INFORMATION FOR ADAM EXPLORERS

```
57538 E0C2  060A      LD    B,n
57540 E0C4  02        LD    (BC),A
57541 E0C5  0B        DEC   BC
57542 E0C6  04        INC   B
57543 E0C7  0603      LD    B,n
57545 E0C9  CDD5E0    CALL  nn     E0D5
57548 E0CC  C31AFD    JP    nn     FD1A
5755[illegible] E0[illegible]  CDD[illegible]  C[illegible]  [illegible]n     E0[illegible]
57[illegible] E0[illegible]  C3[illegible]FD    J[illegible]  [illegible]n     F[illegible]
575[illegible] E0[illegible]5  FDE[illegible]     P[illegible]SH  [illegible]Y
57559 E0D7  4F        LD    C,A                          O
57560 E0D8  FE04      CP    n
57562 E0DA  2008      JR    NZ,e
57[illegible] [illegible]0D[illegible]  [illegible]1F[illegible]  [illegible]D    A,(n[illegible])   F[illegible]6[illegible]    :a
57[illegible] [illegible]0D[illegible]  E6[illegible]2     [illegible]ND   n
57[illegible]9 E0E[illegible]  [illegible]4   [illegible]R    Z,[illegible]                          (
57571 E0E3  79        LD    A,C                          y
57572 E0E4  FE02      CP    n
57574 E0E6  280F      JR    Z,e                          (
[illegible]6 [illegible]E8  EB        EX    DE,HL
[illegible]7 E[illegible]  29        ADD   HL,H[illegible]                        )
[illegible]8 [illegible]EA  29        ADD   HL,H[illegible]                        )
57579 E0EB  B7        OR    A
57580 E0EC  2801      JR    Z,e                          (
57582 E0EE  29        ADD   HL,HL                        )
57583 E0EF  EB        EX    DE,HL
57584 E0F0  E3        EX    (SP),HL
57585 E0F1  29        ADD   HL,HL                        )
57586 E0F2  29        ADD   HL,HL                        )
57587 E0F3  2801      JR    Z,e                          (
57589 E0F5  29        ADD   HL,HL                        )
57590 E0F6  E3        EX    (SP),HL
57591 E0F7  79        LD    A,C                          y
57592 E0F8  0164FD    LD    BC,nn     FD64               d
57595 E0FB  E5        PUSH  HL
57596 E0FC  2600      LD    H,n                          &
57598 E0FE  6F        LD    L,A                          o
57599 E0FF  29        ADD   HL,HL                        )
57600 E100  09        ADD   HL,BC
57601 E101  7E        LD    A,(HL)
57602 E102  23        INC   HL                           #
57603 E103  66        LD    H,(HL)                       f
57604 E104  6F        LD    L,A                          o
57605 E105  19        ADD   HL,DE
57606 E106  EB        EX    DE,HL
57607 E107  E1        POP   HL
57608 E108  C1        POP   BC
57609 E109  C9        RET
57610 E10A  E5        PUSH  HL
57611 E10B  CB7A      BIT   7,D                          z
57613 E10D  2804      JR    Z,e                          (
57615 E10F  26FF      LD    H,n                          &
57617 E111  1802      JR    e
57619 E113  2600      LD    H,n                          &
57621 E115  6A        LD    L,D                          j
57622 E116  29        ADD   HL,HL                        )
57623 E117  29        ADD   HL,HL                        )
57624 E118  29        ADD   HL,HL                        )
57625 E119  29        ADD   HL,HL                        )
57626 E11A  29        ADD   HL,HL                        )
57627 E11B  CB7B      BIT   7,E
57629 E11D  2804      JR    Z,e                          (
57631 E11F  16FF      LD    D,n
57633 E121  1802      JR    e
```

Information for ADAM Explorers

by Peter Hinkle

Published by the author.
117 Northview Rd.
Ithaca N.Y. 14850
$9.95

607-555-1212
607- 273-1319

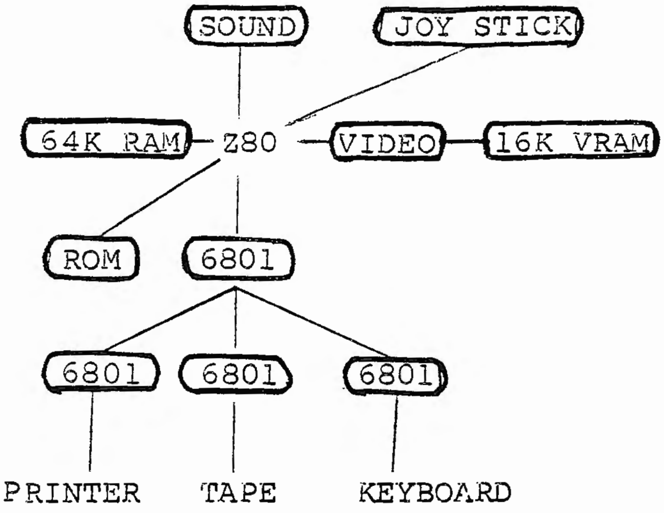
TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION

All home computer owners try to understand their machines and get the most out of them. Of course, some try harder than others. The increasing emphasis on fancy commercial programs does not really change things. For home use a microcomputer will always have a strong hobby element that inspires the budding hacker. Eventually many ADAM owners will discover that the best game for the ADAM is the ADAM itself, delving into the labyrinth of subroutines in RAM, making music, better graphics, cheap tapes, etc.

John Dvorak recently analysed the history of the microcomputer industry in InfoWorld, and concluded that the only factors unique to the two clearly successful machines, Apple and IBM, were complete documentation and encouragement of independent software and hardware developers. Thus it has been disappointing to ADAM owners that Coleco has not released a technical manual to the general public. I hope they still will, but meanwhile I got tired of waiting. This booklet is not a proper technical manual, but it will give owners a good start and some tools for exploring on their own. It is intended to be intelligible to people without technical training, but who have some familiarity with computers. The Z80 instruction set will be given, assuming that you do not have other books on the subject. The emphasis will be on how the major chips work, and how to analyze machine language using the disassembler. A circuit diagram may eventually be added to allow design of boards to plug into the bus connectors, but as of now I do not encourage people to mess with the hardware. It is too fragile. Pinouts of most chips in the ADAM are given in the last chapter, however.

A rudamentary outline of the ADAM circuitry is shown below. The Z80 microprocessor is the central processor, communicating with the 64K RAM via data and address buses, and with the sound, video and 6801 chips via the data bus and decoded lines in a special in/out address space.



The 6801 chips which run the printer, keyboard and tape are microprocessors of the Motorola 6800 family which have 128 bytes of RAM and 2K of ROM on the chip. The operating system and word processor are stored in ROM but the operating system, at least, is copied into the 64K RAM when BASIC is loaded, because it can be modified, indicating it is not in read only memory.

In addition to the obvious things that ADAM owners would like to know, such as how to make sounds and sprites, there are projects you could work on that are less obvious. Figure out how to control the tape drive directly so that files from the word processor can be read by BASIC and printed out with full justification, as a proper word processor should. A BASIC program can easily be written to insert extra spaces between words to make all lines the same length. Better yet, it may be possible to control the printer directly so that the spaces between letters can be changed to create proportional spacing. Such things are probably run by the 6801 in the printer, however, and are not accessible to the Z80. A problem for ADAM owners that others with BASIC in ROM do not have is that various versions of BASIC exist, and the memory map will depend on which version you have. If you buy another BASIC tape all addresses could be changed, although probably not by much. An advantage of having BASIC on tape, however, is that you can change it if you want (and can figure out how). In any case, there are many reasons to learn more about the ADAM, and I hope you find these notes and the disassembler useful in your explorations.

CHAPTER 2. Numbers

Several ways of representing numbers are used with computers, which may be a pain at first but is convenient. The numbers actually handled by the Z80 and stored in RAM are in binary (base 2), where 0 is represented by 0 to 0.5 volts and 1 is represented by 4 to 5 volts. Thus binary is the natural number system for computers because they have two states, just as decimal is the natural number system for us because we have ten fingers. Binary numbers are not used directly to program the ADAM, however, because they are quite awkward. Instead several number systems are used, called hexadecimal (base 16), two`s complement, and floating point, in addition to the usual decimal used in BASIC. The easiest way to convert numbers from binary to decimal or vice versa is to first convert binary to hexadecimal and then hexadecimal to decimal. Conversion of hexadecimal to decimal is done using the table or subroutine for programs shown later. Such subroutines are never there when you need them, however, and the best way to solve the number problem is to buy a hexadecimal-decimal calculator.

BINARY

The binary numbers in the ADAM are stored in 8 bit units called bytes. The digits represent powers of 2 (1,2,4,8,16,32,64,128), represented with the most significant bit (128) on the left and least significant bit (1) on the right. The first 16 numbers in decimal, binary (the lowest 4 bits), and hexadecimal are shown below.

decimal	binary	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A

11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Examples of 8 bit binary numbers are 178 = 10110010 = B2, 55 = 00110111 = $37, 239 =11101111 = EF, 17 = 00010001 = $11. Hexadecimal numbers are indicated by $ when necessary. Binary numbers are not used often by programmers except when certain bits have to be changed or when making shape tables (unless you use a shape-maker program).

Variables in BASIC that are specified as integers by following the name with % (eg. DIM A%(30)),are stored as 2 byte binary numbers, the least significant byte first. Thus the range of possible values is from 0 to FFFF, or 0 to 65,535 decimal. Strings of letters, numbers (0 to 9), and symbols are stored as one byte binary numbers which correspond to the letters etc. according to ASCII code (see the Coleco BASIC manual).

HEXADECIMAL

Hexadecimal representation is convenient when programing in machine language because each digit corresponds to 4 bits in binary, and a byte can always be represented by two hexadecimal digits. Furthermore, addresses in memory are often divided into pages of 256 bytes, and all 64K (65,535) bytes of RAM can be specified by four hexadecimal digits (0000 to FFFF). The problem comes, however, when BASIC is used, since all access to memory (PEEK and POKE) are in decimal. Conversions between hexadecimal and decimal can be made with the table below, finding the decimal number in the table from the first and second hexadecimal digits in the lefthand column and top row, respectively. The reverse conversion is also convenient. Four digit hexadecimal numbers can be easily converted to decimal by looking up the left two digits, multiplying the decimal equivalent times 256, and adding the result to the decimal equivalent of the right two digits.

Hexadecimal to decimal conversion.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
8	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
9	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
B	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
C	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
D	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
E	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
F	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

The table was generated by the following program in BASIC. The "NOT" statements are needed to line up the columns because the TAB command only works to 31, appropriate for the screen but not the printer. It is probably worth printing some of these tables so you can always have one handy.

```
  3 PR #1
  4 PRINT
  5 h$ = "0123456789ABCDEF"
  7 PRINT "    ";
 10 FOR x = 1 TO 16
 20 PRINT MID$(h$, x, 1); "    ";
 30 NEXT: PRINT
 40 FOR x = 1 TO 16
 50 PRINT MID$(h$, x, 1); "  ";
 60 FOR y = 1 TO 16
 65 PRINT " ";
 70 IF  NOT INT(n/100) THEN  PRINT " ";
 80 IF  NOT INT(n/10) THEN  PRINT " ";
 90 PRINT n; : n = n+1
100 NEXT y: PRINT: NEXT x
```

TWO'S COMPLEMENT BINARY.

This convention is used to represent positive and negative numbers in binary or hexadecimal, and is used for relative jumps on the Z80. Positive numbers 0 to 127 decimal (01111111 or 7F) are the same as usual for 8 bits. Negative numbers are made by pretending that the byte is the odometer on your car and driving backwards starting at zero. Thus -1 = 11111111, -2 = 1111110, etc. To complement a binary number means to change all the 1's to 0's and 0's to 1's. Doing just that is called 1's complement. 2's complement is 1's complement plus 1, and the 2's complement of a number from (decimal) 1 to 127 is the negative of the number. Thus in decimal 255 to 128 are negative numbers in this convention. This is logical because arithmatic in 2's complement works if you ignore the carry. For example, adding +9 and -2 gives +7.

$$\begin{array}{rr} +9 & 00001001 \\ \underline{-2} & \underline{11111110} \\ +7 & 00000111 \end{array}$$

Relative jumps on the Z80 are a little more complicated (as usual) because +2 is added to the offset before the jump.

FLOATING POINT

Numerical variables that are not followed by % are stored in floating point representation, which allows a wide range of values. It is similar to "scientific notation" of calculators or BASIC, with a mantissa times the number base to a power or exponent. For most practical purposes the scale can be regarded as continuous, but it is actually 2^{40} discrete numbers, half of which are between -1 and +1. Zero cannot be represented exactly. The mantissa can take values between 1/2 and (almost) 1, in binary 0.10000... and 0.11111..(the "." being the binary equivalent to a decimal point) ,positive or negative. The exponent is from 0 to 127, positive or negative. There are many different formats for the actual representation in RAM. On the ADAM the mantissa is four bytes and the exponent one byte with the following format. The mantissa bytes are stored in RAM in reverse order, with the least significant first. The most significant byte is strange in that the top bit (left) is assumed to be 1 for the purpose of calculating the number but is in fact used to specify the sign,1=-,0=+. The sign of the exponent is specified by the top bit (1=+, 0=-). thus $80=0, $81=1, $78=-2, etc. The following examples should make this clear. To try other numbers add a line to the printmem program which sets a variable to the number and then look on page 206 or 207 for the number in RAM (see BASIC chapter).

decimal	floating point (hex)	top 4 bits	decimal
1	00 00 00 00 81	1000	1/2 * 2^+1
2	00 00 00 00 82	1000	1/2 * 2^+2
3	00 00 00 40 82	1100	3/4 * 2^+2
4	00 00 00 00 83	1000	1/2 * 2^+3
5	00 00 00 20 83	1010	5/8 * 2^+3
6	00 00 00 40 83	1100	3/4 * 2^+3
7	00 00 00 60 83	1110	7/8 * 2^+3
8	00 00 00 00 84	1000	1/2 * 2^+4
9	00 00 00 10 84	1001	9/16* 2^+4
10	00 00 00 20 84	1010	5/8 * 2^+4
0.5	00 00 00 00 80	1000	1/2 * 2^+0
0.25	FF FF FF 7F 7E	1111etc.	1 * 2^-2
0.001	98 6E 12 03 77	-	- * 2^-9
100	00 00 00 48 87	11001	100/128*2^+7
-1	00 00 00 80 81	1000	-1/2 * 2^+1
-10	00 00 00 A0 84	1010	-5/8 * 2^+4
-0.25	FF FF FF FE 7E	1111	-1 * 2^-2

mantissa   exponent   always 1

top bit= sign

To translate a floating point number into hexadecimal, write it out in binary, set the top bit, and place the binary point. Then return to hexadecimal starting at the binary point. For example, the number in the floating point accumulator printed out by Printmem is: 00 00 90 7C 8E. Why?

Convert to binary:

```
  7    C     9     0    0etc
0111 1100  1001  0000  0000...
```

Set the top bit and place the point at 14 (8E):

```
1111 1100  1001  00.00 0000
3   F       2     4
```

3F24 is the address of the "90" byte of the number in RAM, so the FP accumulator held the address being PEEKed and was changing with each PEEK. Since only the "90" byte of the accumulator was changing during the program at that point, the accumulator was caught at the number of the "90" address.

CHAPTER 3. The Z80

The Z80 microprocessor is the central processing unit (CPU) of the ADAM. It steps along programs in RAM, executing simple machine language instructions, much as a calculator is programmed by pushing buttons. The machine language instructions are a series of 8 bit numbers that represent operations that move 8 bit numbers from one register to another, or add two 8 bit numbers, etc. For people to understand what is going on, these operations are usually represented in "assembly language", a series of mnemonics for the instructions which correspond to the machine language numbers. A program which takes mnemonics and turns them into machine language numbers is called an assembler. A program which takes machine language and turns it into mnemonics is called a disassembler. A disassembler, which is given in chapter 5, is useful to print out the machine language programs in the ADAM, which are BASIC and the operating system, in a form that is reasonable to understand. This chapter will give a brief outline of the Z80 which should be enough to allow understanding of a disassembly listing and simple machine language programming. If more advanced information is needed a complete book on the Z80 such as Rodnay Zaks' "How to program the Z80" should be consulted.

