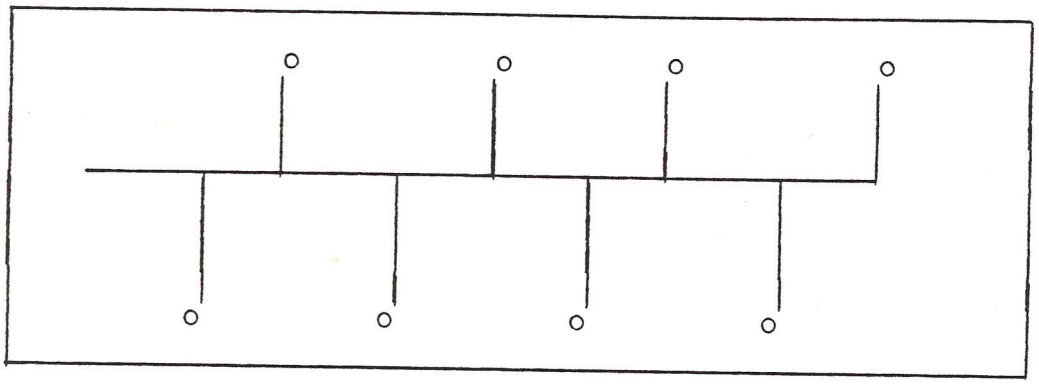
When the cartridge (or ColecoVision) reset switch is pressed, Adam resets to game mode. 32K of Cartridge ROM are switched into upper memory. OS-7 plus 24K of intrinsic RAM are switched into the lower bank of memory. Refer to Subsection 9.1 for the game mode memory map.

# 3. ADAMNET

## 3.1 Introduction

AdamNet is based physically on a shared bus, single master topology, although logically it resembles a token-passing network. The physical design of the network is depicted in Figure 3-1.

FIGURE 3-1: BUS NETWORK



The circles represent nodes (for example, printers or keyboards), which connect to a shared bus, represented by the horizontal line. The network is a four wire bus. The wire definitions are:

Data
Reset
Signal ground
Power

The network resides in all Adam components. The reset line behaves as a master reset signal to every node attached to the network. A transition on this line causes all attached devices to enter the power-on state, allowing the master to bring sanity to the network during periods of erratic behavior.

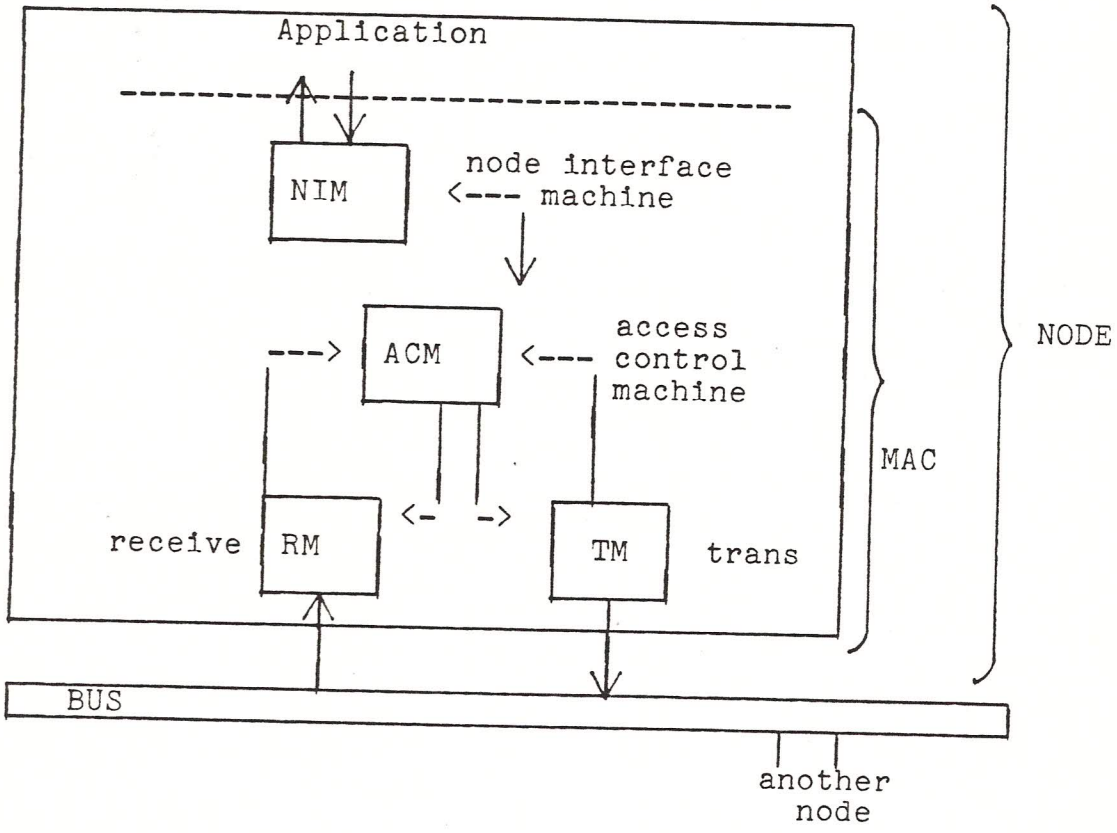
### Logical Design

Logically, AdamNet resembles a token-passing network. In a token-passing network, the right to talk on the bus is passed from node to node. This right, or "token", allows the node to access the bus and send messages. One node functions as the master of the bus. The master gives the token to other nodes with a regulated frequency.

## 3.2 Network Master Concept

Conceptually the master is like any other node on the network. Each node contains an IEEE defined Medium Access Control (MAC) layer of software that enables the node to gain access to the network. The MACs in non-master nodes contain a subset of the functions in the master's MAC. Internally all MACs are composed of four loosely coupled logical machines: Node Interface Machine (NIM), Access Control Machine (ACM), Receive Machine (RM) and Transmit Machine (TM).

FIGURE 3-2: MAC Internals




The MAC's version of the Access Control Machine is significantly different from that of the non-master nodes. The NIM, RM and TM components in the master and non-master nodes are basically equivalent. The RM and TM are physical level I/O routines that accept and send data frames.

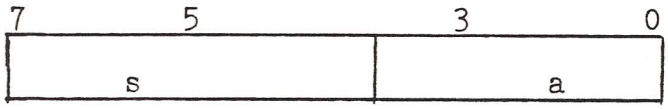
## 3.3 Message Philosophy and Definition

AdamNet messages can be divided into two types: commands and responses. Each type can be subdivided into either data or

control sub-types. Commands are messages initiated by the master, while responses are initiated by non-master nodes. The following four possibilities exist:

- command.control
- command.data
- response.control
- response.data

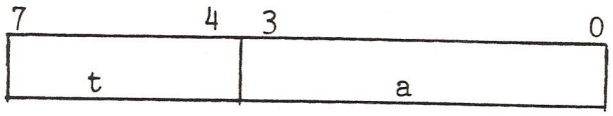
The format of a basic message follows:




Where: a = device address
s = subtype (indicates the nature of the message)

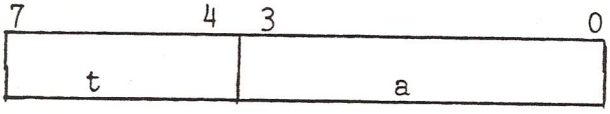
The messages are defined in the following pages.

3.3a. COMMAND.CONTROL (RESET)



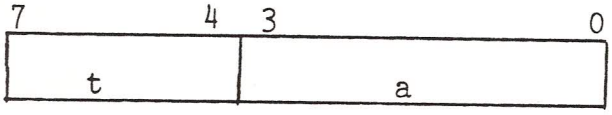

a = target address (1..15)
t = RESET (0)

3.3b. COMMAND.CONTROL (STATUS)



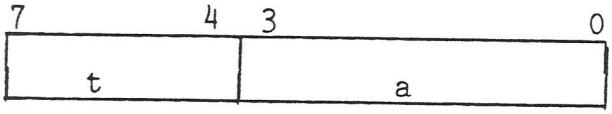

a = target address (1..15)
t = STATUS (1)

3.3c. COMMAND.CONTROL (ACK)



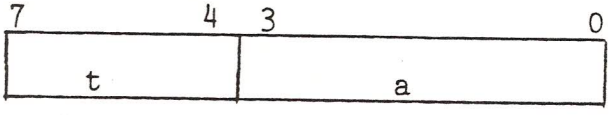

a = target address (1..15)
t = ACK (2)

3.3d. COMMAND.CONTROL (CLR)



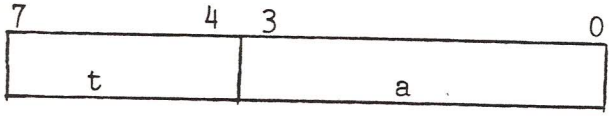

a = target address (1..15)
t = CLEAR (3)

3.3e. COMMAND.CONTROL (RECEIVE)



