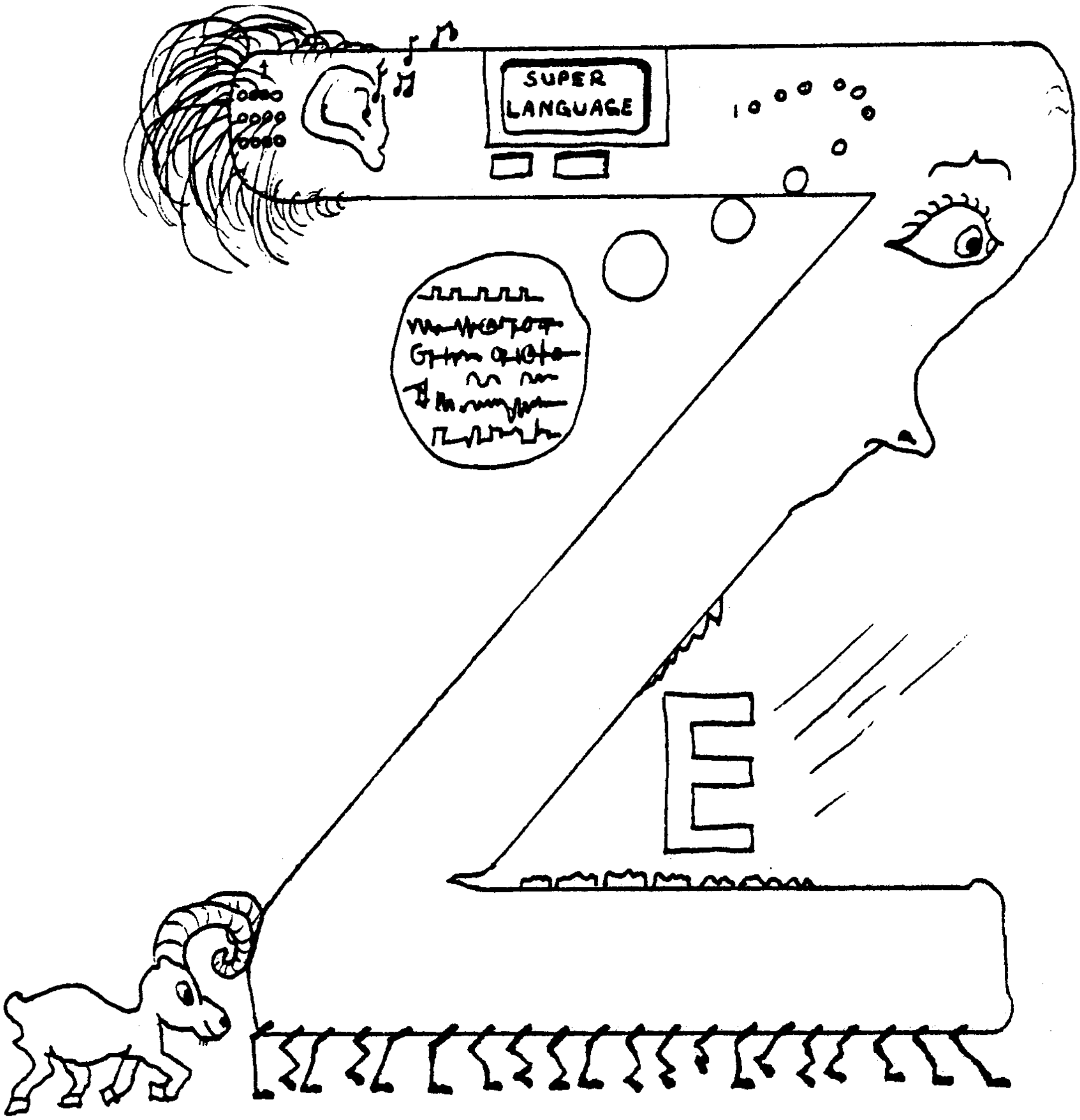


MINICOURSE

TABLE OF CONTENTS

cont. →

INTRODUCTION

The custom chips in the arcade are some of the few, mass produced, and therefore low cost devices available to the consumer for animation and graphics. Since very little has been written about the 'machine' aspects of these chips, it is my objective to provide some information on a few facets of their operation. Hopefully this will dispell some of the mystery regarding the software techniques necessary to take advantage of some of their capabilities.