The Z80 has several registers, as shown below. The A register, or accumulator, is the central register and is used in most arithmetic operations. The F register contains flags, or bits that are set to 1 when certain results of operations occur. The flags are C,Z,P/V,S,N,H.

C=carry flag. C=1 on overflow of arithemitic operations.

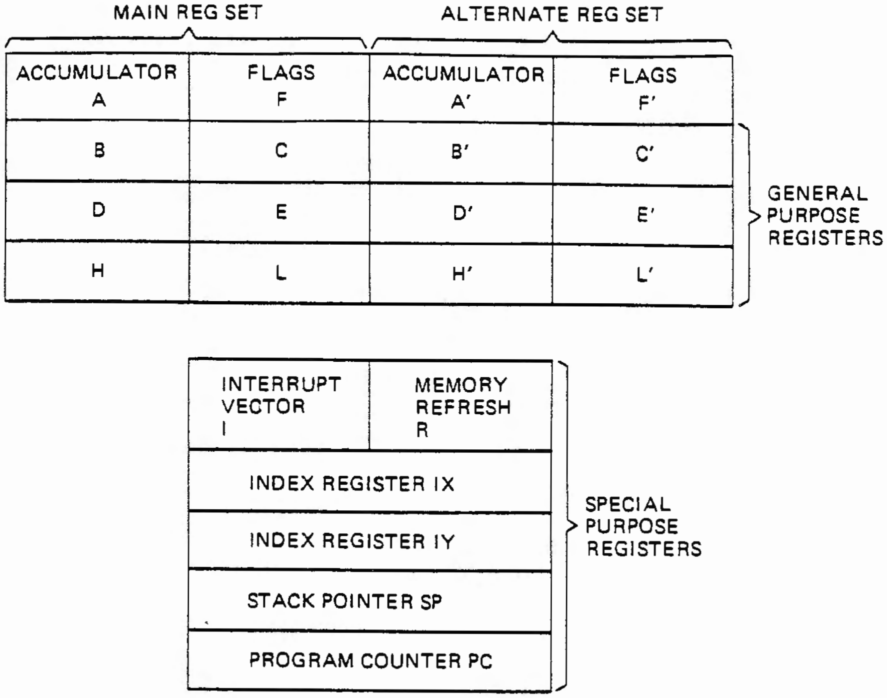
Z=zero flag. Z=1 if result of operation is zero.

S=sign flag. S=1 if the MSB of result is 1.

P/V= parity or overflow flag. For parity P/V=1 if the result is even, 0 if it is odd. For overflow, P/V=1 if operation produces overflow.

H=half carry flag. H=1 if add or subtract produce carry or borrow from bit 4 of the accumulator.

N=add/subtract flag. N=1 if the operation was subtract.



Z80 Registers

The B,C,D,E,H, and L registers are general purpose and are used individually as 8 bits in some instructions and in pairs (DE, BC, HL) as 16 bits in others. The I (interrupt vector) and R (memory refresh) registers are for special purposes and can be ignored for most applications. The IX and IY registers are 16 bit index registers that are used in some instructions to point to and step through tables etc. The SP (stack pointer) register points to the memory location that is the top of the stack, a last-in-first-out memory area similar to the stack in BASIC that stores addresses to return to after GOSUB's, etc. The PC(program counter) register points to the next location in memory for execution of machine language instructions. All of the special purpose registers (F,I,R,IX,IY,SP,PC) essentially take care of themselves in most short programs and can be ignored.

ADDRESSING MODES

The most complicated aspect of the Z80 is the addressing modes. The address in RAM or the Z80 registers can be specified in various ways. The following types of addressing are described and illustrated with examples. To understand the examples better it will probably help to look ahead where mnemonics are described. An important convention to understand is that if a register or number is enclosed in parentheses, eg. (HL) or (nn), then the number used is the number stored at the address in RAM given by the register or the number following the op code.

IMPLIED ADDRESSING

In this mode the address is implied by the instruction. Examples are " LD A,B" which copies the B register into the accumulator, and "AND H" which ands the H and A registers, the A register being implied.

IMMEDIATE ADDRESSING

In this mode the number to be used is specified in the machine code. Examples are "LD A,n" which copies the next number in RAM into the accumulator, and "LD HL,nn" which copies the 16 bit number nn into the HL register.

ABSOLUTE ADDRESSING

In this mode the address in RAM to be used is specified in the two bytes following the op code in machine language. Examples are "LD A,(nn)" which copies the contents of the memory location with address nn to the A register, and "JP nn" which jumps the program to address nn. The 8 bit numbers of the address are put in memory in reverse order with the low order byte before the high order byte. Thus the instruction "JP 34A8" in machine code is "C3 A8 34" (in hexadecimal).

RELATIVE ADDRESSING

In this mode the byte following the op code is a two's complement number which is added to the program counter + 2 to cause a relative jump. An example is "JR z,e", jump relative on result zero. Values of e from 0 to 7F cause a forward jump and values from 80 to FF cause a backward jump .

INDEXED ADDRESSING

In this mode the address is formed by adding the byte following the op code (called the displacement, or d) to the number in an index register (IX or IY). An example is "LD A,(IX+d)" which loads the number in RAM location specified by adding the contents of index register IX to the displacement d into the A register.

INDIRECT ADDRESSING

In this mode the address is the number in a 16 bit register pair (BC,DE, or HL). An example is "LD A,(BC)" which loads the contents of the memory location specified by the BC register into the A register.

BIT ADDRESSING

A single bit in a byte may be set to 1 (SET), reset to 0 (RES), or tested to set the zero flag (BIT). Various addressing modes may be used to specify the byte. Examples are "SET 3,(HL)", "RES 4,A" and "BIT 7,(IY+d)". The numbers after the mnemonic specify the bit to be acted upon.

INSTRUCTION SET

After addressing modes, all there is to learn about the Z80 is the instruction set mnemonics. A list of these with definitions follows.

ADC Add with carry two specified registers. 8 bit additions are made between the A register and any other register or memory location with the result left in the A register. 16 bit additions are between the HL register and other 16 bit registers with the result in HL. In each case the carry flag is added to the result and the carry flag is set if the result exceeds the size of the register.

ADD Add without carry. This instruction is similar to ADC except that the carry flag is not added to the result. The carry flag is set if the result exceeds the size of the register.

AND Logical "AND" the A register with the specified register, number or memory location. Logical AND gives a result where bits in binary are 1 only if they are 1 in both numbers. For example, in binary 10110001 AND 01101001 = 00100001, or in hexadecimal B1 AND 69 = 21, or in decimal 177 AND 105 = 33.

BIT tests the specified bit of the register or memory location addressed and sets the zero flag if the result is zero.

CALL Call subroutine. The program counter is stored on the stack and the address given after the CALL instruction is loaded into the program counter. CALLs may also be conditional.

CCF Complement (reverse) the carry flag.

CP Compare register or memory location with the accumulator. Sets zero flag if the numbers are equal.

CPD Compare with decrement. A is compared with the memory location specified by HL and HL and BC are decremented by 1. The zero flag is set if A = (HL).

CPDR Block compare with decrement. Like CPD but continues until a match is found (A = (HL)) or BC = 0.

CPI Compare with increment. Compares A with (HL), sets zero flag if equal, increments HL by 1 and decrements BC by 1.

CPIR Block compare with increment. Like CPI but continues until A = (HL) or BC = 0.

CPL Complement accumulator. All bits that are 1 are set to 0 and vice versa.

DDA Decimal adjust accumulator. Used in binary coded decimal arithmetic.

DEC Decrement register or memory.

DI Disable interrupts.

DJNZ Decrement B and jump relative on nonzero.

EI Enable interrupts.

EX Exchange specified registers.

EXX Exchange BC, DE, and HL registers with the alternative set.

HALT CPU executes NOP's until an interrupt or reset.

IM Set interrupt mode.

IN Input number to register from port specified by the C register,(C), or number, (n).

INC Increment register or memory location.

IND Input with decrement. Loads (HL) with input from (C), decrements B and decrements HL.

INDR Block input with decrement. Like IND but repeats until B = 0.

INI Input with increment. Loads (HL) with input from (C), increments HL and decrements B.

INIR Block input with increment. Like INI but repeats until B = 0.

JP Jump.

JR Jump relative.

LD Load or copy the contents of a register or memory location to another.

LDD Load with decrement. HL loaded to memory location (DE), DE, HL, and BC are decremented.

LDDR Block load with decrement. Like LDD but repeats until BC = 0.

LDI Load with increment. (HL) is copied to (DE), DE and HL are incremented and BC is decremented.

LDIR Block load with increment. Repeats LDI until BC = 0.

NEG Negate accumulator in two's complement.

NOP No operation. Fills in spaces in machine code and delays about 1 microsecond.

OR Logical OR accumulator with specified register. Logical OR acts on bits. For example,in binary, 10101100 OR 00010111 = 10111111. In hexadecimal, AC OR 17 = BF. In decimal, 172 OR 23 = 191 (same example each time). 1 OR 1, 1 OR 0,and 0 OR 1 all equal 1. 0 OR 0 = 0.

OTDR Block output with decrement. Like OUTD but repeated until B=0.

OTIR Block output with increment. Like OUTI but repeated until B=0.

OUT Output register specified to port given by the C register, (C), or number, (n).

OUTD Output with decrement. The memory location addressed by the HL register is outputted to the C port. The B and HL registers are decremented.

OUTI Output with increment. The memory location addressed by the HL register is outputted to port C. The HL register is incremented and the B register decremented.

POP Pop specified register (16 bit) from stack, as in BASIC.

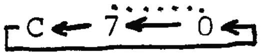
PUSH Push register (16 bit) to stack.

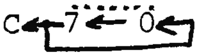
RES Reset. The specified bit is set to zero.

RET Return from subroutine. The program counter is popped from the stack, low byte, high byte.

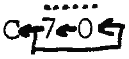
RETI Return from interrupt. Like RET.

RETN Return from non-maskable interrupt. Like RET.

RL Rotate register left through carry flag. 

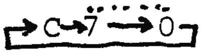
RLCA Rotate accumulator left with branch carry. 

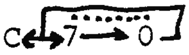
RLC Rotate register or memory location left with branch carry.



RLD Rotate left decimal (for BCD).

RR Rotate register or memory location right through carry flag.



RRC Rotate right with branch carry. 

RRD Rotate right decimal (for BCD).

RSTp Restart at location p*8 in zero page.

SBC Subtract with borrow.

SCF Set carry flag.

SET Set to 1 specified bit of register or memory.

SLA Arithmetic shift left. C←7←0←0 This multiplies the register or memory location by 2.

SRA Arithmetic shift right. ↳7→0→C

SRL Logical shift right. 0→7→0→C

SUB Subtract register specified from the accumulator, the result appearing in the accumulator.

XOR Exclusive OR accumulator and specified register. For example, in binary 10110100 XOR 10001110 = 00111010, or in hexadecimal B4 XOR 8E=3A, or in decimal 180 XOR 142 = 58. XOR A is used to set the accumulator to zero.

How do you use all these codes? To start with you hand assemble some machine language. Some people think you need an assembler to write machine language, but starting with an assembler would be like starting to write english with a word processor. Its unnecessarily complicated.

To illustrate a short machine language program I will show a way around the limitation in BASIC that POKE will not work above 54160. To POKE to higher memory the load commands of the Z80 work fine. In assembly language we write a subroutine as follows:

```
        LD A,n
        LD (nn),A
        RET
```

The code for LD A,n found in the alphabetical assembly language table that follows, is $3E (or 62 in decimal) followed by the 8 bit value of n. The code for LD (nn),A which loads the first n that is now in the accumulator into memory location nn, is $32 (or 50 in decimal). The code for RET (return from subroutine) is $C9 (or 201 in decimal). We can now POKE the decimal numbers into pokable memory as shown in the first five lines of the following program:

```
  5 REM  HIPOKER
 10 DATA 62,0,50,0,0,201
 20 FOR x = 0 TO 5
 30 READ d
 40 POKE 210+x, d
 50 NEXT
 60 INPUT "start address high byte"; adh
 70 INPUT "start address low byte"; alo
 80 INPUT "number"; n
 90 POKE 211,n :POKE 213, alo :POKE 214, adh
100 CALL 210
110 PRINT n; "   "; PEEK(adh*256+alo)
120 alo = alo+l
130 GOTO 80
```

In this case the program was stored in an unused part of zero page. You can put them anywhere they do not erase a necessary part of BASIC or the operating system (the copywrite statement and "hi Cathy" on page 4, for example). Most programs would be best in the same area as shape tables, above BASIC and below the stack (see pages C-16 and C-20 in the BASIC manual). Such an area must be reserved with a HIMEM command at the beginning of the BASIC program.

It is not necessary to PUSH registers on the stack at the beginning of a routine called from BASIC and POP them at the end, because the CALL routine does that for you.

The following table gives a complete list of op codes in alphabetical order which can be used for hand assembly of short machine language routines. The disassembler in this book could also be modified to be a simple assembler to look up op codes for you.

Z80 op codes (Courtesy of Zilog)   05=d, 8405=nn, 20=n, 2E=e

8E	ADC	A,(HL)
DD8E05	ADC	A,(IX+d)
FD8E05	ADC	A,(IY+d)
8F	ADC'	A,A
88	ADC	A,B
89	ADC	A,C
8A	ADC	A,D
8B	ADC	A,E
8C	ADC	A,H
8D	ADC	A,L
CE20	ADC	A,n
ED4A	ADC	HL,BC
ED5A	ADC	HL,DE
ED6A	ADC	HL,HL
ED7A	ADC	HL,SP
86	ADD	A,(HL)
DD8605	ADD	A,(IX+d)
FD8605	ADD	A,(IY+d)
87	ADD	A,A
80	ADD	A,B
81	ADD	A,C
82	ADD	A,D
83	ADD	A,E
84	ADD	A,H
85	ADD	A,L
C620	ADD	A,n
09	ADD	HL,BC
19	ADD	HL,DE
29	ADD	HL,HL
39	ADD	HL,SP
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD29	ADD	IX,IX
DD39	ADD	IX,SP
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD29	ADD	IY,IY
FD39	ADD	IY,SP
A6	AND	(HL)
DDA605	AND	(IX+d)
FDA605	AND	(IY+d)
A7	AND	A
A0	AND	B
A1	AND	C
A2	AND	D
A3	AND	E
A4	AND	H
A5	AND	L

E620	AND	n
CB46	BIT	0,(HL)
DDCB0546	BIT	0,(IX+d)
FDCB0546	BIT	0,(IY+d)
CB47	BIT	0,A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB4E	BIT	1 (HL)
DDCB054E	BIT	1,(IX+d)
FDCB054E	BIT	1,(IY+d)
CB4F	BIT	1,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB56	BIT	2,(HL)
DDCB0556	BIT	2,(IX+d)
FDCB0556	BIT	2,(IY+d)
CB57	BIT	2,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB5E	BIT	3,(HL)
DDCB055E	BIT	3,(IX+d)
FDCB055E	BIT	3,(IY+d)
CB5F	BIT	3,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB66	BIT	4,(HL)
DDCB0566	BIT	4,(IX+d)
FDCB0566	BIT	4,(IY+d)
CB67	BIT	4,A
CB60	BIT	4,B
CB61	BIT	4,C
CB62	BIT	4,D

CB63	BIT	4,E
CB64	BIT	4,H
CB65	BIT	4,L
CB6E	BIT	5,(HL)
DDCB056E	BIT	5,(IX+d)
FDCB056E	BIT	5,(IY+d)
CB6F	BIT	5,A
CB68	BIT	5,B
CB69	BIT	5,C
CB6A	BIT	5,D
CB6B	BIT	5,E
CB6C	BIT	5,H
CB6D	BIT	5,L
CB76	BIT	6,(HL)
DDCB0576	BIT	6,(IX+d)
FDCB0576	BIT	6,(IY+d)
CB77	BIT	6,A
CB70	BIT	6,B
CB71	BIT	6,C
CB72	BIT	6,D
CB73	BIT	6,E
CB74	BIT	6,H
CB75	BIT	6,L
CB7E	BIT	7,(HL)
DDCB057E	BIT	7,(IX+d)
FDCB057E	BIT	7,(IY+d)
CB7F	BIT	7,A
CB78	BIT	7,B
CB79	BIT	7,C
CB7A	BIT	7,D
CB7B	BIT	7,E
CB7C	BIT	7,H
CB7D	BIT	7,L
DC8405	CALL	C,nn
FC8405	CALL	M,nn
D48405	CALL	NC,nn
C48405	CALL	NZ,nn
F48405	CALL	P,nn
EC8405	CALL	PE,nn
E48405	CALL	PO,nn
CC8405	CALL	Z,nn
CD8405	CALL	nn
3F	CCF	
BE	CP	(HL)
DDBE05	CP'	(IX+d)
FDBE05	CP	(IY+d)
BF	CP	A
B8	CP	B
B9	CP	C
BA	CP	D
BB	CP	E
BC	CP	H
BD	CP	L
FE20	CP	n
EDA9	CPD	
EDB9	CPDR	