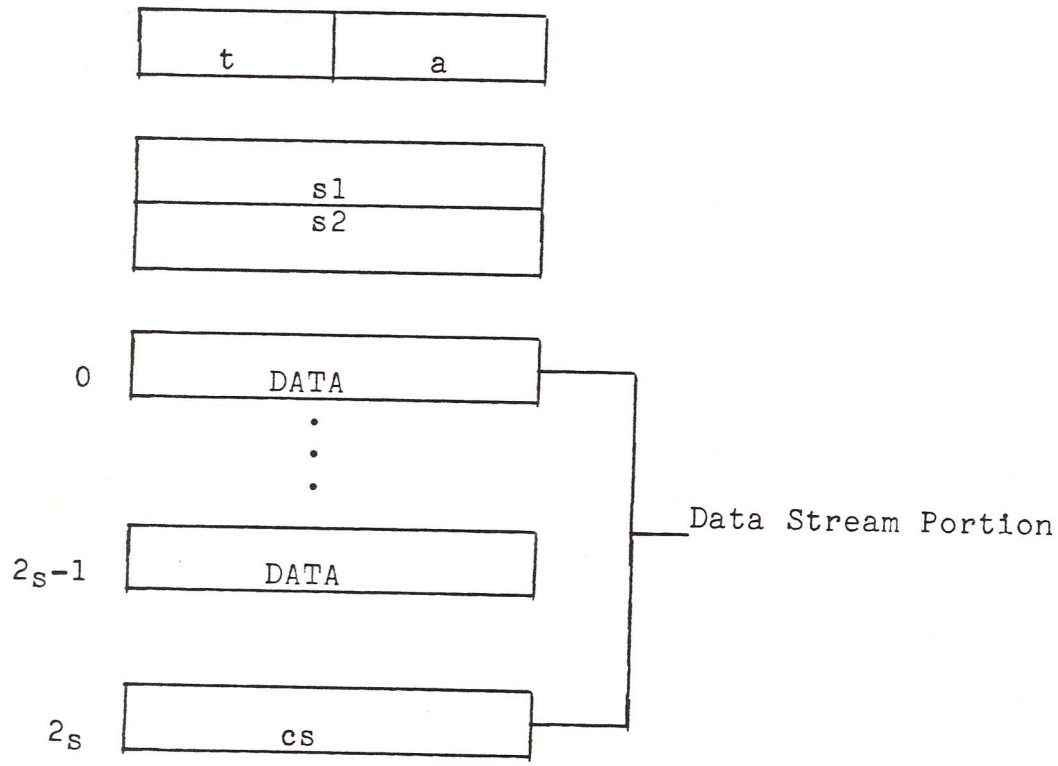

a = target address (1..15)
t = RECEIVE (4)

3.3f. COMMAND.CONTROL (CANCEL)



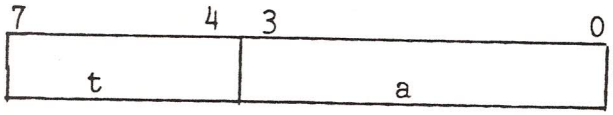

a = target address (1..15)
t = CANCEL (5)

3.3g. COMMAND.DATA (SEND)



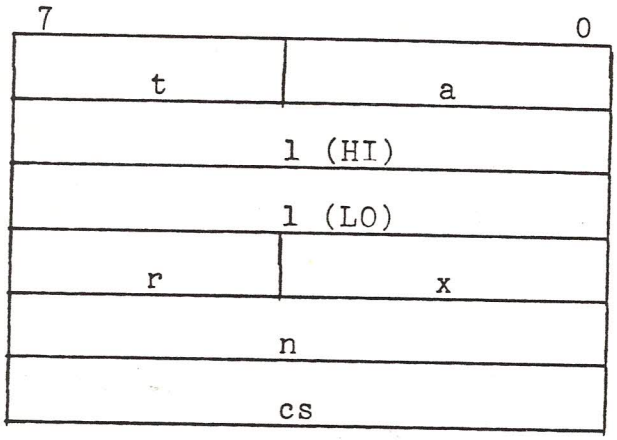

a = target address (1..15)
t = SEND (6)
s1,2 = MAX message size.
DATA = data byte
cs = check sum

3.3h. COMMAND.CONTROL (NACK)



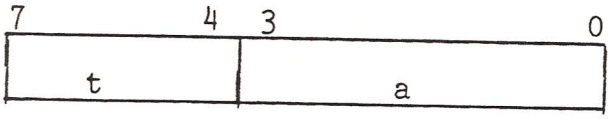

a = target address (1..15)
t = NACK (7)

3.3i. RESPONSE.CONTROL (STATUS)



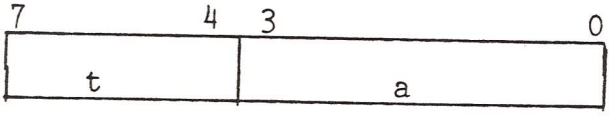

a = source address (1..15)
t = STATUS (8)
x = transmit code: 0 - character mode (0-255)
1 - block mode
r = reserved
n = device deependent status
l = MAX message size
cs = check sum of bytes 1 - 4

3.3j. RESPONSE.CONTROL (ACK)



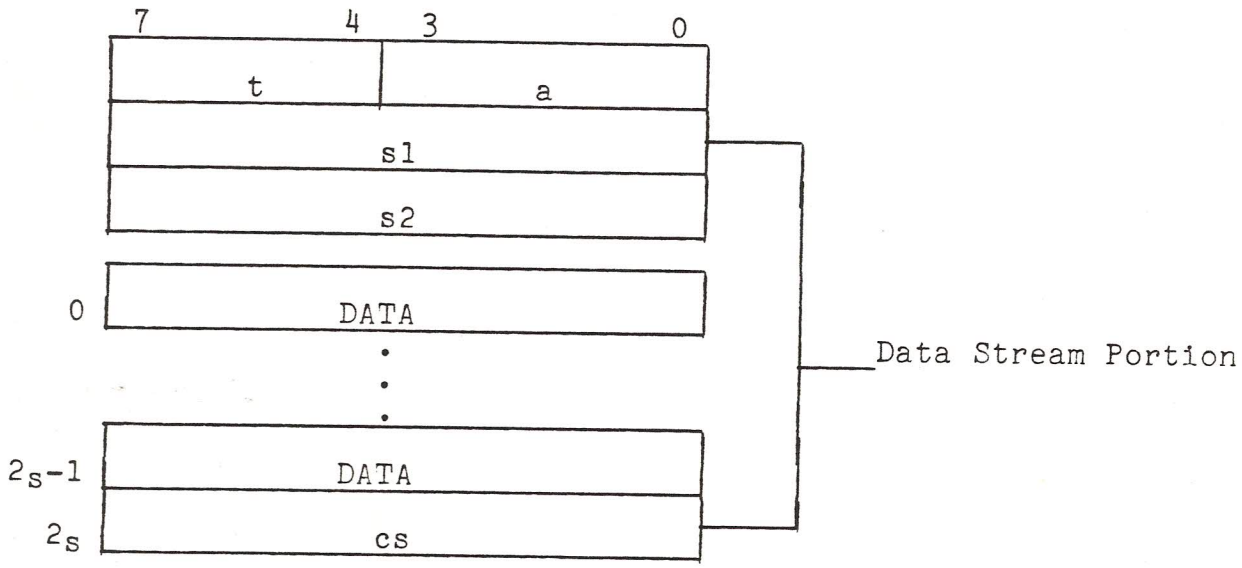

a = source address (1..15)
t = ACK (9)

3.3k. RESPONSE.CONTROL (CANCEL)



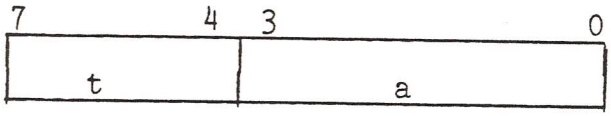

a = source address (1..15)
t = CANCEL (10)

3.31. RESPONSE.DATA (SEND)




a = source address (1..15)
t = SEND (11)
s1,2 = MAX message size.
DATA = data byte
cs = check sum

3.3m. RESPONSE.CONTROL (NACK)

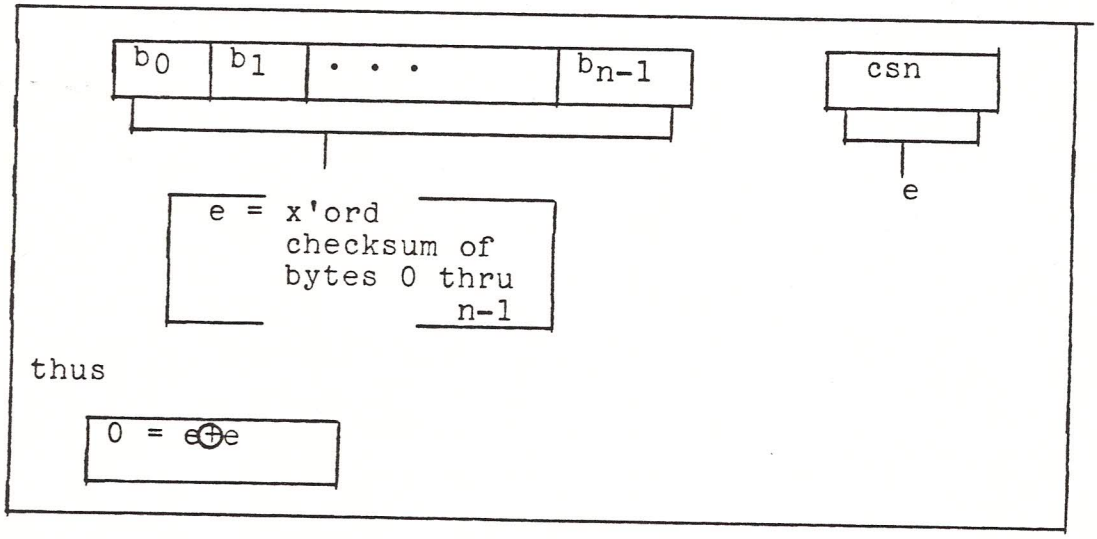



a = source address (1..15)
t = NACK (12)

## 3.4 Error Control

Data messages use an exclusive-or checksum to verify the integrity of the data. More specifically, for a n-1 byte data stream, an nth byte is added which is equal to the exclusive or'd checksum of the zero'th through the n-1 byte of the data stream. This insures that the checksum (exclusive or'd) of the zero'th through the nth byte equals zero.