Though this manual is meant to be more or less a beginners guide to machine programming with the Bally, it is helpful to have somewhat of an understanding of a few programming techniques with the Bally Basic language. Also, good adjuncts to this book would be one of the many paperbacks on the Z-80, and the Z-80 Assembly Language Programming Manual, Which is published by Zilog .

As you read and do the experiments refer to the Z-80 code listing at the end of this book,(Pgs. 43-45) Keep a notebook. If you fully understand everything that is covered, you will be able to do many things that are impossible with Basic alone. You should also end up with a good understanding of how machine language routines can be used with Basic to increase speed, simplify functions, and save bytes. Many ideas, examples, and analogies will be made in reference to Bally Basic.

I've tried to construct this book as something which I wish I had when I started programming in machine language. Hopefully this information will prove to be of benefit to those venturing into machine programming. I hope to continue where this leaves off in Course Two which will cover uses of RST instructions , and onboard ROM routines. If you have any comments, or information that you would like to share, I would appreciate hearing from you. I will be consolidating and organizing all notes and information, to be presented as this is in the future.

Also I would like to officially thank all the computer nuts, and mentor friends, for without whom, this book and my knowledge would not exist. Time warp thanx to THinkradrite and Itragen.

Larry Simioni

BASIC MACHINE PROGRAMMING PRINCIPLES

If you have done any programming in BASIC, you will have very little difficulty understanding the program structuring and flow of machine language. For example, "GOTO", "CALL", "RETURN", and "&(n)=n", can in fact be implemented by using a single machine code instruction.

The main difference is, a machine code instruction or its associated data, is always 8 bits (at a time) wide. Think of this as the 8 wire data buss that runs from the Z-80 to its ROM, RAM, and I.O. devices.

The location of each 8 bit byte of information is determined by 16 bits of data called an address. Think of this as the 16 wire address buss. (Or more primitivly, the "Line number" of the 8 bit data) These 16 bits equate to 65536 possible 1 byte locations, which can hold a value from 0 to 255.

As soon as the power is turned on, the Z-80's PROGRAM COUNTER (P.C.) is reset to 0. The contents (position) of the P.C. then appears on the 16 bit address buss, and the first instruction is fetched (read) from memory location 0. (This is usually ROM memory) The byte at location 0 returns to the Z-80 on the 8 wire data buss.

So the address buss looks like this... 0000 0000 0000 0000.
Let's say the byte it gets back on the data buss looks like this... 0011 1110. This happens to be the instruction LD A,n which is somewhat analogous to "A=n" in BASIC. On this first byte the Z-80 has detected the type and length of the instruction, and now knows it must increment its P.C. to get the next byte. (The "n") Some instructions require the Z-80 to increment its P.C. 3 or 4 times to get the complete instruction and/or data. In this case the following byte is the "n" data which is to be loaded into the Z-80's A register.

Let's backtrack a second and analyze this procedure.
-1- At location 0 it picked up the byte 0011 1110, (LD A,n)
-2- The P.C. is then incremented to 0000 0000 0000 0001
-3- The data for "n" is then read from location 1. This data is any 8 bit number (0-255) and is determined on assembly of the program. (Of course it's un-alterable if loc. 1 is ROM.)
-4- It then executes this instruction (Loads A with n) which in this case takes 3 more clock cycles.