EDB1	CPIR	
EDA1	CPI	
2F	CPL	
27	DAA	
35	DEC	(HL)
DD3505	DEC	(IX+d)
FD3505	DEC	(IY+d)
3D	DEC	A
05	DEC	B
0B	DEC	BC
0D	DEC	C
15	DEC	D
1B	DEC	DE
1D	DEC	E
25	DEC	H
2B	DEC	HL
DD2B	DEC	IX
FD2B	DEC	IY
2D	DEC	L
3B	DEC	SP
F3	DI	
102E	DJNZ	e
FB	EI	
E3	EX	(SP),HL
DDE3	EX	(SP),IX
FDE3	EX	(SP),IY
08	EX	AF,AF'
EB	EX	DE,HL
D9	EXX	
76	HALT	
ED46	IM	0
ED56	IM	1
ED5E	IM	2
ED78	IN	A,(C)
ED40	IN	B,(C)
ED48	IN	C,(C)
ED50	IN	D,(C)
ED58	IN	E,(C)
ED60	IN	H,(C)
ED68	IN	L,(C)
34	INC	(HL)
DD3405	INC	(IX+d)
FD3405	INC	(IY+d)
3C	INC	A
04	INC	B
03	INC	BC
0C	INC	C
14	INC	D
13	INC	DE
1C	INC	E
24	INC	H
23	INC	HL
DD23	INC	IX
FD23	INC	IY
2C	INC	L
33	INC	SP
DB20	IN	A,(n)

Z80 op codes (Courtesy of Zilog)   05=d, 8405=nn, 20=n, 2E=e

EDAA	IND	
EDBA	INDR	
EDA2	INI	
EDB2	INIR	
C38405	JP	nn
E9	JP	(HL)
DDE9	JP	(IX)
FDE9	JP	(IY)
DA8405	JP	C,nn
FA8405	JP	M,nn
D28405	JP	NC,nn
C28405	JP	NZ,nn
F28405	JP	P,nn
EA8405	JP	PE,nn
E28405	JP	PO,nn
CA8405	JP	Z,nn
382E	JR	C,e
302E	JR	NC,e
202E	JR	NZ,e
282E	JR	Z,e
182E	JR	e
02	LD	(BC),A
12	LD	(DE),A
77	LD	(HL),A
70	LD	(HL),B
71	LD	(HL),C
72	LD	(HL),D
73	LD	(HL),E
74	LD	(HL),H
75	LD	(HL),L
3620	LD	(HL),n
DD7705	LD	(IX+d),A
DD7005	LD	(IX+d),B
DD7105	LD	(IX+d),C
DD7205	LD	(IX+d),D
DD7305	LD	(IX+d),E
DD7405	LD	(IX+d),H
DD7505	LD	(IX+d),L
DD360520	LD	(IX+d),n
FD7705	LD	(IY+d),A
FD7005	LD	(IY+d),B
FD7105	LD	(IY+d),C
FD7205	LD	(IY+d),D
FD7305	LD	(IY+d),E
FD7405	LD	(IY+d),H
FD7505	LD	(IY+d),L
FD360520	LD	(IY+d),n
328405	LD	(nn),A
ED438405	LD	(nn),BC
ED538405	LD	(nn),DE
228405	LD	(nn),HL
DD228405	LD	(nn),IX
FD228405	LD	(nn),IY
ED738405	LD	(nn),SP
0A	LD	A,(BC)
1A	LD	A,(DE)
7E	LD	A,(HL)
DD7E05	LD	A,(IX+d)
FD7E05	LD	A,(IY+d)
3A8405	LD	A,(nn)
7F	LD	A,A
78	LD	A,B
79	LD	A,C
7A	LD	A,D
7B	LD	A,E
7C	LD	A,H
ED57	LD	A,I
7D	LD	A,L
3E20	LD	A,n
ED5F	LD	A,R
46	LD	B,(HL)
DD4605	LD	B,(IX+d)
FD4605	LD	B,(IY+d)
47	LD	B,A
40	LD	B,B
41	LD	B,C
42	LD	B,D
43	LD	B,E
44	LD	B,H
45	LD	B,L
0620	LD	B,n
ED4B8405	LD	BC,(nn)
018405	LD	BC,nn
4E	LD	C,(HL)
DD4E05	LD	C,(IX+d)
FD4E05	LD	C,(IY+d)
4F	LD	C,A
48	LD	C,B
49	LD	C,C
4A	LD	C,D
48	LD	C,E
4C	LD	C,H
4D	LD	C,L
0E20	LD	C,n
56	LD	D,(HL)
DD5605	LD	D,(IX+d)
FD5605	LD	D,(IY+d)
57	LD	D,A
50	LD	D,B
51	LD	D,C
52	LD	D,D
53	LD	D,E
54	LD	D,H
55	LD	D,L
1620	LD	D,n
ED5B8405	LD	DE,(nn)
118405	LD	DE,nn
5E	LD	E,(HL)
DD5E05	LD	E,(IX+d)
FD5E05	LD	E,(IY+d)
5F	LD	E,A
58	LD	E,B
59	LD	E,C
5A	LD	E,D
5B	LD	E,E
5C	LD	E;H
5D	LD	E,L
1E20	LD	E,n
66	LD	H,(HL)
DD6605	LD	H,(IX+d)
FD6605	LD	H,(IY+d)
67	LD	H,A
60	LD	H,B
61	LD	H,C
62	LD	H,D
63	LD	H,E
64	LD	H,H
65	LD	H,L
2620	LD	H,n
2A8405	LD	HL,(nn)
218405	LD	HL,nn
ED47	LD	I,A
DD2A8405	LD	IX,(nn)
DD218405	LD	IX,nn
FD2A8405	LD	IY,(nn)
FD218405	LD	IY,nn
6E	LD	L,(HL)
DD6E05	LD	L,(IX+d)
FD6E05	LD	L,(IY+d)
6F	LD	L,A
68	LD	L,B
69	LD	L,C
6A	LD	L,D
6B	LD	L,E
6C	LD	L,H
6D	LD	L,L
2E20	LD	L,n
ED4F	LD	R,A
ED788405	LD	SP,(nn)
F9	LD	SP,HL
DDF9	LD	SP,IX
FDF9	LD	SP,IY
318405	LD	SP,nn
EDA8	LDD	
EDB8	LDDR	
EDA0	LDI	
EDB0	LDIR	
ED44	NEG	
00	NOP	
B6	OR	(HL)
DDB605	OR	(IX+d)
FDB605	OR	(IY+d)
87	OR	A
B0	OR	B
B1	OR	C
B2	OR	D
B3	OR	E
B4	OR	H
B5	OR	L
F620	OR	n
ED8B	OTDR	
EDB3	OTIR	
E079	OUT	(C),A
ED41	OUT	(C),B
ED49	OUT	(C),C
ED51	OUT	(C),D
ED59	OUT	(C),E
ED61	OUT	(C),H
ED69	OUT	(C),L
D320	OUT	(n),A
EDAB	OUTD	
EDA3	OUTI	
F1	POP	AF
C1	POP	BC
D1	POP	DE
E1	POP	HL
DDE1	POP	IX
FDE1	POP	IY
F5	PUSH	AF
C5	PUSH	BC
D5	PUSH	DE
E5	PUSH	HL
DDE5	PUSH	IX
FDE5	PUSH	IY
CB86	RES	0,(HL)
DDCB0586	RES	0,(IX+d)
FDCB0586	RES	0,(IY+d)
CB87	RES	0,A
CB80	RES	0,B
CB81	RES	0,C
CB82	RES	0,D
CB83	RES	0,E
CB84	RES	0,H
CB85	RES	0,L
CB8E	RES	1,(HL)
DDCB058E	RES	1,(IX+d)
FDCB058E	RES	1,(IY+d)
CB8F	RES	1,A
CB88	RES	1,B
CB89	RES	1,C
C88A	RES	1,D
CB8B	RES	1,E
C88C	RES	1,H
CB8D	RES	1,L
CB96	RES	2,(HL)
DDCB0596	RES	2,(IX+d)
FDCB0596	RES	2,(IY+d)
CB97	RES	2,A
CB90	RES	2,B
CB91	RES	2,C
CB92	RES	2,D
CB93	RES	2,E
C894	RES	2,H
CB95	RES	2,L
CB9E	RES	3,(HL)
DDCB059E	RES	3,(IX+d)
FDCB059E	RES	3,(IY+d)

Z80 op codes (Courtesy of Zilog)

05=d, 8405=nn, 20=n, 2E=e

CB9F	RES	3,A
CB98	RES	3,B
CB99	RES	3,C
CB9A	RES	3,D
CB9B	RES	3,E
CB9C	RES	3,H
CB9D	RES	3,L
CBA6	RES	4,(HL)
DDCB05A6	RES	4,(IX+d)
FDCB05A6	RES	4,(IY+d)
CBA7	RES	4,A
CBA0	RES	4,B
CBA1	RES	4,C
CBA2	RES	4,D
CBA3	RES	4,E
CBA4	RES	4,H
CBA5	RES	4,L
CBAE	RES	5,(HL)
DDCB05AE	RES	5,(IX+d)
FDCB05AE	RES	5,(IY+d)
CBAF	RES	5,A
CBA8	RES	5,B
CBA9	RES	5,C
CBAA	RES	5,D
CBAB	RES	5,E
CBAC	RES	5,H
CBAD	RES	5,L
CBB6	RES	6,(HL)
DDCB05B6	RES	6,(IX+d)
FDCB05B6	RES	6,(IY+d)
CBB7	RES	6,A
CBB0	RES	6,B
CBB1	RES	6,C
CBB2	RES	6,D
CBB3	RES	6,E
CBB4	RES	6,H
CBB5	RES	6,L
CBBE	RES	7,(HL)
DDCB05BE	RES	7,(IX+d)
FDCB05BE	RES	7,(IY+d)
CBBF	RES	7,A
C8B8	RES	7,B
CBB9	RES	7,C
CBBA	RES	7,D
CBBB	RES	7,E
CBBC	RES	7,H
CBBD	RES	7,L
C9	RET	
D8	RET	C
F8	RET	M
D0	RET	NC
C0	RET	NZ
F0	RET	P
E8	RET	PE
E0	RET	PO
C8	RET	Z
ED4D	RETI	
ED45	RETN	
CB16	RL	(HL)
DDCB0516	RL	(IX+d)
FDCB0516	RL	(IY+d)
CB17	RL	A
CB10	RL	B
CB11	RL	C
CB12	RL	D
CB13	RL	E
CB14	RL	H
CB15	RL	L
17	RLA	
CB06	RLC	(HL)
DDCB0506	RLC	(IX+d)
FDCB0506	RLC	(IY+d)
CB07	RLC	A
CB00	RLC	B
CB01	RLC	C
CB02	RLC	D
CB03	RLC	E
CB04	RLC	H
CB05	RLC	L
07	RLCA	
ED6F	RLD	
CB1E	RR	(HL)
DDCB051E	RR	(IX+d)
FDCB051E	RR	(IY+d)
CB1F	RR	A
CB18	RR	B
CB19	RR	C
CB1A	RR	D
CB1B	RR	E
CB1C	RR	H
CB1D	RR	L
1F	RRA	
CB0E	RRC	(HL)
DDCB050E	RRC	(IX+d)
FDCB050E	RRC	(IY+d)
C80F	RRC	A
CB08	RRC	B
CB09	RRC	C
CB0A	RRC	D
CB0B	RRC	E
CB0C	RRC	H
C80D	RRC	L
0F	RRCA	
ED67	RRD	
C7	RST	00H
CF	RST	08H
D7	RST	10H
DF	RST	18H
E7	RST	20H
EF	RST	28H
F7	RST	30H
FF	RST	38H
DE20	SBC	A,n
9E	SBC	A,(HL)
DD9E05	SBC	A,(IX+d)
FD9E05	SBC	A,(IY+d)
9F	SBC	A,A
98	SBC	A,B
99	SBC	A,C
9A	SBC	A,D
9B	SBC	A,E
9C	SBC	A,H
9D	SBC	A,L
ED42	SBC	HL,BC
ED52	SBC	HL,DE
ED62	SBC	HL,HL
ED72	SBC	HL,SP
37	SCF	
CBC6	SET	0,(HL)
DDCB05C6	SET	0,(IX+d)
FDCB05C6	SET	0,(IY+d)
CBC7	SET	0,A
CBC0	SET	0,B
CBC1	SET	0,C
CBC2	SET	0,D
CBC3	SET	0,E
CBC4	SET	0,H
CBC5	SET	0,L
CBCE	SET	1,(HL)
DDCB05CE	SET	1,(IX+d)
FDCB05CE	SET	1,(IY+d)
CBCF	SET	1,A
CBC8	SET	1,B
CBC9	SET	1,C
CBCA	SET	1,D
CBCB	SET	1,E
CBCC	SET	1,H
CBCD	SET	1,L
CBD6	SET	2,(HL)
DDCB05D6	SET	2,(IX+d)
FDCB05D6	SET	2,(IY+d)
CBD7	SET	2,A
CBD0	SET	2,B
CBD1	SET	2,C
CBD2	SET	2,D
CBD3	SET	2,E
CBD4	SET	2,H
CBD5	SET	2,L
CBD8	SET	3,B
CBDE	SET	3,(HL)
DDCB05DE	SET	3,(IX+d)
FDCB05DE	SET	3,(IY+d)
CBDF	SET	3,A
CBD9	SET	3,C
CBDA	SET	3,D
C8DB	SET	3,E
CBDC	SET	3,H
CBDD	SET	3,L
CBE6	SET	4,(HL)
DDCB05E6	SET	4,(IX+d)
FDCB05E6	SET	4,(IY+d)
CBE7	SET	4,A
CBE0	SET	4,B
CBE1	SET	4,C
CBE2	SET	4,D
CBE3	SET	4,E
CBE4	SET	4,H
CBE5	SET	4,L
CBEE	SET	5,(HL)
DDCB05EF	SET	5,(IX+d)
FDCB05EE	SET	5,(IY+d)
CBEF	SET	5,A
CBE8	SET	5,B
CBE9	SET	5,C
CBEA	SET	5,D
CBEB	SET	5,E
CBEC	SET	5,H
CBED	SET	5,L
C8F6	SET	6,(HL)
DDCB05F6	SET	6,(IX+d)
FDCB05F6	SET	6,(IY+d)
CBF7	SET	6,A
CBF0	SET	6,B
CBF1	SET	6,C
CBF2	SET	6,D
CBF3	SET	6,E
CBF4	SET	6,H
CBF5	SET	6,L
CBFE	SET	7,(HL)
DDCB05FE	SET	7,(IX+d)
FDCB05FE	SET	7,(IY+d)
CBFF	SET	7,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CB26	SLA	(HL)
DDCB0526	SLA	(IX+d)
FDCB0526	SLA	(IY+d)
CB27	SLA	A
CB20	SLA	B
CB21	SLA	C
CB22	SLA	D
CB23	SLA	E
CB24	SLA	H
CB25	SLA	L
CB2E	SRA	(HL)
DDCB052E	SRA	(IX+d)
FDCB052E	SRA	(IY+d)
CB2F	SRA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L
CB3E	SRL	(HL)
DDCB053E	SRL	(IX+d)
FDCB053E	SRL	(IY+d)
CB3F	SRL	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L
96	SUB	(HL)
DD9605	SUB	(IX+d)
FD9605	SUB	(IY+d)
97	SUB	A
90	SUB	B
91	SUB	C
92	SUB	D
93	SUB	E
94	SUB	H
95	SUB	L
D620	SUB	n
AE	XOR	(HL)
DDAE05	XOR	(IX+d)
FDAE05	XOR	(IY+d)
AF	XOR	A
A8	XOR	B
A9	XOR	C
AA	XOR	D
AB	XOR	E
AC	XOR	H
AD	XOR	L
EE20	XOR	n

CHAPTER 4. Memory Map (all numbers hexadecimal).