FIGURE 3-3: ERROR CONTROL




A retry method is implemented by the Master to determine whether the lack of a response from a node is due to either a busy or failure condition.

## 3.5 Class of Service Concept

Each node is assigned a class of service. Class of service is defined as:

maximum message length
node type

To accept a device onto the network the CPU queries the device with a "COMMAND.CONTROL (STATUS)." In turn, the target device responds with a "RESPONSE.CONTROL (STATUS)" which entails the delivery to the host of a multi-byte package. This package defines to the host the nature and attributes of the responding device.

## 3.6 Power Up/Initialization

The CPU asks each attached device for its class of service message at power up. The CPU is responsible for associating

a logical unit address to each physical port on the bus. See EOS for device assignments.

## 3.7 Link Speed

The baud rate is 62.5k. This translates to a bit cell time of 16us. Thus, the time to transmit one byte (including a start and stop bit) is 160us.

## 3.8 Master Node Interface

AdamNet communicates with the Z80 through a message passing system. The Z80 RAM contains a group of data structures called DCBs (Device Communication Blocks). Every device on AdamNet is associated with a DCB. The structure of a DCB follows:

```
DCB_CMD_STAT        EQU     0
DCB_BA_LO           EQU     1
DCB_BA_HI           EQU     2
DCB_BUF_LEN_LO      EQU     3
DCB_BUF_LEN_HI      EQU     4
DCB_SEC_NUM_0       EQU     5
DCB_SEC_NUM_1       EQU     6
DCB_SEC_NUM_2       EQU     7
DCB_SEC_NUM_3       EQU     8
DCB_DEV_NUM         EQU     9
*reserved           EQU     10
*reserved           EQU     11
*reserved           EQU     12
*reserved           EQU     13
DCB_RETRY_LO        EQU     14
DCB_RETRY_HI        EQU     15
DCB_ADD_CODE        EQU     16
DCB_MAXL_LO         EQU     17
DCB_MAXL_HI         EQU     18
DCB_DEV_TYPE        EQU     19
DCB_NODE_TYPE       EQU     20
```

A DMA mechanism allows the 6801 master to scan the DCBs for pending work. The 6801 master writes completion messages and data to the DCBs or data areas as indicated in DCB messages. The Z80 either deposits commands into the DCBs, or inspects the DCBs for the completion status of previously issued commands.

A single data structure (PCB) allows the Z80 to communicate with the 6801 master.

## 3.9 Functional Overview - Z80 and 6801 Master

The following table provides a functional overview of the Z80 and 6801 master. The three pages following the table provide further details.

| Layer | Responsibilities |
|---|---|
| Z80 | Administer PCB, DCB, and data blocks<br>Signal 6801_Master_App<br>Scheduling and prioritizing |
| 6801_Master_App | Scan DCBs (Events from Z80)<br>Post I/O to 6801Mac<br>Send control signals<br>Collect signals<br>Interpret PCB and DCBs |
| 6801_Master_Mac | Control network<br>Write/read data to/from Z80. MEM based on NIM_BLK |

Z80

Inputs

1. Processor_Control_Block (to function #3)
2. Device_Control_Block (to function #3)
3. Memory Blocks (to function #1)

Functions

1. To manage data for PCB, DCBs and Memory Blocks.
2. To send signals to the 6801_MASTER_APP.
   a. Processor Commands
   b. Device Commands
3. To receive signals from the 6801_MASTER_APP.
   a. Process_Status
   b. Device_Status
4. To administer the priority of scheduling devices by regulating the frequency within which the Z80_MASTER posts commands to the applicable DEVICE_CONTROL_BLOCK_COMMAND.

Outputs

1. PCB (from functions #4, 2)
2. DCB (from functions #4, 2)
3. MEM BLOCKS (from function #4)

Note: Z80_MASTER must allocate the MAX message length buffer for the sending/receiving node's message.

## 6801 Master App

### Inputs

1. PCB (to functions #2, 3)
2. DCBs (to function #1, 3)
3. NIM_BLKS (to function #4)

### Functions

1. To scan the DCBs for changes of state to the DCB(i). DEVICE_COMMAND field.
2. To scan the PCB.
3. To interpret the scanned information.
4. To receive I/O completion signals from the MAC. (Deposited in M_SIG).
5. To post I/O requests to the 6801_MASTER_MAC.
6. To post processor status information to the Z80_MASTER.
7. To post DEVICE_STATUS information to the Z80_MASTER.

### Outputs

1. PCB (from function #6)
2. DCB (from function #7)
3. NIM_BLK (from function #4)

## 6801 Master Mac

### Inputs

1. NET_IN MESSAGES (to functions #1, 3)
2. NIM_BLK (to functions #1, 2, 5)

### Functions

1. To sequence the I/O of the network.
2. To send messages to devices.
3. To receive messages from devices.
4. To signal status information to the 6801_MASTER_APP.
5. To execute 6801_MASTER_APP signals.

### Outputs

1. NET_OUT MESSAGES (to functions #1, 2)

2. NIM_BLK (to functions #1, 4)

# 4. OPERATING SYSTEM

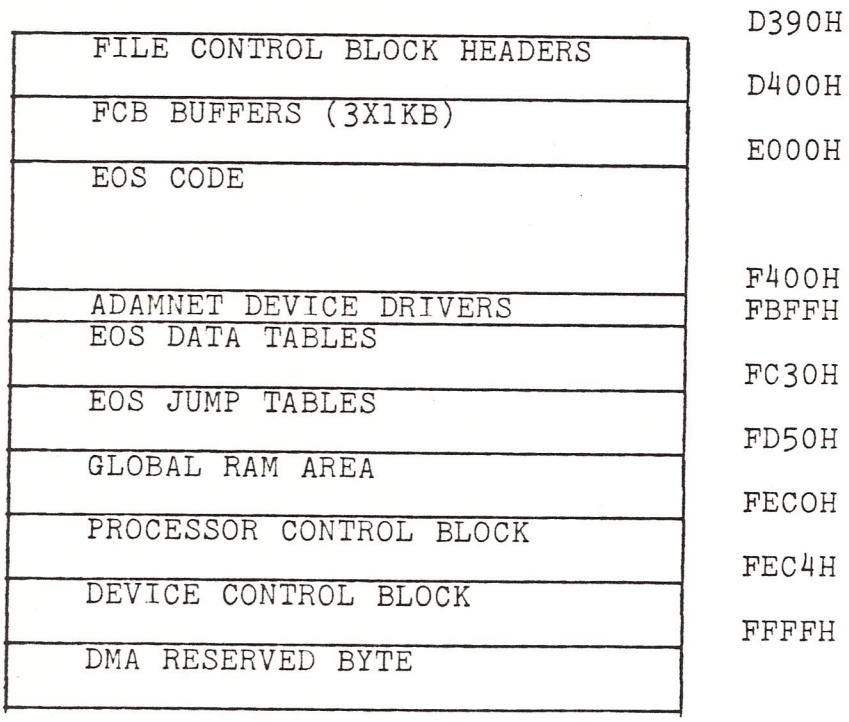
## 4.1 EOS (Elementary Operating System)

This section provides information to allow designers to write programs that operate in an Adam/EOS environment. It describes the organization of the Elementary Operating System (EOS) including file management and executive calls.

The Elementary Operating System is a collection of service routines that provide input/output facilities to peripheral devices. Programs accessing these devices need not be concerned with the physical characteristics of the devices, because EOS logically resides between the physical devices and application programs. EOS shields application programs from the details of AdamNet.

Entries to EOS are defined in Subsection 4.1.7. Application programs should use only these entry points. Access to EOS in any other manner is dangerous and may cause program malfunction. EOS error codes are listed in Subsection 4.1.8.

FIGURE 3-4: EOS MEMORY MAP



### 4.1.1 EOS Overwrite Addresses

Some applications may not require all of the routines provided by EOS. To accommodate this situation, EOS has three defined starting locations. Applications should always access the routines from the defined jump table locations.

Location 0D390H is the lowest EOS starting address. File manager software is located in this part of EOS. If the application program uses file manager software, the application should not overwrite 0D390H or above.

Location 0E000H. If the application does not use the file manger software, it can overwrite EOS up to 0DFFFH. Routines that access the lower level network device drivers, and the device drivers for the VDP, controllers and sound generator are available.