It will then increment its P.C. again and fetch its next instruction from location 2. It will continue fetching and executing instructions just as in BASIC, and will alter its P.C. on jump or call instructions. (GOTO&GOSUB)

Other pins on the Z-80 will be affected by certain instructions as they are executed. It should be remembered that data may go out on, or come in on the same 8 wire data buss. This direction will be determined by the type of instruction being executed.

Don't let words like "accumulator" or "flag" throw you. They are all the same thing, a register (memory) inside the Z-80. The word "accumulator" can be misleading, as it doesn't always "accumulate". The accumulator is the register in any microprocessor which is used the most. It is the object of most ADDS, LOGICALS, and I.O. instructions.

It is also used for temporary storage of data, holding the result of an operation, or holding the data byte to be operated on. The Flag register is not used as other 8 bit registers. It may be thought of rather as a collection of single bits which are usually "tested" by conditional instructions. ("IFs")

On the next page is a chart of all the Z-80 registers and Their "names", though they are often used for anything. Consider that everything it does, it does with these 26 bytes of memory. Sometimes there is considerable shuffling around being done, to and from the Z-80's registers. If the time comes when you write a program only to find that some of your instructions do not exist, you'll be surprised at how well you can dance.

OPCODES and MNEMONICS (Whatsit?)

Before we jump in the lake, there are a few rules that must be learned to read and understand Z-80 opcodes. Opcodes are expressed in MNEMONICS, such as LD A,n from the previous example. Of course there is a specific 1 byte number for the first half of this "partially english" expression , called the opcode.

Remember that the object, or destination, is always specified first. For instance... ADD HL,DE will add HL to DE with the result in HL. (DE unaffected) However according to one of Murphy's laws there must be an exception. An 8 bit ADD, such as ADD B , will add A to B with the result in A. (Each instruction type and its specific action will be covered)

When a memory location EXTERNAL to the Z-80 is specified, parenthesis are used around the memory location. For example; LD (nn),A will cause a memory "write" (store) to location nn from the accumulator. Another example is LD A,(HL). This will execute a memory "read" from the location specified by the contents of HL, to the accumulator.

All lower case letters found in mnemonics specify some form of variable data which is decided by the programmer on assembly. This may be either a register (r), a register pair (pp), or some form of variable numerical data. (This need not be memorized as all data or registers will be specified as the case may be.) Also, the numerical data may be expressed as "n" (8-bit), "nn" (16-bit), "d" (-128 to 127), or "e" (-126 to 129).

The only real important one to remember is the parenthesis. Say "memory location" , then read what's in the parenthesis. Remember that the value in the parenthesis is to be taken as an address.

Other garbage that will be found in mnemonic expressions are as follows;