0000-00FF Zero page. interrupt routines. All C9 (return) except at 66-AB.

0100 Start of BASIC

0101-0104 Pointers for version of Basic. See Coleco manual p.C23 My version has A3 3E C3 4F here.

010B-03A8 Basic word table. Format: token (1 byte), address in address table (2 bytes), number of letters in word (1 byte), word.

03A9-041F Routine address table. Format: number of addresses (1 byte), address(es) (2 bytes each).

0420-047F Hi Cathy and copyright statement.

0480-05B7 Error messages. Format: number of letters (1 byte), message in ASCII.

05B8-3ED8 Basic routines. Identify from word and address tables.

3ED9 Himem pointer.

3EDE Lomem pointer.

3EE3 Pointer to start of numeric variables.

3EED Pointer to end of numeric variables.

3EEF Pointer to start of string space.

3EF3 Pointer to end of string space.

3EFE Line number for ONERR GOTO.

3F01 Speed (FF).

3F02 USR address. CALL is better that USR. Forget it.

3F04 @ address.

3F22-3F26 FP accumulator (see chap. 2).

3F2B-3F2F FP operand.

3F32 number of digits in FP result.

3F40-3FA3 Scratch pad?

3FA4-4045 Basic words, math. Format: number of letters, word, 88 or A8, address.

4EAA-4F4E Tape word table. Format: number of letters, word, address table pointer (1 byte), which gives the offset of the address from the beginning of address table.

4F4F-4FA5 Tape address table. Format: 2 byte address of routine. Pointed to by offset in word table.

4FA6-5E3F Tape routines. see tape word and address tables.

5E40-5EE8 Tape error messages. Format: number of letters, message.

6B00 Approximate location of string variable table. Format: 03 21 address (2 bytes), name (2 bytes).

6B00 After string table is the numeric variable table. Format: 03 01, address (2 bytes), name (2 bytes).

6B00 After numeric table is Basic math word table.

6C00 String space. Format: address in table, number of letters (bytes), string.

CE00-CF00 Numeric variables (see chap. 2). Numbers are preceded by letters of the name after first two.

CF00- Tokenized BASIC program (see chap. 6).

D200- Stack

D400-D700 Buffer from tape: catalog. Format: name, type, 17 bytes (sectors on tape?).

D800- Buffer from tape: last program loaded.

E000	Start of operating system (OS).
E010-	General block output.
E02A-	General block input.
E0CF-	Printer.
E0D5-	Output to VRAM.
FC18- FC2C	Pointers, VRAM table numbers, out addresses.
FC30- FD5E	Start of OS jump table.
FD75	Keyboard input byte.

IN/OUT space.

60-7F	Bus for printer, tape.
A0-BF	Video display processor.
E0-FF	Sound generator.

CHAPTER 5. A DISASSEMBLER

The disassembler listing which follows will translate machine code into assembly language. It is essentially several tables of pointers by which the machine language op code points to the assembly language mnemonic and register or address information. These tables are entered as data statements of letters and symbols which are converted to numbers by the ASCII code because it is shorter and requires less typing. The information is then put into string arrays which are: nm$= mnemonics, t$= names of registers etc.; a$(x), b$(x), c$(x) which have pointers to nm$,t$,t$, respectively;d$(x),e$(x) and f$(x) like a$,b$,c$ when the op code begins with ED; and g$, h$, i$, for op codes which begin with CB. Line 23 prints the address in hexadecimal. Line 25 prints the op code. Lines 30-60 check for special codes and gosub appropriately. In lines 100 and 110 n is the number of bytes expected following the op code. The variables pa,pb,and pc are the pointers as numbers extracted from the string arrays. Lines 3000 to 4000 fill the string arrays when the program is first run. Lines 5000 to 5095 are a decimal to hexadecimal conversion subroutine.

When you run the program it asks for a starting address, which should be in decimal. It then prints out the disassembled listing until you stop it by typing control s or c. If you have fan-fold paper you can leave it going for hours (plan on leaving the house if you have sensitive ears). To avoid disassembling ASCII, tables and garbage etc., consult the memory map and print out relevant areas of RAM with printmem first because it is much faster. Typical output lines are as follows:

```
2010  07DA  79       LD    A,C        y
2011  07DB  08       EX    AF,AF`
2012  07DC  48       LD    C,B        H
2013  07DD  43       LD    B,E        C
2014  07DE  5A       LD    E,D        Z
2015  07DF  1600     LD    D,n
2017  07E1  C9       RET
 address  op code  mnemonic          ASCII
```

The address is first printed in decimal and then in hexadecimal. The op code is then printed in hexadecimal, followed by the mnemonic. On the for right the ASCII symbol of the op code is printed to help identify words in ASCII which were not intended to be op codes.

If you type the program in and it runs alright you may still have made an error by adding an extra data element. To check for that type "? i$(255)" in the immediate mode after running the program. The result should be "@". Checking for substitution errors could be done by driving the program with a for-next loop to generate all op codes and comparing them with the listing at the end of chapter 3.

There may be more efficient ways to write a disassembler for the Z80, but this one works and was enough trouble to write that I am not going to change it. It has some illogical aspects, such as the listing of the mnemonic CPIR twice, that are slightly embarrassing, but still not worth changing. On the other hand it can easily be modified to print addresses instead of "nn" or to input hex numbers, etc. which you are welcome to do. It could even be turned into an assembler by creating string arrays of complete mnemonic statements (complete lines) to be searched through for a match to lines typed in. It would be slow but useful . The major work of designing and typing in the data for the op code tables would be done already for the disassembler.

Printmem is a short program that prints out RAM in a convenient format to interpret before disassembling. The ASCII equivalents of the numbers are printed on the left with = signs for non-ASCII numbers. Lines of 16 hexadecimal numbers are then printed in pages of 256. The format is particularly useful for interpreting tables and variable or string areas. A sample printout of page 4 is shown following the program.

Viewer is a very short program which displays pages of RAM on the screen as ASCII and graphics characters. It is a good one to start with.

Viewchr is a minor modification of viewer, which allows you to see the graphics characters on the screen. The ASCII values can be seen from the position on the screen.

```
]

  2 REM      Z80 disassembler by P. Hinkle,March 1984
  5 GOTO 1000
 10 INPUT "start addr"; ad
 11 PR #1
 20 PRINT: op = PEEK(ad)
 21 n = 0: nl = 0: dc = 0
 22 PRINT ad; TAB(7);
 23 GOSUB 5000
 25 GOSUB 120
 30 IF op = 203 THEN  GOSUB 200: GOTO 150
 40 IF op = 221 THEN  GOSUB 400: GOTO 150
 50 IF op = 237 THEN  GOSUB 600: GOTO 150
 60 IF op = 253 THEN  GOSUB 800: GOTO 150
 66 GOSUB 70
 67 GOTO 150
 70 pa = ASC(a$(op))
 80 pb = ASC(b$(op))
 90 pc = ASC(c$(op))
100 IF pb = 78 OR pb = 94 OR pc = 78 OR pc = 94 THEN  n = 2: nl = 2
110 IF pb = 86 OR pb = 71 OR pb = 89 OR pc = 86 OR pc = 71 OR pc = 89 THEN  n
= 1: nl = 1
115 RETURN
118 ad = ad+1: op = PEEK(ad)
120 PRINT MID$(x$, INT(op/16)+1, 1);
130 PRINT MID$(x$, (op/16-INT(op/16))*16+1, 1);
140 RETURN
150 IF n > 0 THEN  ad = ad+1: n = n-1: op = PEEK(ad): GOSUB 120
160 IF n > 0 THEN  ad = ad+1: op = PEEK(ad): GOSUB 120
170 PRINT TAB(23)
180 PRINT nm$(pa-49); TAB(29); t$(pb-64);
181 IF pc = 117 THEN  GOTO 185
183 PRINT ","; t$(pc-64);
185 IF nl = 2 THEN  PRINT SPC(4): GOSUB 120: op = PEEK(ad-1): GOSUB 120
187 pp = POS(0)
188 IF pp < 20 THEN  pp = pp+31
189 PRINT SPC(60-pp);
190 IF nl = 2 THEN  GOSUB 5100
192 IF nl = 1 THEN  GOSUB 5100
194 GOSUB 5100
199 ad = ad+1: GOTO 20
200 REM                   CB routine
210 GOSUB 118
230 pa = ASC(g$(op))
240 pb = ASC(h$(op))
250 pc = ASC(i$(op))
260 GOSUB 100: RETURN
400 REM              DD routine
420 GOSUB 118
430 IF op = 203 THEN  GOSUB 118: GOSUB 200: dc = 1: GOTO 450
440 GOSUB 70
450 IF pb = 95 THEN  pb = 96: IF dc = 0 THEN  GOSUB 118
452 IF pb = 72 THEN  pb = 76
454 IF pc = 95 THEN  pc = 96: IF dc = 0 THEN  GOSUB 118
456 IF pc = 72 THEN  pc = 76
460 RETURN
600 REM                   ED routine
610 GOSUB 118
630 pa = ASC(d$(op-64))
640 pb = ASC(e$(op-64))
650 pc = ASC(f$(op-64))
660 GOSUB 100: RETURN
800 REM                   FD routine
```

```
] 800 REM                    FD routine
  810 GOSUB 118
  820 IF op = 203 THEN  GOSUB 118: GOSUB 200: dc = 1: GOTO 850
  830 GOSUB 70
  850 IF pb = 95 THEN  pb = 97: IF dc = 0 THEN  GOSUB 118
  852 IF pb = 72 THEN  pb = 77
  854 IF pc = 95 THEN  pc = 97: IF dc = 0 THEN  GOSUB 118
  856 IF pc = 72 THEN  pc = 77
  860 RETURN
 1000 x$ = "0123456789ABCDEF"
 2000 DATA           A,B,C,D,E,H,L,n,HL,BC,DE,SP,IX,IY,nn,M,NC,NZ,P,PE,PO,Z,e,(S
P),(C),(n),(IX)
 2001 DATA     (IY),(BC),(DE),(nn),(HL),(IX+d),(IY+d),0,1,2,3,4,5,6,7,I,R,00H,08H
,10H,18H,20H,28H,30H,38H,?,  ,AF,AF`,(A),(HL)
 2002 DATA     ADC,ADD,AND,BIT,CALL,CCF,CP,CPD,CPDR,CPIR,CPI,CPL,DAA,DEC,DI,DJNZ,
EI,EX,EXX,HALT
 2003 DATA     IM,IN,INC,IND,INDR,INI,INIR,JP,JR,LD,LDD,LDDR,LDI,LDIR,NEG,NOP,OR,
OTDR,OTIR,OUT,OUTD
 2004 DATA     OUTI,POP,PUSH,RES,RET,RETI,RETN,RL,RLA,RLC,RLCA,RLD,RR,RRA,RRC,RRC
A,RRD,RST,SBC,SCF
 2005 DATA    SET,SLA,SRA,SRL,SUB,XOR,RETI,?,CPIR
 2010 DATA    T,N,N,G,G,>,N,d,B,2,N,>,G,>,N,i,@,N,N,G,G,>,N,b,M,2,N,>,G,>,N,g,M,N
,N,G,G,>,N
 2011 DATA     =,M,2,N,>,G,>,N,<,M,N,N,G,G,>,N,m,M,2,N,>,G,>,N,6
 2012 DATA     N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N
 2013 DATA     N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,N,D,N,N,N,N,N,N,N,N,N
 2014 DATA     2,2,2,2,2,2,2,2,l,l,l,l,l,l,l,l,r,r,r,r,r,r,r,r,l,l,l,l,l,l,l,l
 2015 DATA     3,3,3,3,3,3,3,3,s,s,s,s,s,s,s,s,U,U,U,U,U,U,U,U,7,7,7,7,7,7,7,7
 2016 DATA     ^,[,L,L,5,\,2,k,^,^,L,t,5,5,l,k,^,[,L,X,5,\,r,k,^,C,L,F,5,t,l,k
 2017 DATA     ^,[,L,B,5,\,3,k,^,L,L,B,5,t,s,k,^,[,L,?,5,\,U,k,k,N,L,A,5,t,7,k
 2020 DATA     u,I,\,I,A,A,A,u,v,H,@,I,B,B,B,u,V,J,],J,C,C,C,u,V,H,@,J,D,D,D,u,Q,
H
 2021 DATA     ^,H,E,E,E,u,U,H,H,H,F,F,F,u,P,K,^,K,_,_,_,u,B,H,@,K,@,@,@,u
 2022 DATA     A,A,A,A,A,A,A,A,B,B,B,B,B,B,B,B,C,C,C,C,C,C,C,C,D,D,D,D,D,D,D,D
 2023 DATA     E,E,E,E,E,E,E,E,F,F,F,F,F,F,F,F,_,_,_,_,_,_,u,_,@,@,@,@,@,@,@,@
 2024 DATA     @,@,@,@,@,@,@,@,@,@,@,@,@,@,@,@,A,B,C,D,E,F,_,@,@,@,@,@,@,@,@,@
 2025 DATA     A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
 2027 DATA     Q,I,Q,N,Q,I,@,l,U,u,U,t,U,N,@,m,P,J,P,Y,P,J,G,n,B,u,B,@,B,t,@
 2028 DATA     o,T,H,T,W,T,H,G,p,S,_,S,J,S,t,G,q,R,v,R,u,R,v,G,r,O,K,O,u,O,u,G,s

 2030 DATA     u,N,@,u,u,u,G,u,w,I,\,u,u,u,G,u,u,N,@,u,u,u,G,u,u,J,],u,u,u,G,u,V,
N,H,u,u,u,G,u,V,H
 2031 DATA     ^,u,u,u,G,u,V,N,@,u,u,u,G,u,V,K,^,u,u,u,G,u
 2040 DATA     A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
 2041 DATA     A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,u,@,A,B,C,D,E,F,_,@
 2050 DATA     A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,u,u,u,u,u,u,u,u,A,B,C,D,E,F,_,@
 2051 DATA     u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
 2060 DATA     u,u,N,u,N,u,G,u,u,u,N,t,N,u,G,u,u,u,N,@,N,u,u,u,u,u,N,G,N,t,G,u,u,
u,N,H,N,u,u,u,u,u
 2061 DATA     N,H,N,t,u,u,u,u,N,u,N,u,u,u,u,H,N,u,N,u,u,u
 2070 DATA      F,X,l,N,S,`,E,N,F,X,l,N,u,_,u,N,F,X,l,N,u,u,E,N,F,X,l,N,u,u,E,N,F
,X,l,u,u,u,u,j,F,X,l,u,u,u,u
 2071 DATA      e,u,u,l,N,u,u,u,u,F,X,l,N,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
 2072 DATA    .V,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,Q,;,J,Z,u,u,u,u
 2073 DATA     O,8,H,Y,u,u,u,u,R,v,K,W,u,u,u,u,P,9,I
 2079 DATA      A,X,H,^,u,u,b,j,B,X,H,I,u,u,u,k,C,X,H
 2080 DATA      ^,u,u,c,@,D,X,H,J,u,u,d,@,E,X,H,u,u,u,u,u,F,X,H,u,u,u,u,u,u,u,H,^
,u,u,u,u,@
 2081 DATA      X,H,K,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
 2082 DATA     u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
 2083 DATA      X,A,I,I,u,u,@,u,X,B,I,^,u,u,u,@,X,C,J,J,u,u,u,j,X,D,J,^,u,u,u,k,X
,E,H,u,u,u,u,u,X,F,H,u,u,u,u,u,u,u,K,K
 2084 DATA      u,u,u,u,X,@,K,^,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
,u
 2085 DATA     u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,
u,u,u,u
 2086 DATA     c,c,c,c,c,c,c,c,h,h,h,h,h,h,h,h,a,a,a,a,a,a,a,a,f,f,f,f,f,f,f,f
 2087 DATA     o,o,o,o,o,o,o,o,p,p,p,p,p,p,p,p,u,u,u,u,u,u,u,u,q,q,q,q,q,q,q,q
 2088 DATA     4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
 2089 DATA     4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4,4
 2090 DATA     ],],],],],],],],],],],],],],],],],],],],],],],],],],],],],],],]
 2091 DATA     ],],],],],],],],],],],],],],],],],],],],],],],],],],],],],],],]
 2092 DATA     n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n
 2093 DATA     n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n,n
```

```
2100 DATA   A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2101 DATA   A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,t,t,t,t,t,t,t,t,A,B,C,D,E,F,_,@
2110 DATA   b,b,b,b,b,b,b,b,c,c,c,c,c,c,c,c,d,d,d,d,d,d,d,d,e,e,e,e,e,e,e,e
2111 DATA   f,f,f,f,f,f,f,f,g,g,g,g,g,g,g,g,h,h,h,h,h,h,h,h,i,i,i,i,i,i,i,i
2112 DATA   b,b,b,b,b,b,b,b,c,c,c,c,c,c,c,c,d,d,d,d,d,d,d,d,e,e,e,e,e,e,e,e
2113 DATA   f,f,f,f,f,f,f,f,g,g,g,g,g,g,g,g,h,h,h,h,h,h,h,h,i,i,i,i,i,i,i,i
2114 DATA   b,b,b,b,b,b,b,b,c,c,c,c,c,c,c,c,d,d,d,d,d,d,d,d,e,e,e,e,e,e,e,e
2115 DATA  f,f,f,f,f,f,f,f,g,g,g,g,g,g,g,g,h,h,h,h,h,h,h,h,i,i,i,i,i,i,i,i
2120 DATA  u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
2121 DATA  u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u,u
2122 DATA  A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2123 DATA   A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2124 DATA  A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2125 DATA  A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2126 DATA  A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
2127 DATA  A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@,A,B,C,D,E,F,_,@
3000 DIM nm$(69)
3001 DIM t$(57)
3002 DIM a$(255)
3003 DIM b$(255): DIM c$(255)
3004 DIM d$(122): DIM e$(122): DIM f$(122)
3005 DIM g$(255): DIM h$(255): DIM i$(255)
3010 FOR x = 0 TO 57: READ t$(x): NEXT
3020 FOR x = 0 TO 69: READ nm$(x): NEXT
3021 FOR x = 0 TO 255: READ a$(x): NEXT
3022 FOR x = 0 TO 255: READ b$(x): NEXT
3023 FOR x = 0 TO 255: READ c$(x): NEXT
3030 FOR x = 0 TO 122: READ d$(x): NEXT
3031 FOR x = 0 TO 122: READ e$(x): NEXT
3032 FOR x = 0 TO 122: READ f$(x): NEXT
3040 FOR x = 0 TO 255: READ g$(x): NEXT
3041 FOR x = 0 TO 255: READ h$(x): NEXT
3042 FOR x = 0 TO 255: READ i$(x): NEXT
4000 GOTO 10
5000 a = INT(ad/4096)
5010 PRINT MID$(x$, a+1, 1);
5020 b = ad-a*4096
5030 c = INT(b/256)
5040 PRINT MID$(x$, c+1, 1);
5050 d = b-c*256
5060 e = INT(d/16)
5070 PRINT MID$(x$, e+1, 1);
5080 f = d-e*16
5090 PRINT MID$(x$, INT(f)+1, 1);
5092 PRINT "  ";
5095 RETURN
5100 jj = PEEK(ad-nl)
5110 IF jj > 33 AND jj < 123 THEN  PRINT CHR$(jj);
5120 nl = nl-1: RETURN

]
```