Locations 0F400H to 0FC30H contain the lowest level device drivers for AdamNet. The application can extend up to 0F3FFH if it is using OS_7 or its own drivers for the VDP, controllers and sound generator.

### 4.1.2 EOS Files

The file structure takes advantage of the sequential design of the data pack. (For more information on tapes and tape formats, see Chapter 3, Section 5.) A file directory keeps track of the location of files on tape, and other pertinent information such as size, creation date, and protection attributes. Some programs, such as super games, do not require directories.

Data is stored on tape in blocks of 1K (1024 bytes). An EOS file occupies a number of contiguous blocks, allocated upon file creation. EOS files can be any number of blocks but; are limited by the physical storage space available.

Block 0 is reserved for a cold start loader. Block 1 and up, depending on the directory size, contain the directory.

The EOS file manager accesses and controls a file through a File Control Block (FCB) maintained in RAM. The FCB contains static and dynamic information about the file. The FCB is created when a file is opened and destroyed when the file is closed. Application programs are limited to two FCBs at any one time. Each FCB is 36 bytes long. A data buffer, 1024 bytes long, is associated with each FCB.

## The Directory

The directory contains two types of records: Volume and File Records. The first 26 bytes of the first block in the directory is the Volume Record. The Volume Record is followed by a number of File Records, also 26 bytes in length. The 13th byte of the Volume Record determines the size of the directory. Up to 127 blocks may be allocated for the directory, allowing a maximum of 4953 file entries, and 39 directory entries per 1K block.

| Size (bytes) (12) | (1) | (4) | (4) | (2) | (3) |
|---|---|---|---|---|---|
| VOL_NAME | DIRECTORY SIZE | DIR_CHECK | VOL_SIZE | reserved | CREATION DATE |

Volume Record

| Size (bytes) (12) | (1) | (4) | (2) | (2) | (2) | (3) |
|---|---|---|---|---|---|---|
| DIR_NAME | DIR_ ATTRIB | DIR_START BLOCK | DIR_MAX LENGTH | DIR_USED LENGTH | DIR_LAST COUNT | CREATION DATE |

File Record

| Name | Length | Explanation |
|---|---|---|
| VOL_NAME | 12 bytes | Logical Volume Name |
| DIRECTORY SIZE | 1 byte | |
| VOL_ATTR | Bit 7 | Delete protection if set to 1 |
| VOL_DIRSIZE | Bits 6-0 | Directory size in blocks |
| DIR_CHECK | 4 bytes | A unique code that indicates the presence of a directory. |
| VOL_SIZE | 4 bytes | Total number of blocks allocated |
| CREATION DATE | 3 bytes | (1) VOL_YEAR; (1) VOL_MONTH; (1) VOL_DAY |
| DIR_NAME | 12 bytes | file name. 12th byte denotes file type (A, a, H or h). Terminated by ETX (03H). (See 4.1.3) |

DIR_ATTRIB 1 byte File attribute byte

Bits:

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| p | w | r | u | s | a | e | f |

p: permanently protected bit
w: write protected
r: read protected
u: user file
s: system file
a: file has been deleted
e: execute protect
(0 = execute
(1 = no execute)
f: not a file

A typical user-defined file would have a DIR-ATTTRIB of 10H. An executable file, which is loaded and run from 100H, has a DIR_ATTRIB of 0C8H.

DIR_START_BLOCK 4 bytes Starting block number of the file

DIR_MAX_LENGTH 2 bytes File size in blocks

DIR_USED_LENGTH 2 bytes Number of blocks used, including any partially used blocks

DIR_LAST_COUNT 2 bytes Number of bytes in the last block. (0 - 400)

CREATION DATE 3 bytes

| DIR_ YR | DIR_MNTH | DIR_DAY |
|---|---|---|
| | | |

## File Control Block (FCB) Organization

| # bytes | | |
|---|---|---|
| 12 | FCB_NAME | Up to eleven characters, free format |
| 1 | FCB_ATTR | Same as attribute byte of File Record |
| 4 | FCB_START_BLOCK | File starting block number on tape |
| 2 | FCB_MAX_LENGTH | Total number of blocks allocated |
| 2 | FCB_USED_LENGTH | Number of blocks used within the file |
| 2 | FCB_LAST_COUNT | Number of bytes in last block of the file |
| 1 | FCB_DEVICE | Device containing the file. See below. |
| 1 | FCB_MODE | File open modes. See below. |
| 4 | FCB_BLOCK | Block number currently in FCB_BUFFER |
| 4 | FCB_LAST_BLOCK | Last block number of the file |
| 2 | FCB_POINTER | FCB_BUFFER pointer |
| 1 | FCB_LENGTH | Data block size of the current FCB_DEVICE |
| 1024 | FCB_BUFFER | 1K buffer allocated for each file currently opened. Matches the block size on the digital data pack. Controlled by the EOS file manager to transfer data between the user program and mass storage devices. |

FCB_DEVICE

A unique number assigned to all devices (present and future):

00H - Master
01H - Keyboard
02H - Printer
03H - Reserved
04H - Floppy Disk Drive - 1
05H - Floppy Disk Drive - 2
06H - Reserved
07H - Reserved
08H - Data Pack Drive - 1
18H - Data Pack Drive - 2
09H - Reserved
0AH - Reserved
0BH - Reserved
0CH - Reserved
0DH - Parallel Interface Module

0EH - RS-232C Interface Module
0FH - Gateway

### 4.1.3 File Types and Headers

The 12th character of DIR_NAME indicates the file type. Valid file types are A, a, h or H. The lower case file types indicate backup versions of a file. These files have the user file attribute bit set. The first three bytes of an "H" file define the length of the header and the application code. Following these bytes are the header and then ASCII information. Refer to Subsection 6.2 for an example of an "H" file.

The application code defines the format of the header. The following application codes have been assigned:

| | |
|---|---|
| 1 | SmartWRITER |
| 2 | SmartBASIC |
| 16 | Electronic FlashCard Maker |

"A" files have the user bit set, but do not contain headers. ASCII information begins in the first byte of the file.

### 4.1.4 EOS Executive Calls

This subsection summarizes EOS executive subroutines that can be called from application programs. The executive calls fall into three categories: system operations, simple device I/O and mass storage file I/O.

#### System Operations

| | |
|---|---|
| _EOS_START | Starts EOS by going through initialization steps |
| _HARD_INIT | Initializes the system at powerup |
| _SOFT_INIT | Initializes the system anytime but powerup |
| _HARD_RESET_NET | Applies hardware reset (38H) to AdamNet |
| _SYNC | Synchronizes Z80 with Master 6801 |
| _SCAN_ACTIVE | Scans devices on AdamNet and establishes the Device Control Blocks (DCB) |
| _RELOC_PCB | Relocates PCB and DCBs to different addresses after powerup if necessary |
| _FIND_DCB | Finds the DCB address for the assigned device |
| _GET_DCB_ADDR | Same as _FIND_DCB |
| _GET_PCB_ADDR | Finds the current PCB address of the system |
| _REQUEST_STATUS | Issues a status request command to a device |

## Simple Device Operations

| | |
|---|---|
| _CONS_INIT | Initializes the system console |
| _CONS_OUT | Displays a character or performs screen control on the system console |
| _REQ_KBD_STAT | Requests keyboard status |
| _REQ_PR_STAT | Requests printer status |
| _RD_CH_DEV | Indicates a read command to a character device |
| _RD_KBD | Reads a character from the keyboard |
| _START_RD_CH_DEV | Sets up a character device DCB to issue read command (concurrent operation) |
| _END_RD_CH_DEV | Checks the status of character device DCB after command has been sent to read (concurrent) |
| _START_RD_KBD | Starts a keyboard read command (concurrent) |
| _END_RD_KBD | Checks the keyboard status (concurrent) |
| _WR_CH_DEV | Initiates a write command to a character device |
| _PR_CH | Prints a character on the printer |
| _PR_BUFF | Prints string of ASCII characters terminated by ETX (03) |
| _START_WR_CH_DEV | Sets up a character device DCB to issue write command (concurrent) |
| _END_WR_CH_DEV | Checks the status of character device DCB after command has been sent to write (concurrent) |
| _START_PR_CH | Starts a printer print command (concurrent) |
| _END_PR_CH | Checks the printer status (concurrent) |
| _START_PR_BUFF | Starts the command of printing a string of characters (concurrent) |
| _END_PR_BUFF | Checks the printer status (concurrent) |

## Mass Storage Device Operations

### File Manager Section

| | |
|---|---|
| _SET_DATE | Sets the current date |
| _GET_DATE | Gets the current date |
| _QUERY_FILE | Reads directory entry of a file |
| _SET_FILE | Sets the file directory entry |
| _MAKE_FILE | Creates a file in the directory |
| _OPEN_FILE | Opens a file by setting up the FCB |
| _CLOSE_FILE | Closes a file by marking the FCB |
| _RESET_FILE | Resets the file by pointing back to the first byte |
| _READ_FILE | Reads data from a file into user's buffer |
| _WRITE_FILE | Writes data to a file |
| _FMGR_INIT | Initializes the file manager |
| _SCAN_FOR_FILE | Scans the directory block(s) for a file |

## Device Driver Section

| | |
|---|---|
| _RD_1_BLOCK | Reads a block of data (1024 bytes) from mass storage device |
| _WR_1_BLOCK | Writes a block of data to mass storage device. I/O buffers containing data to be transferred to tape can reside anywhere in Z80 RAM. |
| _REQ_TAPE_STAT | Requests status of the data pack drive |
| _START_RD_1_BLOCK | Sends read command to read a block of data from a block device (concurrent) |
| _END_RD_1_BLOCK | Checks status after _START_RD_1_BLOCK |
| _START_WR_1_BLOCK | Sends write command to write a block of data to a block device (concurrent) |
| _END_WR_1_BLOCK | Checks status after _END_WR_1_BLOCK |

### 4.1.5 EOS Routines Adapted from OS_7

A section of EOS contains device drivers for the video processor, sound generator and controllers. Many of these routines are functionally the same as routines in the OS_7 ROM. The inputs and outputs for the EOS version of these routines is not necessarily the same as those for equivalent routines in OS_7. Entry points are defined in Subsection 4.1.7. The file A_uOS_00 in the EOS Source Code specifies parameter passing.