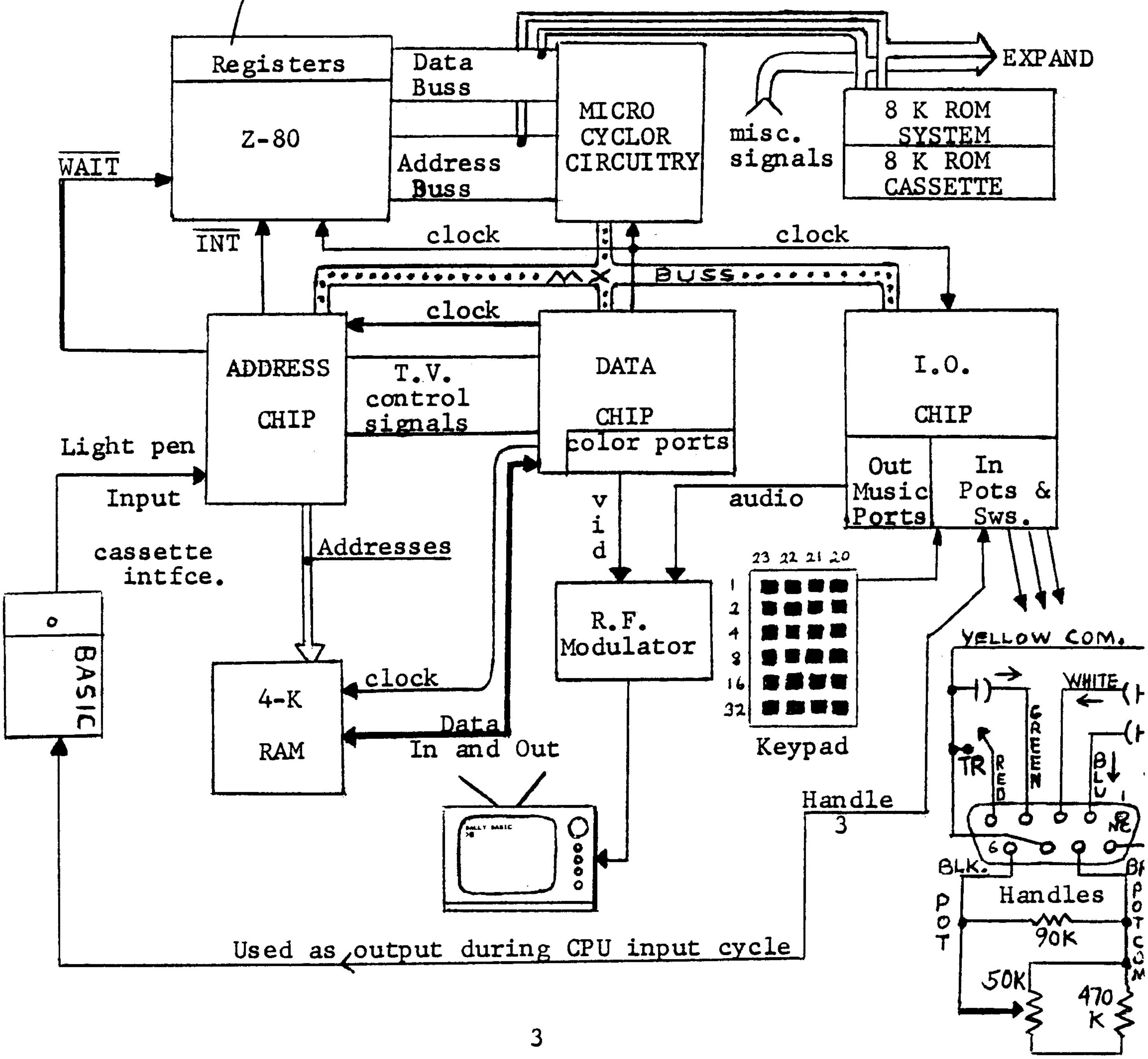
C carry Z zero NC no carry PE parity even
M minus P plus PO parity odd NZ non zero

"There are 74 generic opcodes (such as LD), 25 operand key words (such as A), and 694 legitimate combinations of opcodes and operands in the Z-80 instruction set."

Zilog Corp.

Z-80 INTERNAL
REGISTER CONFIGURATION

	STACK POINTER		4-16 bit registers (8 bytes)
	PROGRAM COUNTER		
	INDEX REGISTER		
	INDEX REGISTER		+
General Purpose Registers	A ccumulator	A'ccumulator	Alternate Register Set (16 bytes)
	F lag	F'lag	
	B yte	B'yte	
	C ounter	C'ounter	
	D estination	D'estination	
	E	E	+
	H igh	H'igh	
	L ow	L'ow	
	I nterrupt	R efresh	Special Purpose 2-bytes



BALLY SYSTEM OPERATION

As is the case with any computer system, it is necessary for the programmer to understand certain aspects of the hardware in order to properly utilize it. More specifically, all systems have different I.O. functions, with their ports (address locations) in different places. Also the video system and its associated ports are unique in the Bally system. Eventually you should understand the way the onboard software, and Bally Basic, handles various functions of the hardware, and incoming software. As was mentioned, other pins on the Z-80 will be affected by the execution of certain instructions. This "electronics knowledge" is necessary to understand *why* certain things happen during a program. "Input" pins on the Z-80 may also be activated by the hardware. (C.P.U. control functions)

The easiest to understand is the port configuration. All the ports which may be accessed by the Z-80 are shown on the next page. Notice that the various ports are located (physically) in different chips, and that some are for inputting data (with IN instructions), and some are for outputting data, (with OUT instructions).