A sample disassembly is shown on the next page.

```
57344 E000  C5        PUSH  BC
57345 E001  EB        EX    DE,HL
57346 E002  CDE9E1    CALL  nn      E1E9
57349 E005  69        LD    L,C                           i
57350 E006  C1        POP   BC
57351 E007  EB        EX    DE,HL
57352 E008  79        LD    A,C                           y
57353 E009  4B        LD    C,E                           K
57354 E00A  50        LD    D,B                           P
57355 E00B  14        INC   D
57356 E00C  47        LD    B,A                           G
57357 E00D  B7        OR    A
57358 E00E  2806      JR    Z,e                           (
57360 E010  EDA3      OUTI
57362 E012  00        NOP
57363 E013  00        NOP
57364 E014  20FA      JR    NZ,e
57366 E016  15        DEC   D
57367 E017  20F7      JR    NZ,e
57369 E019  C9        RET
57370 E01A  C5        PUSH  BC
57371 E01B  EB        EX    DE,HL
57372 E01C  CDE7E1    CALL  nn      E1E7
57375 E01F  69        LD    L,C                           i
57376 E020  C1        POP   BC
57377 E021  EB        EX    DE,HL
57378 E022  79        LD    A,C                           y
57379 E023  4B        LD    C,E                           K
57380 E024  50        LD    D,B                           P
57381 E025  14        INC   D
57382 E026  47        LD    B,A                           G
57383 E027  B7        OR    A
57384 E028  2806      JR    Z,e                           (
57386 E02A  EDA2      INI
57388 E02C  00        NOP
57389 E02D  00        NOP
57390 E02E  20FA      JR    NZ,e
57392 E030  15        DEC   D
57393 E031  20F7      JR    NZ,e
57395 E033  C9        RET
57396 E034  59        LD    E,C                           Y
57397 E035  3A29FC    LD    A,(nn)      FC29              :)
57400 E038  4F        LD    C,A                           O
57401 E039  ED59      OUT   (C),E                         Y
57403 E03B  78        LD    A,B                           x
57404 E03C  F680      OR    n
57406 E03E  ED79      OUT   (C),A                         y
57408 E040  78        LD    A,B                           x
57409 E041  B7        OR    A
57410 E042  7B        LD    A,E
57411 E043  2004      JR    NZ,e
57413 E045  3261FD    LD    (nn),A      FD61              2a
57416 E048  C9        RET
57417 E049  05        DEC   B
57418 E04A  C0        RET   NZ
57419 E04B  3262FD    LD    (nn),A      FD62              2b
57422 E04E  C9        RET
57423 E04F  3A29FC    LD    A,(nn)      FC29              :)
```

```
]

  1 REM   PRINTMEM  by P. Hinkle
  2 PR #1
  3 h$ = "0123456789ABCDEF"
  5 INPUT "page"; p
  6 PRINT p
 10 FOR j = 0 TO 240 STEP 16
 15 PRINT "          ";
 20 FOR i = 0 TO 15
 30 x = p*256+i+j
 40 t = PEEK(x)
 41 IF t < 32 OR t > 126 THEN  t = 61
 50 PRINT CHR$(t);
 60 NEXT i
 65 GOSUB 200
 70 PRINT
 80 NEXT j
 85 PRINT: PRINT: PRINT: PRINT: PRINT
 90 p = p+1: GOTO 6
200 PRINT TAB(30);
210 FOR i = 0 TO 15
220 a = PEEK(p*256+i+j)
230 b = a/16
240 c = INT(b)
250 GOSUB 300
260 c = (b-INT(b))*16
270 GOSUB 300
280 PRINT " ";
290 NEXT i
295 RETURN
300 c = c+1
310 d$ = MID$(h$, c, 1)
~~315 ww = FRE(9)~~
320 PRINT d$;
330 RETURN

==:=:==:C>=:i>=:          02 1B 3A 80 3A 05 1B 3A 43 3E 1B 3A 69 3E 1B 3A
=6>=:=N>=:='>=:=          02 36 3E 1B 3A 02 4E 3E 1B 3A 02 27 3E 1B 3A 04
Hi Cathy=FATAL S          48 69 20 43 61 74 68 79 12 46 41 54 41 4C 20 53
YSTEM ERROR==             59 53 54 45 4D 20 45 52 52 4F 52 1C 0C 20 20 20
  Coleco SmartBA          20 20 43 6F 6C 65 63 6F 20 53 6D 61 72 74 42 41
SIC V1.0 (c) 198          53 49 43 20 56 31 2E 30 20 28 63 29 20 31 39 38
3, Lazer MicroSy          33 2C 20 4C 61 7A 65 72 20 4D 69 63 72 6F 53 79
stems Inc=]==:==          73 74 65 6D 73 20 49 6E 63 01 5D 00 01 3A 01 0D
=NEXT without FO          10 4E 45 58 54 20 77 69 74 68 6F 75 74 20 46 4F
R=Syntax=RETURN           52 06 53 79 6E 74 61 78 14 52 45 54 55 52 4E 20
without GOSUB=Ou          77 69 74 68 6F 75 74 20 47 4F 53 55 42 0B 4F 75
t of DATA=Illega          74 20 6F 66 20 44 41 54 41 10 49 6C 6C 65 67 61
l Quantity=Overf          6C 20 51 75 61 6E 74 69 74 79 08 4F 76 65 72 66
low=Out of Memor          6C 6F 77 0D 4F 75 74 20 6F 66 20 4D 65 6D 6F 72
y=Stack Overflow          79 0E 53 74 61 63 6B 20 4F 76 65 72 66 6C 6F 77
=Undefined State          13 55 6E 64 65 66 69 6E 65 64 20 53 74 61 74 65
```

```
]
  1 REM   VIEWER  by P. Hinkle
  5 INPUT "page"; p
 10 FOR j = 0 TO 240 STEP 16
 15 PRINT "      ";
 20 FOR i = 0 TO 15
 30 x = p*256+i+j
 40 t = PEEK(x)
 41 IF t = 12 OR t = 13 OR t = 16 OR t = 128 OR t = 10 THEN  t = 61
 42 IF t = 0 OR t = 7 OR t = 8 OR t = 9 THEN  t = 61
 43 IF t = 22 OR t = 24 OR t = 28 THEN  t = 61
 44 IF t > 159 AND t < 164 THEN  t = 61
 45 IF t = 148 OR t = 151 THEN  t = 61
 50 PRINT CHR$(t);
 60 NEXT i
 70 PRINT
 80 NEXT j
 90 GOTO 5
```

```
  1 REM    VIEWCHR
 10 FOR j = 0 TO 240 STEP 16
 15 PRINT "     ";
 20 FOR i = 0 TO 15
 40 t = x
 41 IF t = 12 OR t = 13 OR t = 16 OR t = 128 OR t = 10 THEN  t = 61
 42 IF t = 0 OR t = 7 OR t = 8 OR t = 9 THEN  t = 61
 43 IF t = 22 OR t = 24 OR t = 28 THEN  t = 61
 44 IF t > 159 AND t < 164 THEN  t = 61
 45 IF t = 148 OR t = 151 THEN  t = 61
 50 PRINT CHR$(t);
 51 x = x+1
 60 NEXT i
 70 PRINT
 80 NEXT j
100 INPUT x: PRINT CHR$(x); : GOTO 100
```

CHAPTER 6. BASIC

BASIC and the "OS" or operating system are in the 64K RAM space as outlined in the memory map. The best approach to identify routines where different commands are carried out is to decipher the tables of words which point to RAM. These routines can then be called from machine language programs, although in most cases it is easier to do everything in machine language yourself because the routines from BASIC require extensive setup.

The first table is on pages 1-3, beginning with GOSUB, GOTO, etc. Print out these pages of RAM with printmem and you will see the following pattern: number of word (token), address (2 bytes reversed), number of letters in word, word. For example, 02 AD 03 05 47 4F 53 55 42, means 2=token, 03AD=address, 5 letters, and GOSUB in ASCII. Token 1 has no letters and the same address as LET, which presumably means "ignore it". The address of GOSUB, 03AD, is to a table in page 3 after the word table which gives the number of routines (in this case 1), and the address (in this case 3D8C). In this way all the routine addresses can be obtained, except a group including STOP, NEW, etc. that have 03D0 which points to a 0, ie. no address. At the end of the word table there are some words and symbols which are used in conjunction with other words. These are given tokens only, with no addresses.

The next table of BASIC words is on page 3F (63), which also holds various pointers, the floating point accmulator (3F22-6), etc. This table of math functions is organized as: number of letters, word, 88 or A8, address.

A table of tape key words is on pages 4E and 4F. These words (OPEN, APPEND, READ, etc.) do not have tokens, and the address of each command is listed in order in the address table following the name table. Thus in my copy of BASIC OPEN is at 4E03, APPEND at 4E0F, etc. If you experiment with these routines do not use a tape you care about.

BASIC programs are stored in RAM on page CF (207) by line number (2 bytes reversed), followed by an address in page D0, D1 or higher. At the address is the tokenized line, based on the tokens in the first BASIC table and others. Print out pages 207-209 with printmem and compare it with a listing of printmem. Add new lines which do not do anything and print pages 207-209 again to see how the new line is stored.

Numeric variables are stored in pages CF, CE, etc. just below the tokenized program. The first two letters of each variable are in a table in page 6B (107) which lists the address of the variable. If variables have more that two letters, the remaining letters are in page CF (207) or vacinity. String variables are also listed in the variable table on page 6B, and are stored on page 6C and following. All these tables are in different locations if HIMEM or LOMEM are used, but they still point to each other in the same way.

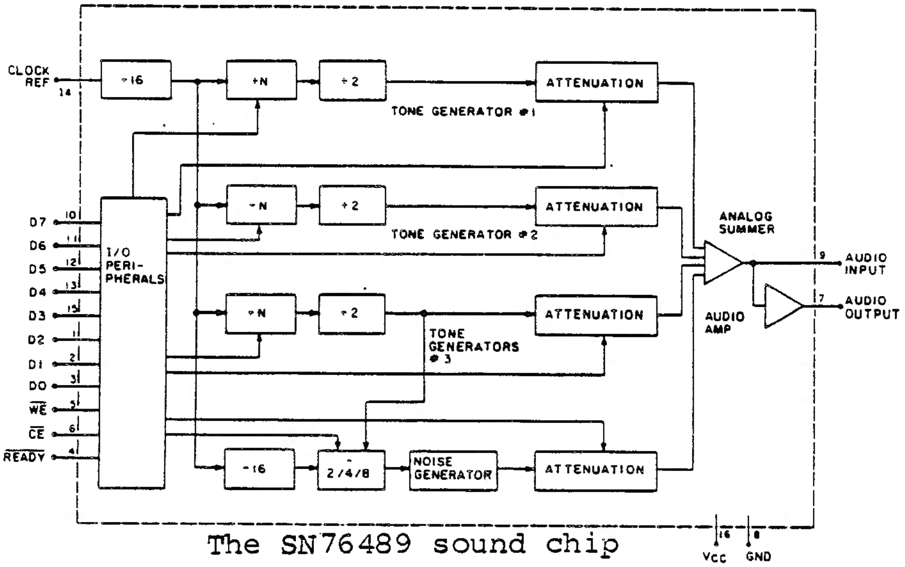
Input from tape is stored directly in a buffer in pages D4 (212) to D8 (216). This area contains the CATALOG of the last tape and the last program loaded, which appears exactly as it was typed in. The CATALOG lists the name of a file, the type, and 17 bytes beginning with 03 which presumably give information about where the file is on tape. This information plus disassembling the tape routines pointed to by the key words, should allow a complete analysis of the tape operating system (TOS), except that the tape is actually run by a 6801 with 2K ROM which is not accessible.

The operating system in RAM from E000 on is a series of routines called by BASIC and TOS which do inout functions, etc. The addresses of important routines (but not names!) are listed in a jump table starting at FC30. This table was made so that the OS could be changed without changing the entry points, which are the jump table. The OS does not seem to have been changed so far, unlike BASIC, as early and recent ADAMS have the same jump addresses. In general, routines from FC5D to FC9C have to do with the printer and routines from FD14 to FD3B have to do with the screen. Identifying these routines is a major task, however, which is best approached by analyzing them when they are called by BASIC.

One simple way to modify BASIC that can be fun to surprise people who know BASIC, is to change the key words in tables by poking new ASCII into RAM. It is easiest if the number of letters is not changed. After such changes BASIC will only respond to the new words.

CHAPTER 7. Sound

The sound chip on the Colecovision (top) board is the Texas Instruments SN76489A. I learned about this chip from articles in the December, 1980 Kilobaud Microcomputing by Steve Marum and in the July, 1982 Byte by Steve Ciarcia. It has three square wave tone generators and a noise generator, not nearly as sophisticated as the Commodore CID chip, but definately fun to play with. A block diagram of the chip is shown below.