Routines for the Video Processor

| | |
|---|---|
| WRITE_VRAM | READ_VRAM |
| WRITE_REGISTER | READ_REGISTER |
| FILL_VRAM | INIT_TABLE |
| PUT_VRAM | GET_VRAM |
| CALC_OFFSET | PX_TO_PTRN_POS |
| LOAD_ASCII | PUT_ASCII |
| WR_SPR_ATTRIBUTE | |

Routines for the Controllers

| | |
|---|---|
| DECODER | POLLER |
| SPINNER | |

Routines for the Sound Processor

| | |
|---|---|
| DECLSN | DECMSN |
| MSNTOLSN | ADD8TO16 |
| SOUND_INIT | TURN_OFF_SOUND |
| PLAY_IT | SOUNDS |
| EFFECT_OVER | |

Adam Routines for OS_7 Access

| | |
|---|---|
| SWITCH_MEM | PORT_COLLECTION |

### 4.1.6 Initializing EOS

No initializaion of EOS is necessary if an application program boots from disk or data pack. If a program boots from the cartridge slot or a connector, EOS must be loaded from ROM to its executable location, according to the following procedure. This procedure must be executed from address range 8000H through 0DFFFH in intrinsic RAM.

Step 1: Select the SmartWRITER option in lower memory.

```
out 7FH,00H
```

Step 2: Strobe the EOS_ENABLE line by writing the value 02H to port 3FH.

```
out 3FH,02H
```

Step 3 Copy EOS from location 6000H to location 0E000H. Do not overwrite the DCBs.

```
LD HL,6000H
LD DE,0E000H
LD BC,0FEC0H - 0E000H
LDIR
```

Step 4: Deselect EOS ROM. EOS ROM should be deselected if the SmartWRITER ROMS will be accessed by the application program. LOAD_ASCII is an example of an EOS routine which accesses the EOS ROM.

```
out 3FH,0
```

Step 5: Initialize EOS tables. All RAM should be cleared to 0. See Subsection 4.1.7 for the equate values.

```
LD  BC,CLEAR_RAM_SIZE
LD  DE,CLEAR_RAM_START+1
LD  HL,CLEAR_RAM_START
XDR A
LD  [HL],A
LDIR
```

Step 6: Initialize I/O ports. EOS collects port values from the OS_7 ROM. EOS routines use these ports when they access the video processor, controllers and sound generator.

```
CALL  PORT_COLLECTION
```

Step 7: Initialize AdamNet. _HARD_INIT performs the 6801/Z80 synchronization, performs a roll call poll on the network, and establishes the DCBs.

```
CALL _HARD_INIT
```

## 4.1.7 EOS Entry Points

```
ADD816              EQU 0FD4DH ;P
BLK_STRT_PTR        EQU 0FDDCH ;D
BLOCKS_REQ          EQU 0FE0CH ;D
BUF_END             EQU 0FE0AH ;D
BUF_START           EQU 0FE08H ;D
BYTES_REQ           EQU 0FE02H ;D
BYTES_TO_GO         EQU 0FE04H ;D
CALC_OFFSET         EQU 0FD32H ;P
CLEAR_RAM_SIZE      EQU 00147H ;A
CLEAR_RAM_START     EQU 0FD60H ;D
COLORTABLE          EQU 0FD6CH ;D
CONTROLLER_0_PO     EQU 0FC2BH ;A
CONTROLLER_1_PO     EQU 0FC2CH ;A
CURRENT_DEV         EQU 0FD6FH ;D
CURRENT_PCB         EQU 0FD70H ;D
CURSOR              EQU 0FEA5H ;D
CUR_BANK            EQU 0FD6EH ;D
DCB_IMAGE           EQU 0FD8BH ;D
DECLSN              EQU 0FD44H ;P
DECMSN              EQU 0FD47H ;P
DEFAULT_BT_DEV      EQU 0FD6FH ;D
DEVICE_ID           EQU 0FD72H ;D
DIR_BLOCK_NO        EQU 0FDD9H ;D
EFFECT_OVER         EQU 0FD5CH ;P
EOS_DAY             EQU 0FDE2H ;D
EOS_MONTH           EQU 0FDE1H ;D
EOS_STACK           EQU 0FE58H ;D
EOS_YEAR            EQU 0FDE0H ;D
FCB_BUFFER          EQU 0FDBAH ;D
FCB_DATA_ADDR       EQU 0FDFFH ;D
FCB_HEAD_ADDR       EQU 0FDFDH ;D
FILENAME_CMPS       EQU 0FDD8H ;D
FILE_COUNT          EQU 0FDD4H ;D
FILE_NAME_ADDR      EQU 0FD73H ;D
FILE_NUMBR          EQU 0FDD7H ;D
FILL_VRAM           EQU 0FD26H ;P
FMGR_DIR_ENT        EQU 0FDE3H ;D
FNUM                EQU 0FE01H ;D
FOUND_AVAIL_ENT     EQU 0FDDBH ;D
GET_VRAM            EQU 0FD2FH ;P
INIT_TABLE          EQU 0FD29H ;P
INT_VCTR_TBL        EQU 0FBFFH ;A
KEYBOARD_BUFFER     EQU 0FD75H ;D
LINEBUFFER_         EQU 0FE7EH ;D
LOAD_ASCII          EQU 0FD38H ;P
MEM_CNFG00          EQU 0FC17H ;A
MEM_CNFG01          EQU 0FC18H ;A
MEM_CNFG02          EQU 0FC19H ;A
MEM_CNFG03          EQU 0FC1AH ;A
MEM_CNFG04          EQU 0FC1BH ;A
MEM_CNFG05          EQU 0FC1CH ;A
MEM_CNFG06          EQU 0FC1DH ;A
MEM_CNFG07          EQU 0FC1EH ;A
MEM_CNFG08          EQU 0FC1FH ;A
MEM_CNFG09          EQU 0FC20H ;A
MEM_CNFG0A          EQU 0FC21H ;A
MEM_CNFG0B          EQU 0FC22H ;A
MEM_CNFG0C          EQU 0FC23H ;A
MEM_CNFG0D          EQU 0FC24H ;A
MEM_CNFG0E          EQU 0FC25H ;A
MEM_CNFG0F          EQU 0FC26H ;A
MEM_SWITCH_PORT     EQU 0FC27H ;A
MOD_FILE_COUNT      EQU 0FDD5H ;D
MSNTOLSN            EQU 0FD4AH ;P
NET_RESET_PORT      EQU 0FC28H ;A
NEW_HOLE_SIZE       EQU 0FE1AH ;D
NEW_HOLE_START      EQU 0FE16H ;D
NUM_COLUMNS         EQU 0FEA0H ;D
NUM_LINES           EQU 0FE9FH ;D
OLDCHAR_            EQU 0FE79H ;D
PATTRNGENTBL        EQU 0FD6AH ;D
PATTRNNAMETBL       EQU 0FD68H ;D
PCB                 EQU 0FEC0H ;A
PERSONAL_DEBOUN     EQU 0FE5AH ;D
PLAY_IT             EQU 0FD56H ;P
POLLER              EQU 0FD3EH ;P
PORT_COLLECTION     EQU 0FD11H ;P
PORT_TABLE          EQU 0FC27H ;A
PRINT_BUFFER        EQU 0FD76H ;D
PTRN_NAME_TBL       EQU 0FEA3H ;D
PTR_TO_LST_OF_S     EQU 0FE6EH ;D
PTR_TO_S_ON_0       EQU 0FE70H ;D
PTR_TO_S_ON_1       EQU 0FE72H ;D
PTR_TO_S_ON_2       EQU 0FE74H ;D
PTR_TO_S_ON_3       EQU 0FE76H ;D
PUT_ASCII           EQU 0FD17H ;P
PUT_VRAM            EQU 0FD2CH ;P
PX_TO_PTRN_POS      EQU 0FD35H ;P
QUERY_BUFFER        EQU 0FDA0H ;D
READ_REGISTER       EQU 0FD23H ;P
READ_VRAM           EQU 0FD1DH ;P
RETRY_COUNT         EQU 0FDD6H ;D
REV_NUM             EQU 0FD60H ;D
SAVE_CTRL           EQU 0FE78H ;D
SECTORS_TO_INIT     EQU 0FD86H ;D
SECTOR_NO           EQU 0FD87H ;D
SOUNDPORT           EQU 0FC2FH ;A
SOUNDS              EQU 0FD59H ;P
SOUND_INIT          EQU 0FD50H ;P
SPIN_SW0_CT         EQU 0FE58H ;D
SPIN_SW1_CT         EQU 0FE59H ;D
SPRITEATTRTBL       EQU 0FD64H ;D
SPRITEGENTBL        EQU 0FD66H ;D
START_BLOCK         EQU 0FE12H ;D
STROBE_RESET_PO     EQU 0FC2EH ;A
STROBE_SET_PORT     EQU 0FC2DH ;A
SWITCH_MEM          EQU 0FD14H ;P
SWITCH_TABLE        EQU 0FC17H ;A
TEMP_STACK          EQU 0FE6EH ;D
TURN_OFF_SOUND      EQU 0FD53H ;P
UPDATE_SPINNER      EQU 0FD41H ;P
UPPER_LEFT          EQU 0FEA1H ;D
USER_BUF            EQU 0FE06H ;D
USER_NAME           EQU 0FE10H ;D
VDP_CTRL_PORT       EQU 0FC29H ;A
VDP_DATA_PORT       EQU 0FC2AH ;A
VDP_MODE_WORD       EQU 0FD61H ;D
VDP_STATUS_BYTE     EQU 0FD63H ;D
VECTOR_08H          EQU 0FBFFH ;A
VECTOR_10H          EQU 0FC02H ;A
VECTOR_18H          EQU 0FC05H ;A
VECTOR_20H          EQU 0FC08H ;A
VECTOR_28H          EQU 0FC08H ;A
```