The SN76489 sound chip

Texas Instruments uses an odd convention for describing the order of bits in a byte and calls the most significant bit (MSB) 0, or D0 for the data bus, instead of 7, or D7. In this description I have changed the TI nomenclature to the conventional designation of the MSB as 7 and the least significant bit (LSB) as 0. The pin numbers of the SN76489A are also shown in the figure. The chip is addressed via the WE (write enable), CE (chip enable) and ready inputs. It is mapped in the IN/OUT address space of the Z80 at F0 (actually the lower 5 bits are not decoded so any number between E0 and FF, or 224 and 255 in decimal, will access the chip using "OUT" instructions in machine language). There is only one port to address and the various functions are accessed by the numbers given to the port. These 8 bit numbers are divided up, as shown below, to give a 10 bit frequency value (divided between two bytes of input), a 3 bit control register which specifies eight functions, a 4 bit attenuator value which controls the volume, a noise type bit and a 2 bit noise clock value.

UPDATE FREQUENCY (2 BYTE TRANSFER)

1	REG ADDR			DATA			
	R0	R1	R2	F6	F7	F8	F9

FIRST BYTE

0	DATA						
	X1	F0	F1	F2	F3	F4	F5

SECOND BYTE

UPDATE NOISE SOURCE (SINGLE BYTE TRANSFER)

1	REG ADDR					SHIFT	
	R0	R1	R2	X	FB	NF0	NF1

UPDATE ATTENUATOR (SINGLE BYTE TRANSFER)

1	REG ADDR			DATA			
	R0	R1	R2	A0	A1	A2	A3

Types of data bytes sent to the SN76489.

When the MSB is 1 the next three bits are the control register that specifies the meaning of the lower 4 bits. When the MSB is 0 the lower 6 bits are the most significant bits of the 10 bit frequency value for the most recently specified tone generator. The frequency of the square wave produced is the clock frequency divided by 32 times the 10 bit number specified as the frequency value.

The control register, specified by R0, R1, and R2 indicates the following functions:

0	tone 1 frequency
1	tone 1 volume
2	tone 2 frequency
3	tone 2 volume
4	tone 3 frequency
5	tone 3 volume
6	noise type
7	noise volume

The noise generator can be controlled to produce different types of noise at different volumes. The types are white (hiss) and perodic (motors). The frequency generating both noise types has 4 values specified by the 2 bit number formed by NF1 and NF0, or can be driven by voice 3, allowing continuously variable noise frequencies of phaser type sounds.

In practice it is likely that you will program the SN76489A in BASIC via a short machine language subroutine, and so the numbers you will use will be decimal. The table below shows the numbers used to control the chip in decimal.

Sound control numbers in decimal.

	Pitch		Volume	
	first byte	second byte	high	off
voice 1	128-143	0-63	144-159	
voice 2	160-175	0-63	176-191	
voice 3	192-207	0-63	208-223	

noise	224-227	perodic	(227=voice 3)
	228-231	white	(231=voice 3)
	240-255	volume	(255=off)

Pitch control

I=frequency value =0 to 1023
note frequency = clock /32*I
for voice 1: byte 1 (128-143) = 128*I-INT(I/16)*16
             byte 2 (0-63) = INT(I/16)

For voice 2 or 3 start with 160 or 192 for the first byte, instead of 128. For a chromatic scale use I=120,127,134,142,150,159,169,179,190,201,213,225,240 and multiples of these numbers. This scale was generated by dividing an octave (factor of two in frequency) into twelve notes spaced equally on a logrithmic scale. The frequency of the next note (half step) is the frequency of the current note times the twelth root of two.

To pass numbers to the SN76489 from BASIC a short machine language subroutine is needed. A simple example is:

```
   LD  A,n
   LD  C,FO
   OUT(C), A
   RET
```

This code can be poked into RAM as illustrated in the following programs. The first can be used to experiment with the chip, and the second is an interesting random music generator.

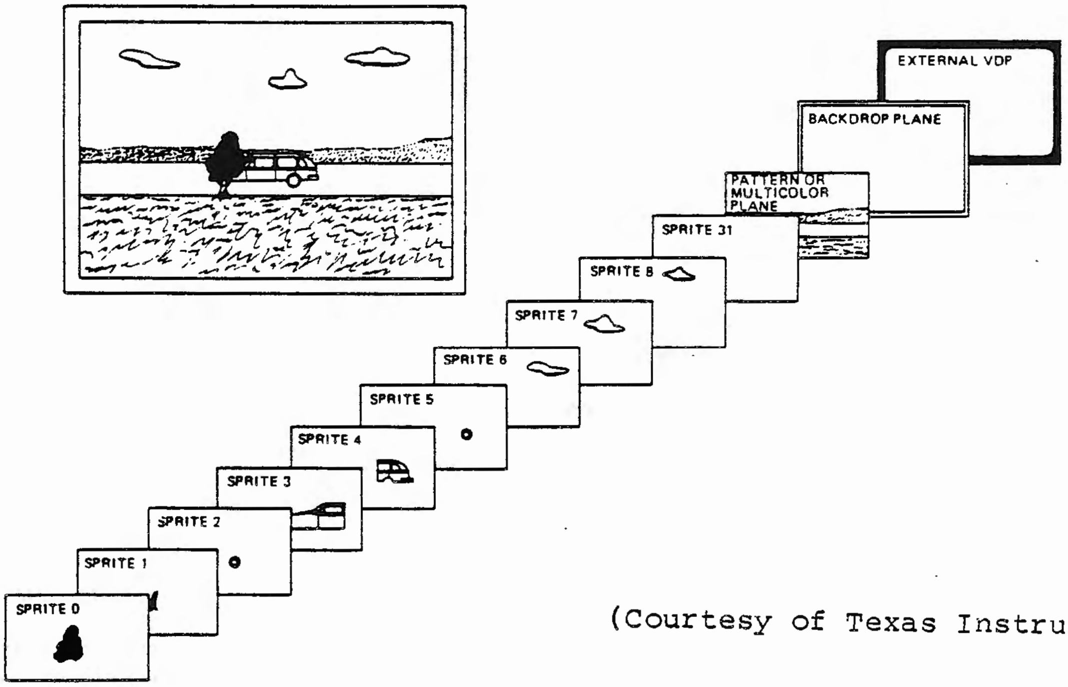
```
]   5 REM  SOUNDTEST
    6 REM
   10 HIMEM :53000
   14 REM     poke in machine code
   15 DATA          62,0,14,245,237,121,201
   20 FOR x = 1 TO 7
   30 READ d: POKE 53000+x, d
   40 NEXT
  100 INPUT "number (0-255)"; n
  110 POKE 53002, n
  120 CALL 53001
  130 GOTO 100

]
```

```
]
    5 REM    RNDMUSIC
    6 REM
   10 HIMEM :53000
   14 REM    poke in machine code
   15 DATA          62,0,14,245,237,121,201
   20 FOR x = 1 TO 7
   30 READ d: POKE 53000+x, d
   40 NEXT
  190 FOR t = 300 TO 1 STEP -1
  199 REM    think of note
  200 v = RND(9)*255
  202 IF v > 223 AND v < 240 THEN  v = ~~231~~ 227
  205 REM  play note
  210 POKE 53002, v
  220 CALL 53001
  230 REM  delay
  240 FOR w = 1 TO t: NEXT
  250 NEXT t
  260 GOTO 190
```

CHAPTER 8. The Video Display Processor

The video signal to the TV is produced in the ADAM by the Texas Instruments video display processor (VDP), TMS9918A. It is very different from the Apple graphics in BASIC, and has modes, patterns, backgrounds, and sprites. I learned about this chip from an article in August, 1982 Byte by Steve Ciarcia and from a book sent free from Texas Instruments, Semiconductor Group, P.O.Box 1143, Houston TX, 77001. This book is hard to relate to the ADAM, and has all examples in 9900 assembly language, but it has all the facts. I will try to distill them into these notes.

The VDP is organized as multiple screens (or planes) in series, as shown below. The sprites are in the foreground and can be used for moving or stationary objects. Sprites can be moved by simply changing their x and y coordinates in a table. They move cleanly without changing the colors of nearby objects, as occurs with Coleco`s implementation of Apple graphics.



(Courtesy of Texas Instruments)

Behind the 32 sprites is a pattern plane which is a matrix of blocks, each 8x8 pixels that can be defined by the user. These pattern blocks are used to form the text in BASIC, but could also be used for landscapes etc. Behind the pattern plane is a backdrop plane which specifies the color of all pixels not set by the previous planes.

Throughout, transparency is a possible "color". Finally, behind the background plane is the possibility, not implemented on the ADAM, of having the output of the VDP viewed on top of any other TV picture. With a TV camera, video recorder and a minor modification to the ADAM, you could make home videos of your children playing with sprites!

After some experiments where I could change the screen output but wasn't sure why (eg. CALL 57545), I looked inside and found that the three address lines of the VDP are connected to the Z80 as follows: mode to A0 of the Z80, CSR (chip select read) and CSW (write) to A5, A6, WR (write read), and IORQ (inout request) of the Z80 via a 74138 decoder such that the chip appears in the inout space as 160, 161 to 190 or 191 decimal even-odd pairs. I will use 190 and 191. Knowing this allowed tests with short machine language subroutines illustrated later. I will first describe the VDP chip and then give examples of how to use it directly.

The 9918A is a very complex chip which is connected to 16K of RAM,"VRAM", for its own use. It has four modes of operation which, together with the arrangement of tables in VRAM and a few other things, are specified by eight control registers which can be written to but not read. The control registers, a read-only register, and VRAM are accessed by the Z80 according to the following table.

Operation	Bits	CSW	CSR	Mode	inout
write to register					
byte 1:data	D7------------D0	0	1	1	191
byte 2:reg.sel.	1 0 0 0 0R2R1R0	0	1	1	191
Write to VRAM					
byte 1:address	A7------------A0	0	1	1	191
byte 2:address	0 1 A13------A8	0	1	1	191
byte 3:data	D7------------D0	0	1	0	190
Read from register 8					
byte 1:data	D7------------D0	1	0	1	191
Read from VRAM					
byte 1:address	A7------------A0	0	1	1	191
byte 2:address	0 0 A13------A8	0	1	1	191
byte 3:data	D7------------D0	1	0	0	190

Bytes 1 and 2 of the write to VRAM procedure are needed for only the first byte transfered. Additional data bytes are automatically put into the next higher addresses. In addition, I have not yet made the read from VRAM procedure work on the ADAM, which may be because of some timing problems.

CONTROL REGISTERS

Register 0,
contains two option control bits.
bit 1, M3=1 specifies graphics mode 2
bit 0, EV=1 enables external input. Keep EV=0.

Register 1,
contains seven option control bits.
bit 7, 4/16K RAM. Keep at 1 (16K).
bit 6, 0 blanks display. Keep at 1.
bit 5, interrupt enable. 1= enabled.
bit 4, M1=1 specifies text mode.
bit 3, M2=1 specifies multicolor mode.
bit 2 always =0.
bit 1, size. 0= 8x8 sprites, 1= 16x16 sprites.
bit 0, mag. 0= sprites x1, 1= sprites x2.

Register 2.
The upper 4 bits are always 0. The number in the lower 4 bits (0 to 15) times $400 (1024) is the base address in VRAM of the pattern name table. Each byte in the name table corresponds to a region on the screen, and the number in the table specifies the pattern to be displayed there.

Register 3.
This number (0 to 255) times $40 (64) is the base address in VRAM of the color table.

Register 4.

This number (0 to 7) times $800 (2048) is the base address in VRAM of the pattern generator table.

Register 5.

This number (0 to 127) times $80 (128) is the base address in VRAM of the sprite attribute table.

Register 6.

This number (0 to 7) times $800 (2048) is the base address in VRAM of the sprite pattern generator table, where shapes of sprites are defined.

Register 7.

The upper 4 bits (0 to 15)x16 specify the color of text in the text mode (not used by Coleco). The lower 4 bits (0 to 15) specify the background color in text mode and backdrop color in other modes.

Register 8.

This is the status, read-only register. It contains three flags and a fifth sprite number and can be read during programs to check certain conditions. Reading the register clears all flags to 0.

bit 7, flag F. Interrupt flag, is set to 1 at the end of the last raster scan on the TV.

bit 6, fifth sprite flag (5S). Only four sprites are allowed on any given horizontal scan line. When a fifth sprite crosses a horizontal line this flag is set to 1 and the number of the sprite is placed in the lower 5 bits of the register.

bit 5, flag C. This coincidence or collision flag is set to 1 when two sprites collide. Collisions are checked only 60 times per second and so may be missed.

COLOR CODES

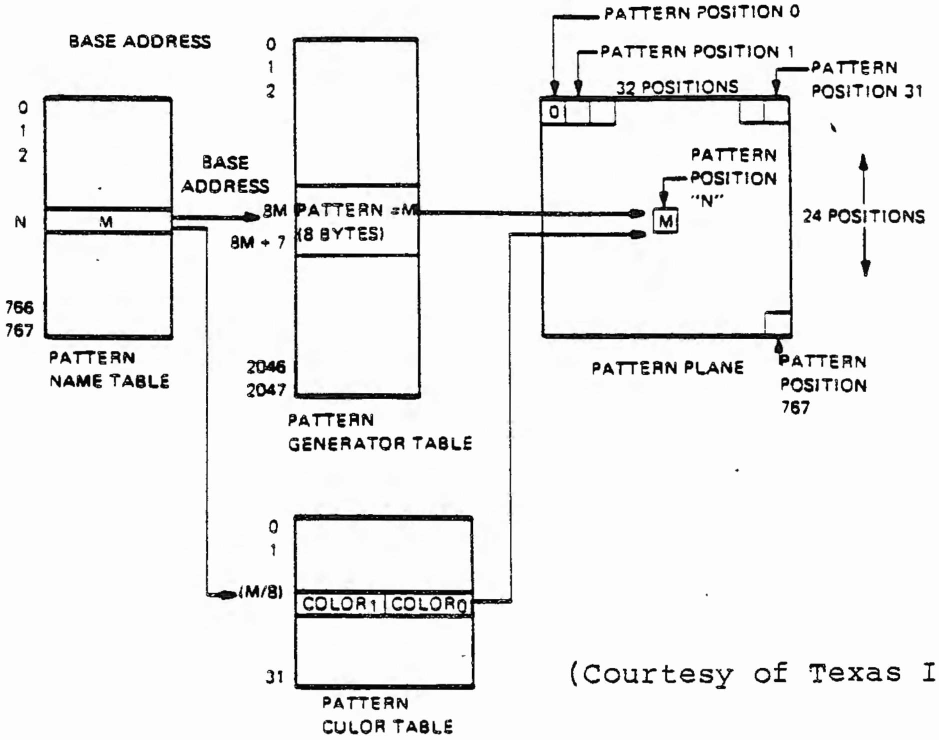
The colors that are specified for sprites, backgrounds, etc. have the following codes.

0	transparent	8	medium red
1	black	9	light red
2	medium green	10	dark yellow
3	light green	11	light yellow
4	dark blue	12	dark green
5	light blue	13	magenta
6	dark red	14	gray
7	cyan	15	white

MODES

Graphics mode 1. (M1,M2,M3=0)

This is the simplest graphics mode and, strangely, is used by BASIC to display text. The pattern plane is divided into 32 columns by 24 rows of blocks (768) each containing 8x8 pixels. Three tables in VRAM are used to create the pattern plane, as shown below.



(Courtesy of Texas Instruments)

The pattern name table is a 768 byte block of VRAM beginning on a 1K boundary pointed to by control register 2. Each byte corresponds to a region of the screen (ordered from left to right and top to bottom) and specifies the number of the pattern in the pattern generator table and the n/8th entry in the pattern color table to be displayed at that point. More that one pattern name table can be made, allowing rapid switching between pattern planes by simply changing the number in control register 2. The color table has only 32 numbers, and is pointed to by control register 3 times $40. Each number specifies the color of 1`s in the pattern by the top 4 bits and of 0`s by the bottom 4 bits. One number in the color table applies to 8 patterns in the pattern generator table, so patterns of the same colors should be grouped together.