## 4.1.7 EOS Entry Points

```
VECTOR_30H          EQU 0FC0EH ;A
VECTOR_38H          EQU 0FC11H ;A
VECTOR_66H          EQU 0FC14H ;A
VOL_BLK_SZ          EQU 0FDDCH ;D
VRAM_ADDR_TABLE     EQU 0FD64H ;D
WRITE_REGISTER      EQU 0FD20H ;P
WRITE_VRAM          EQU 0FD1AH ;P
WR_SPR_ATTRIBUT     EQU 0FD38H ;P
X_MAX               EQU 0FE7BH ;D
X_MIN               EQU 0FE7AH ;D
Y_MAX               EQU 0FE7DH ;D
Y_MIN               EQU 0FE7CH ;D
_CHECK_FCB          EQU 0FCF0H ;P
_CLOSE_FILE         EQU 0FCC3H ;P
_CONS_DISP          EQU 0FC33H ;P
_CONS_INIT          EQU 0FC36H ;P
_CONS_OUT           EQU 0FC39H ;P
_CV_A               EQU 0FD0EH ;P
_DELETE_FILE        EQU 0FCE1H ;P
_DLY_AFT_HRD_RE     EQU 0FC3CH ;P
_END_PR_BUFF        EQU 0FC3FH ;P
_END_PR_CH          EQU 0FC42H ;P
_END_RD_1_BLOCK     EQU 0FC45H ;P
_END_RD_CH_DEV      EQU 0FC48H ;P
_END_RD_KBD         EQU 0FC4BH ;P
_END_WR_1_BLOCK     EQU 0FC4EH ;P
_END_WR_CH_DEV      EQU 0FC51H ;P
_EOS_1              EQU 0FD05H ;P
_EOS_2              EQU 0FD08H ;P
_EOS_3              EQU 0FD0BH ;P
_EOS_START          EQU 0FC30H ;P
_FILE_QUERY         EQU 0FCFFH ;P
_FIND_DCB           EQU 0FC54H ;P
_FMGR_INIT          EQU 0FCBAH ;P
_GET_DATE           EQU 0FCDBH ;P
_GET_DCB_ADDR       EQU 0FC57H ;P
_GET_PCB_ADDR       EQU 0FC5AH ;P
_GOTO_WP            EQU 0FCE7H ;P
_HARD_INIT          EQU 0FC5DH ;P
_HARD_RESET_NET     EQU 0FC60H ;P
_INIT_TAPE_DIR      EQU 0FCBDH ;P
_MAKE_FILE          EQU 0FCC9H ;P
_MODE_CHECK         EQU 0FCF9H ;P
_OPEN_FILE          EQU 0FCC0H ;P
_POSIT_FILE         EQU 0FD02H ;P
_PR_BUFF            EQU 0FC63H ;P
_PR_CH              EQU 0FC66H ;P
_QUERY_FILE         EQU 0FCCCH ;P
_RD_1_BLOCK         EQU 0FC69H ;P
_RD_DEV_DEP_STA     EQU 0FCE4H ;P
_RD_KBD             EQU 0FC6CH ;P
_RD_KBD_RET_COD     EQU 0FC6FH ;P
_RD_PR_RET_CODE     EQU 0FC72H ;P
_RD_RET_CODE        EQU 0FC75H ;P
_RD_TAPE_RET_CO     EQU 0FC78H ;P
_READ_BLOCK         EQU 0FCF3H ;P
_READ_EOS           EQU 0FCEAH ;P
_READ_FILE          EQU 0FCD2H ;P
_RELOC_PCB          EQU 0FC7BH ;P
_RENAME_FILE        EQU 0FCDEH ;P
_REQUEST_STATUS     EQU 0FC7EH ;P
_REQ_KBD_STAT       EQU 0FC81H ;P
_REQ_PR_STAT        EQU 0FC84H ;P
_REQ_TAPE_STAT      EQU 0FC87H ;P
_RESET_FILE         EQU 0FCC6H ;P
_SCAN_ACTIVE        EQU 0FC8AH ;P
_SCAN_FOR_FILE      EQU 0FCFCH ;P
_SET_DATE           EQU 0FCD8H ;P
_SET_FILE           EQU 0FCCFH ;P
_SOFT_INIT          EQU 0FC8DH ;P
_SOFT_RES_DEV       EQU 0FC90H ;P
_SOFT_RES_KBD       EQU 0FC93H ;P
_SOFT_RES_PR        EQU 0FC96H ;P
_SOFT_RES_TAPE      EQU 0FC99H ;P
_START_PR_BUFF      EQU 0FC9CH ;P
_START_PR_CH        EQU 0FC9FH ;P
_START_RD_1_BLO     EQU 0FCA2H ;P
_START_RD_CH_DE     EQU 0FCA5H ;P
_START_RD_KBD       EQU 0FCA8H ;P
_START_WR_1_BLO     EQU 0FCABH ;P
_START_WR_CH_DE     EQU 0FCAEH ;P
_SYNC               EQU 0FCB1H ;P
_TRIM_FILE          EQU 0FCEDH ;P
_WRITE_BLOCK        EQU 0FCF6H ;P
_WRITE_FILE         EQU 0FCD5H ;P
_WR_1_BLOCK         EQU 0FCB4H ;P
_WR_CH_DEV          EQU 0FCB7H ;P
```

### 4.1.8 EOS Error Codes

```
DCB_NOT_FOUND       EQU     1
DCB_BUSY            EQU     2
DCB_IDLE_ERR        EQU     3

NO_DATE_ERR         EQU     4
NO_FILE_ERR         EQU     5
FILE_EXISTS_ERR     EQU     6
NO_FCB_ERR          EQU     7
MATCH_ERR           EQU     8
BAD_FNUM_ERR        EQU     9
EOF_ERR             EQU     10
TOO_BIG_ERR         EQU     11
FULL_DIR_ERR        EQU     12
FULL_TAPE_ERR       EQU     13
FILE_NM_ERR         EQU     14
RENAME_ERR          EQU     15
DELETE_ERR          EQU     16
RANGE_ERR           EQU     17

CANT_SYNC1          EQU     18
CANT_SYNC2          EQU     19
PRT_ERR             EQU     20

RQ_TP_STAT_ERR      EQU     21
DEVICE_DEPD_ERR     EQU     22
PROG_NON_EXIST      EQU     23
NO_DIR_ERR          EQU     24
```

## 4.2 OS 7

### 4.2.1 Introduction

This subsection presents an overview of the ColecoVision Operating System, referred to as the OS or OS_7. In contrast to the typical definition of an operating system as a run-time executive, the ColecoVision OS is a run-time user's library that provides access to modules which control events related to graphics, sounds, timing, etc.

Appendix 3 contains the detail sections of the ColecoVision Programmer's Manual, which document each module. Included are the calling sequence, input/output parameters, side effects, and and calls to other OS routines. The documentation assumes that the programmer is familiar with the Z80 instruction set.

The ColecoVision OS provides power-up procedures and system-defined entry points for the user. It handles input/output operations and housekeeping functions. The OS provides the displayable ASCII character set and vectors into the cartridge memory space that correspond to the Z80 hardware restart and interrupt vectors.

The OS software can be divided functionally as follows:

- Graphics Generation
- Sound Generation
- Interrupt Handling and Write Deferral
- Timing
- Controller Interface
- Boot-up
- Miscellaneous Utilities
- Defined Reference Locations

### 4.2.2 Graphics Generation Software

Software in this area is divided into the chip driver level, the table level and the object level. First, there are chip drivers for the TMS 9928 level. Software to initialize and manipulate tables belongs to the table level. Video graphics are generated by objects on the object level. Each object has its own definition of tables, frames and display locations called out by the user.

### 4.2.3 Sound Generation Software

Sound generation software produces tones and music by table "look-up." The software also has provisions to produce special effects. A prioritization scheme allows important

sounds to overlay less important background sounds without destroying their continuity.

### 4.2.4 Interrupt Handling and Write Deferral Software

When the OS is accessing the VDP register or VRAM, protection may be needed when a VDP interrupt occurs. Interrupt deferral routines handle this situation when top-level graphics software is in use. Bulletin No. 10 in Appendix D of the ColecoVision Programmer's Manual lists additional interrupt deferral software to fix the problem at low-level VDP access.

### 4.2.5 Timing Software

In a real-time application, timing is essential to the system operation. The OS timing software manages the software timers allocated by the user. It allows the user to start the timer, update the timer, check if it is timed out and then relinquish the timer. The number of software timers that can be supported depends on available cartridge RAM.

### 4.2.6 Controller Interface

The controller interface has several features:

The process of scanning, debouncing and decoding controller inputs is automatic upon invoking the OS routines.

The software only debounces and decodes those inputs that the user wishes to access.

The user has the option of bypassing automatic scanning and decoding and accessing the raw controller data, if necessary.

### 4.2.7 Boot-Up Software

The OS performs certain initialization tasks, such as turning off the sound chip, initializing buffers and flags, and display of the standard logo screen. If no game cartridge is present, it warns the user to turn off the system before attempting to insert a cartridge or expansion module.

If a cartridge is present, the OS reads the cartridge title from a predefined location and displays it along with copyright information. This screen is displayed for a short time before the OS relinquishes control to the cartridge software.

### 4.2.8 Miscellaneous Utilities

This software includes some low-level utilities, evolved in conjunction with the graphics and sound packages. It inclu-

des a nibble arithmetic package and a routine that displays a standard game-option screen.

### 4.2.9 Defined Reference Locations

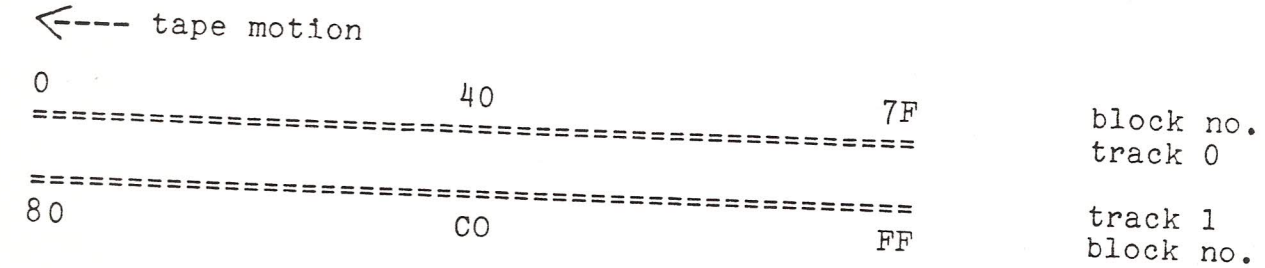
OS_7 has a number of system-defined reference locations in the areas of cartridge RAM, OS ROM and Cartridge ROM. The user-accessible locations are listed in the file, OS_SYMBOLS, Rev. 2 (Appendix F of the ColecoVision Programmer's Manual). It is IMPORTANT for the user to call the OS subroutines through those authorized entry points (Jump Table). The user also must be aware of the limitation of using cartridge RAM as a scratch pad due to the predefined OS data areas and stack memory. Locations at the beginning of the cartridge program may be accessed for pointers to the tables and buffers in CRAM. Also in this area, restart and interrupt vectors are defined for the user. Section X of the ColecoVision Programmers Manual lists all the system defined locations in all memory areas.

## 5. TAPE FORMAT AND OTHER TAPE CONSIDERATIONS

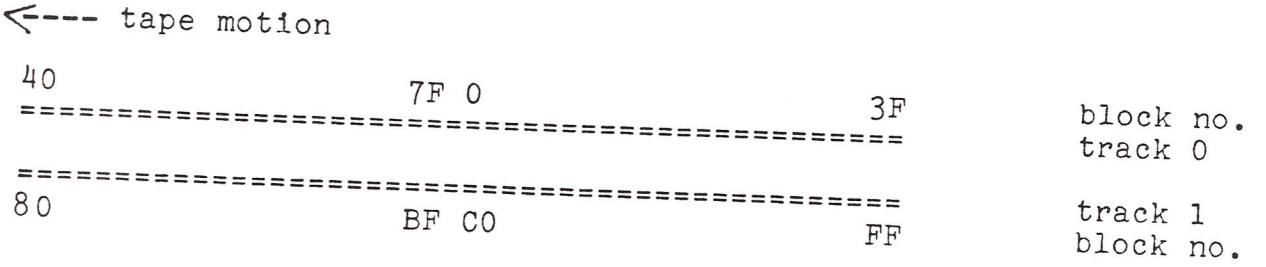
There are two types of tape format for ADAM data packs. Type GW tapes have block 0 at the end. Type HE tapes have block 0 in the middle. An example of a type GW is The Buck Rogers™ Plaent of Zoom™ Super Game. Both SmartBASIC and the blank data pack are examples of type HE tapes.

The capacity of the tape is 256K. Blocks are defined as 1K in length. There are two tracks, 128 blocks per track. Block numbers refer to the block of data on a data pack referenced by application software. The following illustration shows physical block positions.

FORMAT GW (Block 0 at the end)




Format HE (Block 0 in the middle)



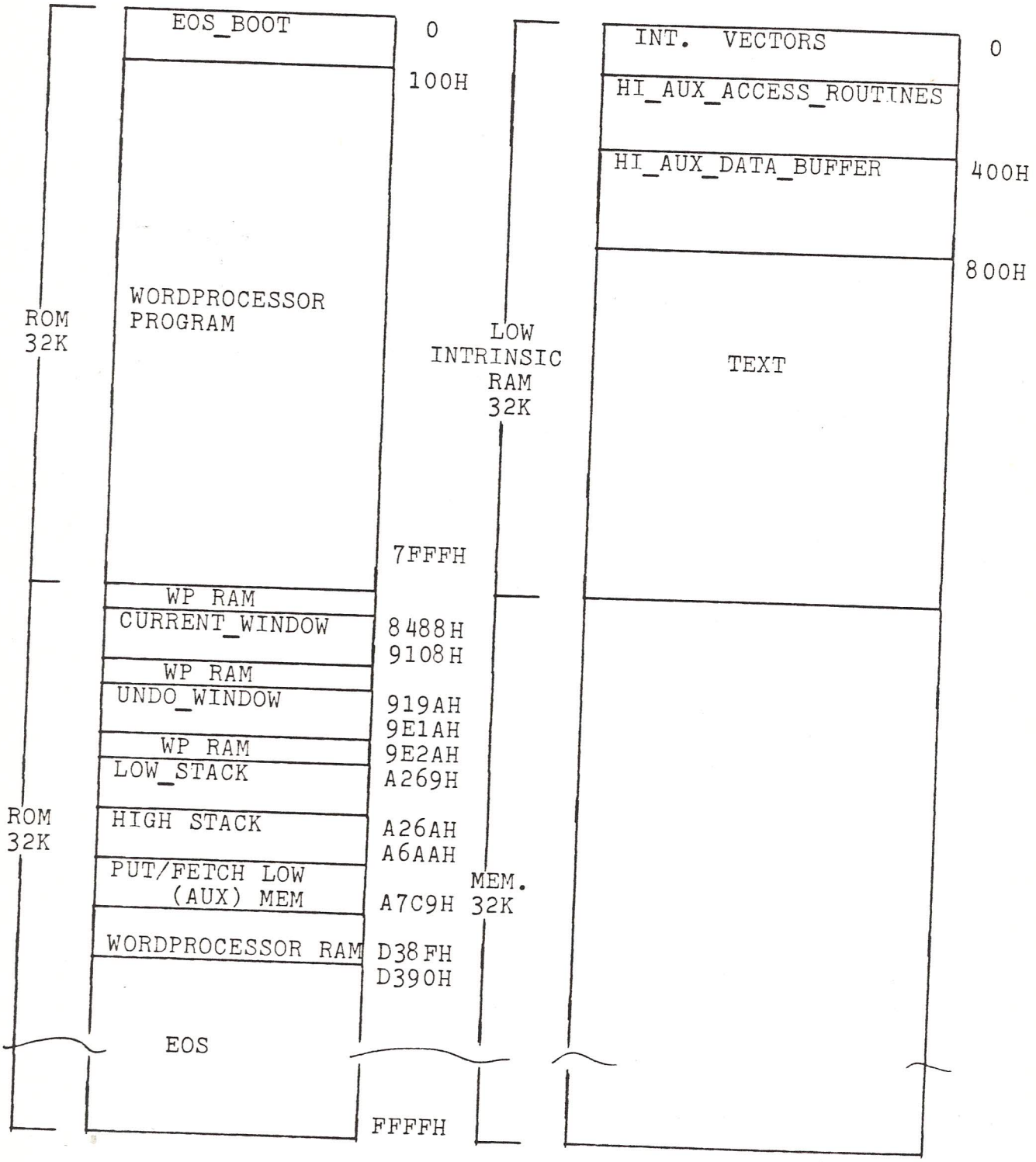

### Considerations in Choosing Tape Format

Type GW tapes do not contain a directory. Type GW tapes cannot be readily copied, so users are not able to easily copy one GW tape to another GW tape.