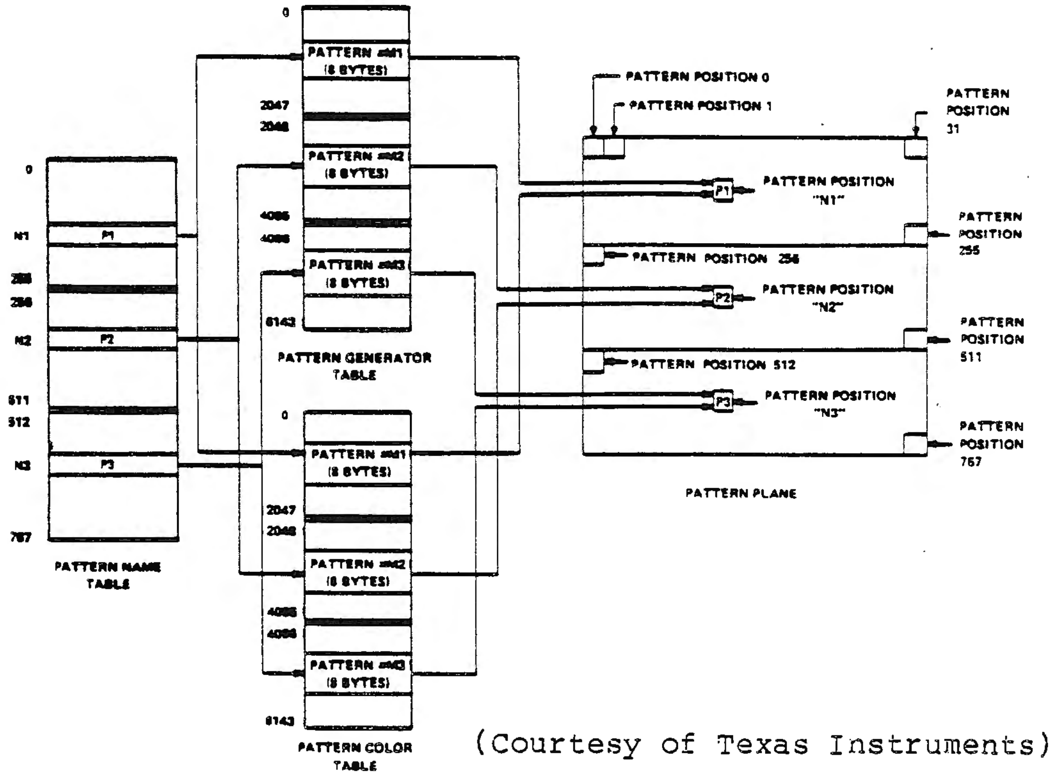
The pattern generator table, pointed to by control register 4, consists of 8 bytes which form an 8x8 matrix of 1`s and 0`s as illustrated below.

BYTE	BINARY	HEX
0	0 0 1 1 1 1 0 0	3C
1	0 1 1 1 1 1 1 0	7E
2	1 1 1 1 1 0 1 1	FB
3	1 1 1 1 1 1 1 1	FF
4	1 1 1 1 1 0 0 0	F8
5	1 1 1 1 1 1 0 0	FC
6	0 1 1 1 1 1 1 0	7E
7	0 0 1 1 1 1 0 0	3C

The same type of 8x8 matrix is used for sprites. As many as 256 patterns can be defined, taking 2048 bytes, but any smaller number can also be defined. An all-0 pattern should be included to point to for blank areas of the screen. Sprites can be used in all graphics modes, and the only limitation in mode 1 is that each 8x8 block in the pattern plane can have only two colors.

Graphics mode 2 (M3=1,M2 and M1=0)

Graphics mode 2 enhances the resolution over mode 1 by increasing the length of the pattern generator table from 2048 bytes to 6114 bytes (x3), and increasing the color table from 32 bytes to 6144 bytes. This allows every pixel to be set independantly and the color to be specified every 4 pixels (equal numbers of pattern and color bytes means 4 bits of color, or 1 color, for 4 bits of pattern, or 4 pixels). The pattern groups of 8 bytes are addressed by the name table as shown below.

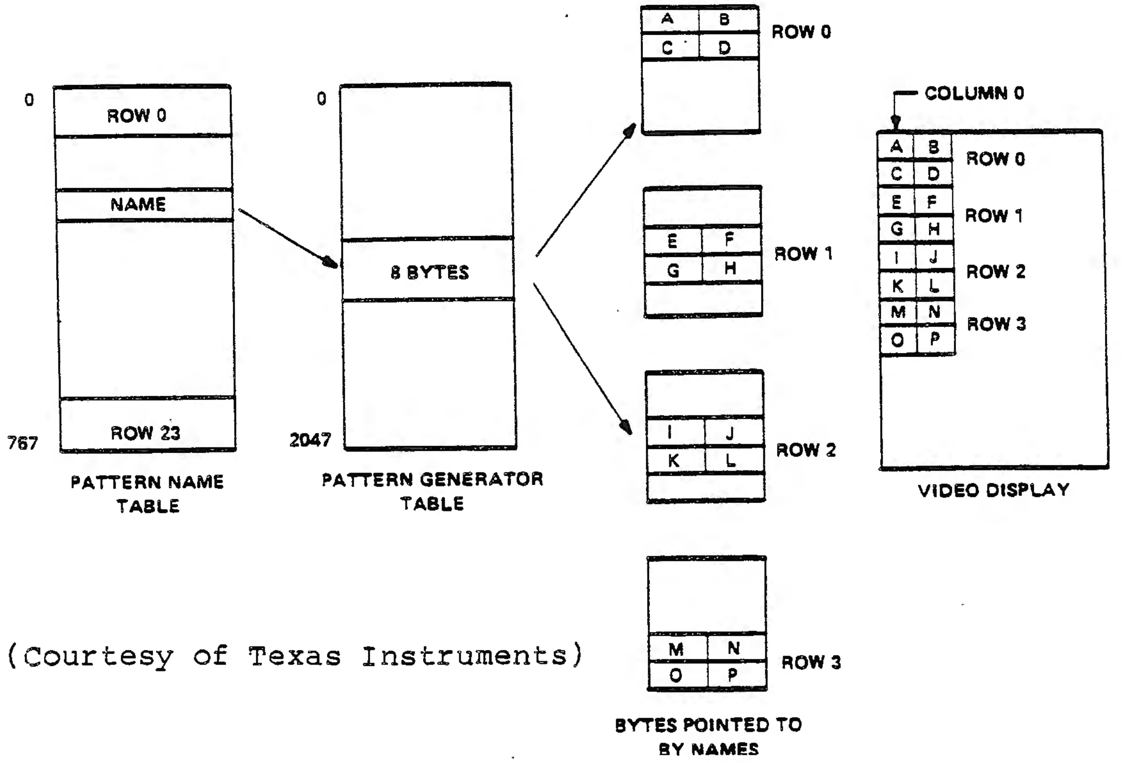


(Courtesy of Texas Instruments)

This mode is used for hires in BASIC but is awkward for such use because it was designed for backgrounds only. Sprites can be used in mode 2, and it is ideal to combine sprite routines with BASIC hires.

Multicolor Mode (M2=1, M1 and M3=0)

This mode is like lores graphics in BASIC, but gives a 64x48 block (of 4x4 pixels) display with any color allowed for any block. The blocks are specified as shown below.

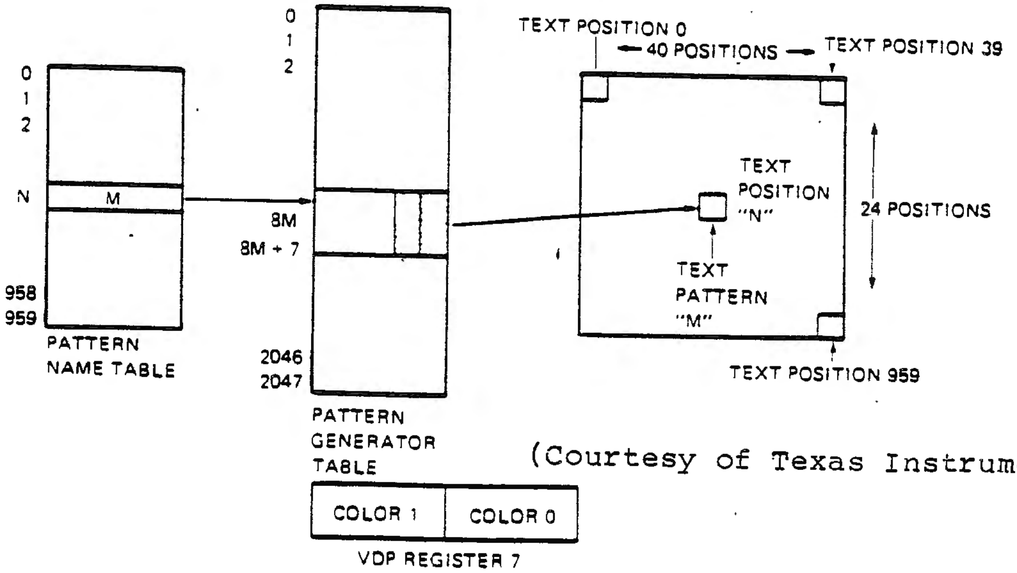


(Courtesy of Texas Instruments)

An entry in the pattern name table specifies 4 blocks, such as ABCD in row 0. If a byte in the name table which is in row 1 addresses the same pattern generator block, the colors will be EFGH, given by the third and forth bytes in the pattern. The first two bytes in a pattern apply to rows 0,4,8,12,16,20. The second two bytes apply to rows 1,5,9,13,17,21, etc.

Text Mode (M1=1, M2 and M3=0)

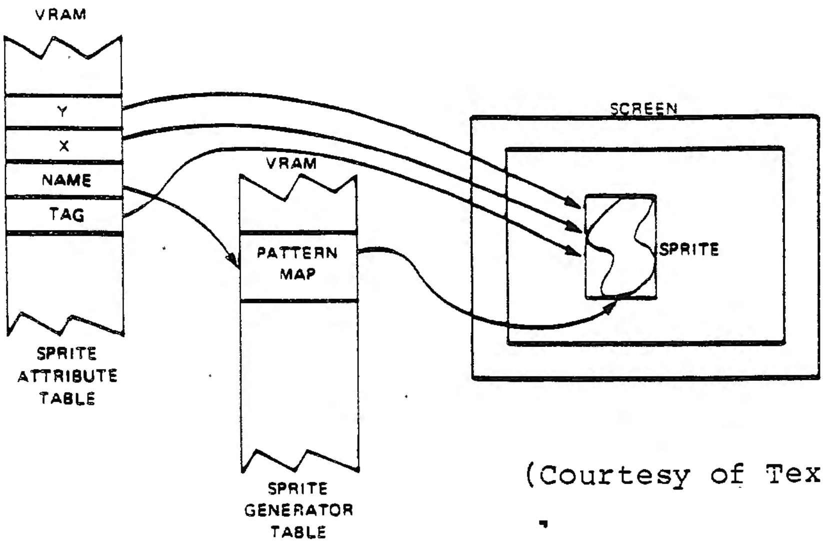
In this mode the screen is divided into a grid of 40x24 patterns (presumably letters and numbers), and the colors are specified by control register 7. Each pattern is 6 pixels across by 8 down, and the lowest two bits of each byte in the pattern generator table are ignored.The mapping in text mode is shown below. Sprites are not available in text mode.



(Courtesy of Texas Instruments)

SPRITES

Sprites are controlled by 4 bytes in the sprite attribute table, which specify the position of the sprite on an approximately 256x92 grid, point to the sprite generator table block, and specify the color of the sprite. The addressing mechanism is shown below.



(Courtesy of Texas Instruments)

In the sprite attribute table a sprite is defined by 4 bytes. The first byte is the vertical position, and the second byte is the horizontal position. The third byte is the sprite name which points to an 8 byte block in the sprite generator table. The forth byte has the sprite color in the lower 4 bits, 0's in bits 4,5, and 6, and something called the early clock bit in the top bit. When this bit is 1 the sprite is moved 32 pixels to the left, and it can probably be safely ignored. The sprite attribute table is ended by the number 208 decimal, so that the number of sprites showing can easily be changed from a maximum of 32 to less by inserting 208 in the vertical position byte of one sprite, blocking display of it and all further sprites in the attribute table.

The size and resolution of sprites is controlled by the size and mag bits in control register 1, as follows.

SIZE	MAG	Area	Resolution	Bytes/pattern
0	0	8x8	single pixel	8
1	0	16x16	single pixel	32
0	1	16x16	2x2 pixels	8
1	1	32x32	2x2 pixels	32

To use the VDP you must first decide where the various tables will be in VRAM, and then fill them. To integrate your own graphics with BASIC graphics you must avoid using the same areas of VRAM that BASIC uses. This would logically be done by reading VRAM to see what is there, but as I mentioned earlier this doesn't work. Combining sprites with hires (HGR or HGR2) works, however, if all tables are as high as possible.

If you lose control of a program and cannot see why because the screen is in an altered mode, typing control C, (return), and TEXT once or twice will often restore control because the TEXT command puts all the proper tables in VRAM.

The program below makes a sprite and moves it on the HGR2 screen. The second program is a good shape table maker, and the third a sprite editor which makes sprite generator tables and bsaves them on tape to be used in your programs. The latter two were written by my son who is fourteen.

```
   5 REM      SPRITE DEMO
   6 HGR2
  10 HIMEM :51399
  19 REM       load machine language code
  20 DATA       62,0,211,191,201,62,00,211,190,201
  30 FOR x = 51400 TO 51409: READ p: POKE x, p: NEXT
  34 REM
  35 REM   background
  36 FOR s = 1 TO 25
  37 HCOLOR  = 5
  38 HPLOT 100+1*s, 0 TO 10*s, 191
  39 NEXT
  40 REM
  50 REM       load sprite generator
  55 a = 0: GOSUB 1000: a = 120: GOSUB 1000
  60 DATA   60,126,195,219,219,195,126,60
  70 FOR x = 1 TO 8
  80 READ d: GOSUB 1100
  90 NEXT
 100 REM       load sprite attribute
   110 a = 128: GOSUB 1000: a = 127: GOSUB 1000
   120 d = 70: GOSUB 1100: GOSUB 1100: d = 0: GOSUB 1100
   121 d = 7: GOSUB 1100: d = 208: GOSUB 1100
   199 REM
   200 REM       load control registers
   230 a = 127: GOSUB 1000
   240 a = 133: GOSUB 1000
   250 a = 7: GOSUB 1000
   260 a = 134: GOSUB 1000
   299 REM
   300 REM       MOVE IT
   310 t = t+.05
   320 x = 60*SIN(t)+70
   330 y = 60*COS(t)+70
   340 a = 128: GOSUB 1000
   350 a = 127: GOSUB 1000
   360 d = INT(x): GOSUB 1100
   370 d = INT(y): GOSUB 1100
   380 GOTO 310
   999 REM
  1000 POKE 51401, a
  1010 CALL 51400
  1020 RETURN
  1100 POKE 51406, d
  1110 CALL 51405
  1120 RETURN
```

```
]
    1 REM  -shape table maker by Ben Hinkle.modified from program by Mark Pelcza
rski in Softtalk, July 1982
    5 HIMEM :51455
    7 INPUT "how many shapes in the shape table?"; e
   10 w = 51456: POKE w, e: POKE w+1, 0: POKE w+2, 2*e+2: POKE w+3, 0: w = w+4:
POKE 16766, 0: POKE 16767, 201
   15 w = (2*e+2)+51456: il = 2*e+2: sn = 1: st = 51460
   20 p = 0: POKE w, 0: POKE w+1, 0: sw = 1: GR
   22 x = 20: y = 20
   23 PRINT "use the arrow keys to move,     'home' to plot,and 'f'to finish the
shape.";
   24 PRINT "This is shape #"; sn: PRINT "you are now at (x,y):";
   25 il = w-51456
   27 VTAB 24: HTAB 22: PRINT "      "; : VTAB 24: HTAB 22
   28 PRINT x; ","; y;
   90 COLOR  = 13: PLOT x, y
  100 GET a$: a = ASC(a$)
  110 IF a$ = "f" THEN 300
  111 COLOR  = 0: PLOT x, y
  113 IF p = 0 THEN 120
  115 COLOR  = 4: PLOT x, y
  120 IF a = 128 THEN  p = 4: GOTO 90
  130 IF a = 160 THEN  m = 0: y = y-1: GOTO 200
  140 IF a = 161 THEN  m = 1: x = x+1: GOTO 200
  150 IF a = 162 THEN  m = 2: y = y+1: GOTO 200
  160 IF a = 163 THEN  m = 3: x = x-1: GOTO 200
  180 GOTO 25
  200 v = m+p
  205 p = 0
  210 IF sw = 1 THEN  sw = 2: vl = v: POKE w, v: POKE w+1, 0: GOTO 25
  220 IF v+vl = 0 THEN  POKE w, 88: w = w+1: POKE w, 0: vl = 0: GOTO 25
  230 IF v = 0 THEN  POKE w, vl+192: w = w+1: POKE w, 0: vl = 1: GOTO 25
  240 v = v*8+vl: POKE w, v: w = w+1
  250 sw = 1: POKE w, 0
  260 GOTO 25
  300 IF sw = 2 THEN  POKE w, vl: w = w+1
  305 POKE w, 0
  310 GOSUB 2000
  311 HOME: INPUT "are you satified with this shape (y/n)?"; a$: IF a$ = "y" THE
N 315
  312 IF PEEK(w-1) = 0 THEN  GR: GOTO 22
  313 w = w-1: GOTO 312
  315 w = w+1: il = w-51456
  317 IF sn < e THEN 350
  318 HOME: INPUT "Do you want to save it (y/n)?"; x$: IF x$ = "n" THEN  END
  320 INPUT "Shape table name?"; a$
  330 PRINT CHR$(4); "bsave "; a$; " ,a51456, l"; w-51455
  340 TEXT: PRINT "done"
  345 END
  350 sn = sn+1
  360 p = INT(il/256)
  370 POKE st+1, p: POKE st, il-p
  400 st = st+2
  410 GR
  420 GOTO 22
 2000 HGR: HCOLOR  = 12: SCALE  = 1: ROT  = 0
 2010 DRAW sn AT 100, 100
 2020 RETURN

]
```