Type HE tapes are compatible with other type HE tapes. If a program needs to store information on a separate data pack, access or be accessed by other software, then a type HE tape is suggested.

# 6. SmartWRITER

## 6.1 Memory Map



## SmartWRITER-Compatible Files

Files must meet certain criteria to be compatible with SmartWRITER. SmartWRITER files have the user file attribute bit set, and the last character of the file name is H. SmartWRITER files start with a header and ASCII information begins after the header. The first two bytes of the file define the length of the header. The third byte contains the application code (1 for SmartWRITER). The format of the header is determined by the application code. Backup versions of SmartWRITER files have a lower-case "h" as the file type.

An example of a SmartWRITER file follows:

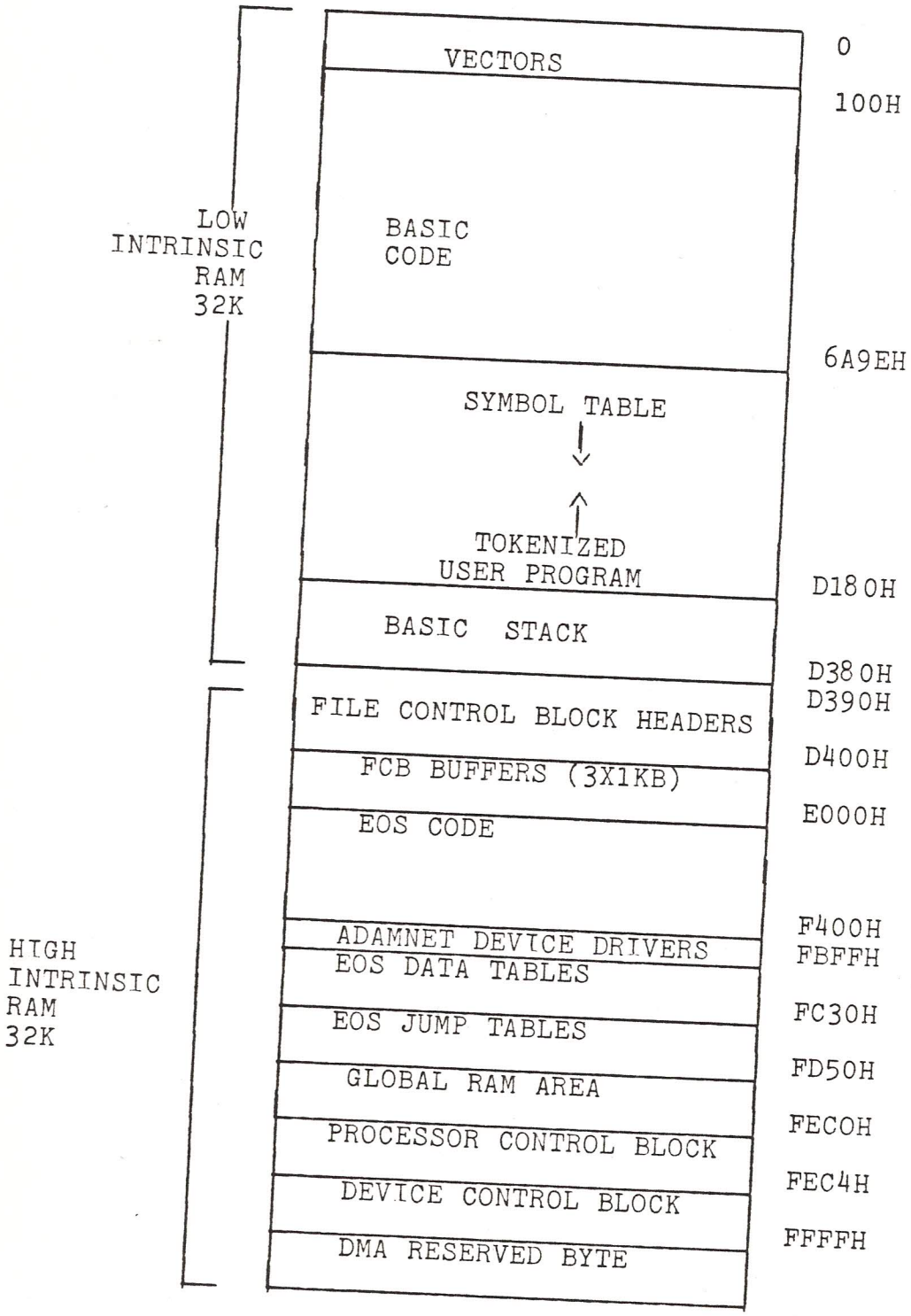
| BYTE | DESCRIPTION |
|---|---|
| 0 | HEADER_SIZE_LOW (=256) |
| 1 | HEADER_SIZE_HIGH (=0) |
| 2 | FILE_TYPE_CODE (=1) |
| 3 | TOP_MARGIN |
| 4 | BOTTOM_MARGIN |
| 5 | LEFT_MARGIN |
| 6 | RIGHT_MARGIN |
| 7 | LINE_SPACING |
| 8 | TAB_ARRAY (1) [0 = NO MRG, 1 = MRG] |
| 89 | TAB_ARRAY (80) [0 = NO MRG, 1 = MRG] |
| 90 | UNUSED |
| 258 | UNUSED |
| 259 | FIRST ASCII DATA BYTE |
| n | ASCII DATA |

## 7. SmartBASIC

SmartBASIC is generally source-code compatible with Applesoft BASIC. PEEKS and POKES, and other machine-dependent features of Applesoft BASIC are different in SmartBASIC. SmartBASIC graphics feature four modes:

Text mode - 24 lines of 31 characters
low resolution graphics - 40 x 40, with four lines of text
high resolution graphics - 256 x 160 with four lines of text
Pure high resolution graphics - 256 x 192

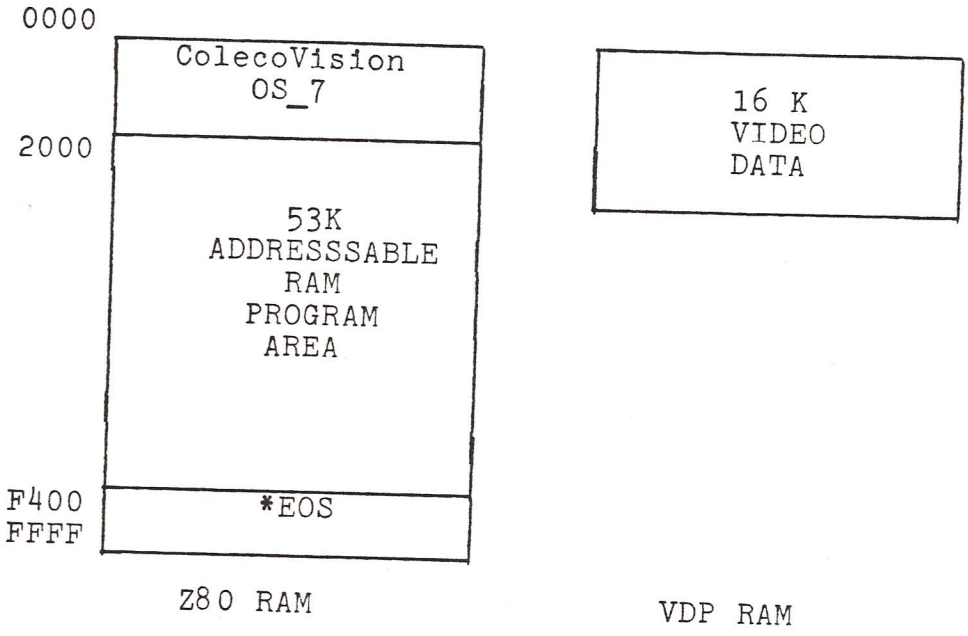
## 7.1 Memory Map



## 8. SUPER GAMES AND OTHER PROGRAMS USING OS7

Refer to Chapter 5, Section 2 for a detailed explanation of the Buck Rogers™ Planet of Zoom™ Super Game. The Buck Rogers game serves as an example for developers of super games.

### 8.1 MEMORY MAP




* ADAM EOS starts at D390. In this example of a super game, only the AdamNet device drivers in EOS are needed. Refer to Chapter 3, Subsection 4.1.1, EOS Overwrite Addresses.

Address 0000 through 1FFF

OS_7 ROM is available for program use and requires 8K.

Address F400 through FFFF

EOS is used to access peripherals and files, and requires 3K.

Address 2000 through F3FF

The remaining 53K of Z80 RAM is available for program and data storage. OS_7 uses RAM from 7000H through 73FFH.

Address 0000 through 3FFF VDP RAM

Unused portions of the 16K VDP RAM may be used as temporary storage. VDP RAM may not be loaded directly from tape, but must be read into intrinsic RAM, then transferred using one of the VDP access routines.

# 9. ROM-BASED CARTRIDGE PROGRAMS

## 9.1 Memory Map