'BSAVE MUSIC, A27407, L1947'

'BLOAD MUSIC, A27407'

'CALL 27407'

```
]
   2 REM      sprite editor     by Ben Hinkle
   3 REM
   4 HIMEM :50999: ra = 51000
   5 TEXT: PRINT: PRINT: INPUT "How many sprites would you like to have (1-32)?
"; n: IF n < 1 OR n > 32 THEN 5
  10 PRINT: PRINT: PRINT: PRINT "Would you like to have:": PRINT
  12 PRINT "  1.8x8 sprites": PRINT "  2.16x16 sprites": PRINT: INPUT "(1,2)?";
 s
  20 IF s < 1 OR s > 2 THEN  TEXT: GOTO 10
  30 rb = s*8+11: bb = s*8+1: FOR d = 1 TO n
  50 GR: COLOR  = 10: x = 11: y = 1
  60 VLIN 0, bb AT 10: VLIN 0, bb AT rb: HLIN 10, rb AT 0: HLIN 10, rb AT bb
  70 PRINT "    arrow keys to move cursur"
  80 PRINT "'a'-plot", "'d'-erase"
  90 PRINT "'return' when done with sprite"
  95 PRINT "sprite #"; d;
 100 COLOR  = 12: PLOT x, y
 110 GET a$: p = ASC(a$)
 120 IF e = 1 THEN  COLOR  = 8: PLOT x, y: GOTO 140
 130 COLOR  = 0: PLOT x, y
 140 IF p = 97 THEN  COLOR  = 8: PLOT x, y
 150 IF p = 100 THEN  COLOR  = 0: PLOT x, y: e = 0
 155 IF p = 13 THEN 200
 160 IF p = 163 AND x-1 > 10 THEN  x = x-1: e = 0
 165 IF p = 161 AND x+1 < rb THEN  x = x+1: e = 0
 167 IF p = 160 AND y-1 > 0 THEN  y = y-1: e = 0
 170 IF p = 162 AND y+1 < bb THEN  y = y+1: e = 0
 180 IF SCRN(x, y) = 8 THEN  e = 1
 190 GOTO 100
 200 IF s = 2 THEN 280
 210 aa = 8: ab = 1: ac = 18: ad = 11: GOSUB 230
 220 NEXT d: GOTO 400
 230 FOR y = ab TO aa: i = 0
 240 FOR x = ac TO ad STEP -1
 250 IF SCRN(x, y) = 8 THEN  i = i+2^(ac-x)
 260 NEXT x: POKE ra, i: ra = ra+1: NEXT y
 270 RETURN
 280 aa = 8: ab = 1: ac = 18: ad = 11: GOSUB 230
 290 aa = 16: ab = 9: ac = 18: ad = 11: GOSUB 230
 300 aa = 8: ab = 1: ac = 26: ad = 19: GOSUB 230
 310 aa = 16: ab = 9: ac = 26: ad = 19: GOSUB 230
 320 NEXT d
 400 TEXT: PRINT: PRINT: INPUT "Would you like to print out the sprites?"; a$
 410 IF a$ <> "y" AND a$ <> "n" THEN 400
 420 IF a$ = "n" THEN 500
 430 PR #1: FOR m = 51000 TO ra-1 STEP 8
 435 FOR h = 0 TO 7
 440 PRINT PEEK(m+h); " "; : NEXT h: PRINT: NEXT m
 450 PR #0
 500 TEXT: PRINT: PRINT: INPUT "Would you like to save the     sprites (y/n)?";
 a$
 510 IF a$ <> "y" AND a$ <> "n" THEN 500
 520 IF a$ = "n" THEN  PRINT "End of program": END
 530 INPUT "Type in the name for the file:"; a$: ra = ra-51000
 540 PRINT CHR$(4); "bsave "; a$; ",a51000,1"; ra
 550 PRINT "done"
```

name   mem loc

BSAVE- File, A22402, L1917

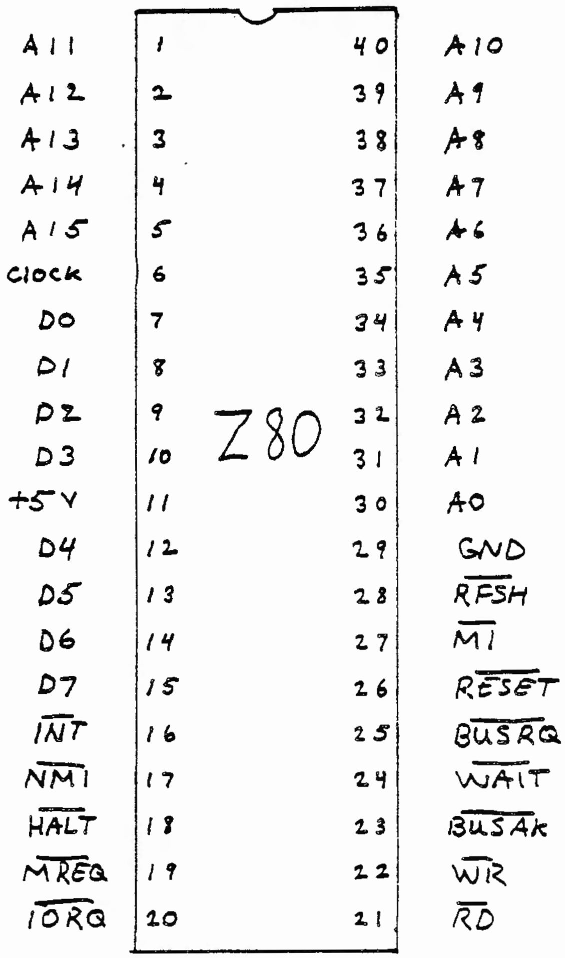
Bload- File, A 27407

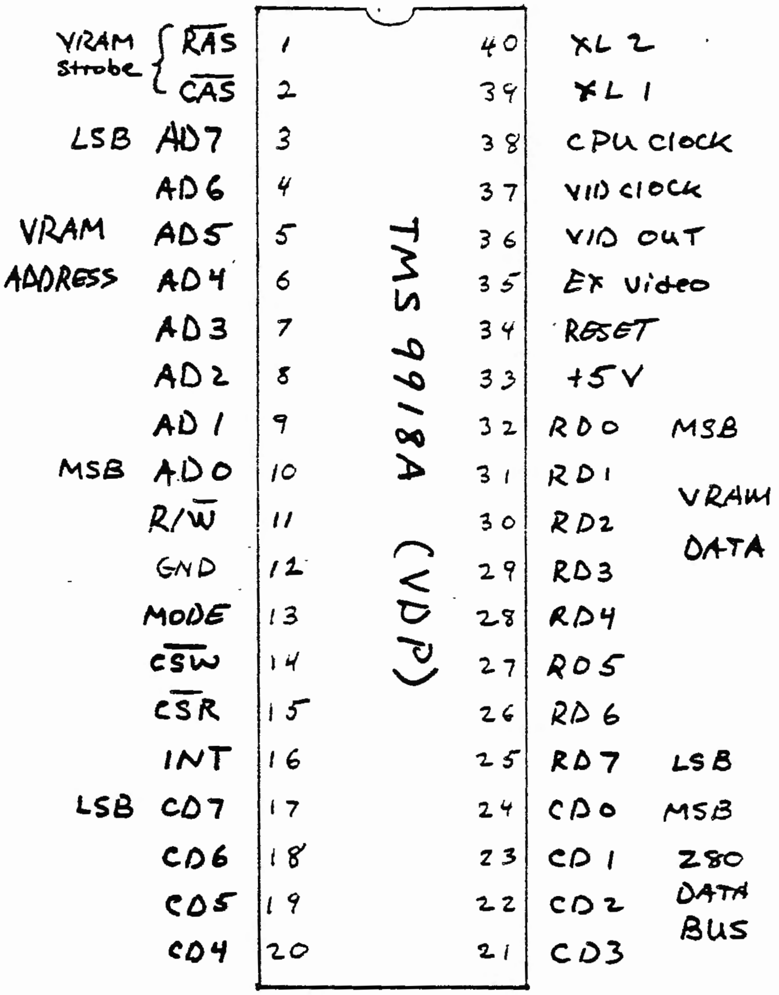
CALL 27407 - RUN

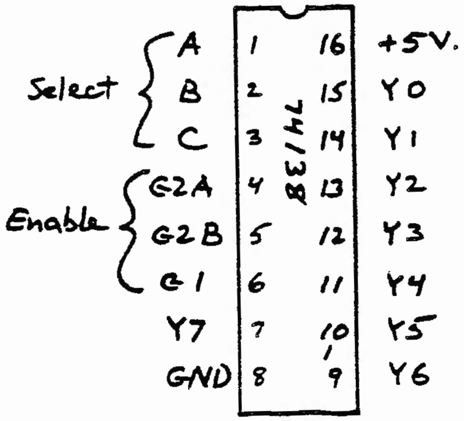
CHAPTER 9. Pinouts

The following chips are diagramed:

Z80 microprocessor
TMS9918A video display processor
SN76489A sound generator
7400 quad NAND gates
7402 quad NOR gates
7404, 7405 hex inverters
7474 dual flip-flops
74126 3-state bus driver
74138 3 to 8 line decoder
74157 quad data selectors
74541 octal bus driver

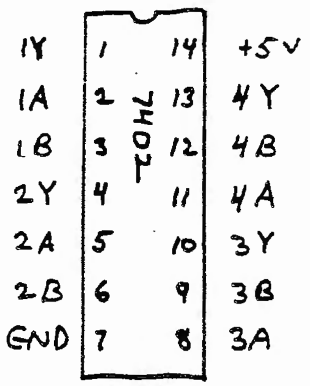

Z80
A11 1 40 A10
A12 2 39 A9
A13 3 38 A8
A14 4 37 A7
A15 5 36 A6
Clock 6 35 A5
D0 7 34 A4
D1 8 33 A3
D2 9 32 A2
D3 10 31 A1
+5V 11 30 A0
D4 12 29 GND
D5 13 28 $\overline{RFSH}$
D6 14 27 $\overline{M1}$
D7 15 26 $\overline{RESET}$
$\overline{INT}$ 16 25 $\overline{BUSRQ}$
$\overline{NMI}$ 17 24 $\overline{WAIT}$
$\overline{HALT}$ 18 23 $\overline{BUSAK}$
$\overline{MREQ}$ 19 22 $\overline{WR}$
$\overline{IORQ}$ 20 21 $\overline{RD}$



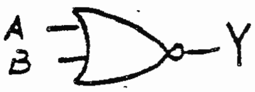


3 to 8 line Decoders

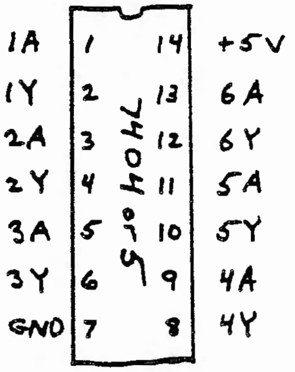
Enable		Select			Out Put (all H except specified)
G1	G2A+B	C	B	A	
X	H	X	X	X	-
L	X	X	X	X	-
H	L	L	L	L	Y0
H	L	L	L	H	Y1
H	L	L	H	L	Y2
H	L	L	H	H	Y3
H	L	H	L	L	Y4
H	L	H	L	H	Y5
H	L	H	H	L	Y6
H	L	H	H	H	Y7



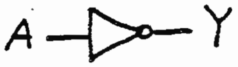
+NOR



$Y = \overline{A+B}$

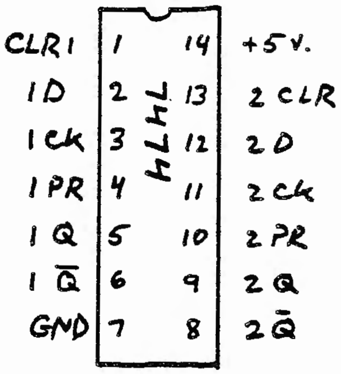


HEX INVERTERS



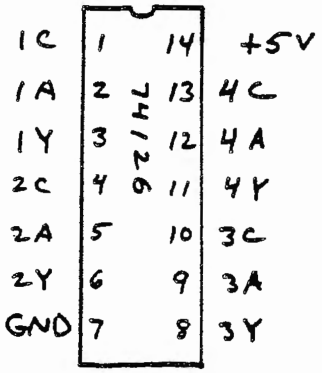
$Y = \bar{A}$

7405 has open-collector outputs

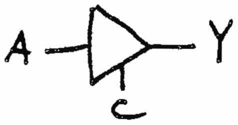


DUAL FLIP-FLOP

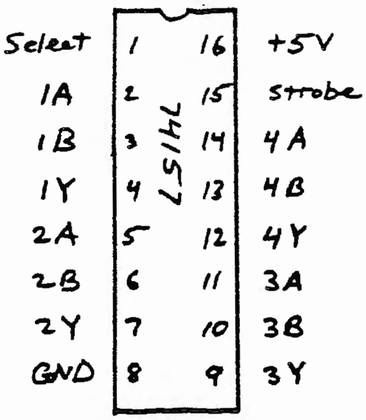
INPUTS				OUTPUTS	
PR	CLR	ck	D	Q	$\bar{Q}$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	–	–
H	H	↑	H	H	L
H	H	↑	L	L	H
H	H	L	X	Q_0	$\bar{Q}_0$



QUAD BUS BUFFER
3-state OUTPUTS

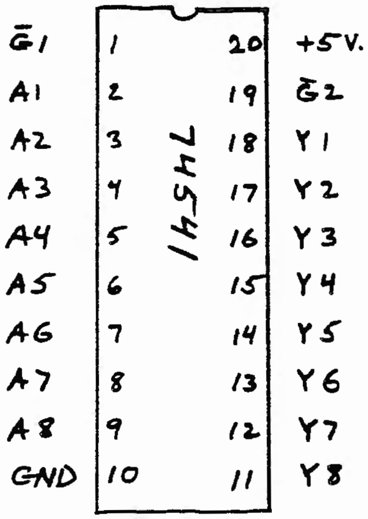


Y = A, output is disabled when C is low.

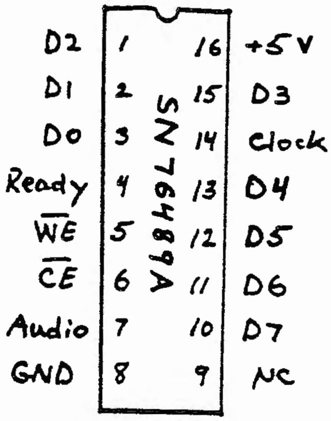


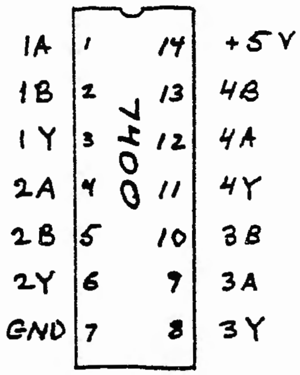
QUAD 2 to 1 line
DATA Selectors

Strobe	Select	A	B	OUT(Y)
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H



OCTAL BUFFERS
3-State OUTPUTS
NON-INVERTING





+ NAND

$Y = \overline{AB}$