Starting with the clock, all other system clocks are produced by the DATA chip. This poor little guy is really kept busy. His various duties are ;

-1- Control and pass all memory reads and writes.
-2- Produce all T.V. signals (To sync T.V. and other hardware)
-3- Decode and assemble video and color information.
-4- Multiplexing demultiplexing and controlling the system data buss.(MX-buss)
-5- Produce precise gated clocks for Z-80, memory, ADDRESS and I.O. chips

The ADDRESS chip produces and/or passes all addresses to RAM memory. On command from the data chip it "scans" the memory for the next line of video. During this process a "WAIT" signal is sent to the Z-80 which suspends its operation until the scan cycle is completed. Also a short WAIT signal is sent out for every memory read or write because of the multiplexed buss system. The ADDRESS chip also produces the INT (interrupt) signal in association with a particular scan line, or light pen data. This will be taken up in more detail in the section on interrupts. Basically this signal causes the Z-80 to jump to a location in the sotware for processing an interrupt routine dealing with the particular device that's causing the interrupt.

The I.O. chip is very straightforward, being simply a conglomeration of input ports to the Z-80 for the keypad and player handles. And the output ports for/and the music generator system.

The system ROM has many routines which are often accessed by other ROM cartridges and/or software. The most important function of any system ROM is the hardware initialization needed to get the system running. In this case it's simply a short section that initializes certain custom chip operating modes. All the alphanumeric display routines and characters are also located here. Other functions for which it is resposible are; Writing and moving game patterns. Doing certain math conversions for the custom chips. Displaying and operating timers. Producing random numbers. Organizing and playing music patterns. Also included in the system ROM are multiprecision math routines, other software organization and initialization procedures, and a routine to display and operate a menu node.

I.O. PORT CONFIGURATIONS

OUTPUT PORTS---DATA chip

	bits
0----Color register (right)	00
1----Color register	01
2----Color register	10
3----Color register	11
4----Color register (left)	00
5----Color register	01
6----Color register	10
7----Color register	11
8----Low/High resolution	
9----Horizontal boundary	
10---Vertical blank register	
11---Color block transfer	
12---Magic register	
25---Expand register	

INPUT PORT---DATA chip

8----Intercept feedback

OUTPUT PORTS---ADDRESS chip

13---Interrupt feedback (LSB)
14---Interrupt enable and mode
15---Interrupt line (scan line #)

OUTPUT PORTS---I.O. chip

16---Master oscillator
17---Tone A frequency
18---Tone B frequency
19---Tone C frequency
20---Vibrato
21---Tone C volume/Noise modulation
22---Tone A & B volume
23---Noise volume
24---Sound block transfer

INPUT PORTS---ADDRESS chip

14---Vertical line feedback
15---Horizontal address feedback

INPUT PORTS---I.O. chip

	values returned
16---Player handle #1	UP----1
17---Player handle #2	DOWN--2
18---Player handle #3	LEFT--4
19---Player handle #4	RIGHT-8
	TRIG.-16

	values returned
20---Keypad column 0 (right)	TOP---1
21---Keypad column 1	2
22---Keypad column 2	4
23---Keypad column 3 (left)	8
	16
	BOTTOM---32

28---Pot 0	counterclockwise 255
29---Pot 1	
30---Pot 2	clockwise 0
31---Pot 3	

SYSTEM MEMORY MAP(w/Basic)

0-8191	SYSTEM ROM
(0-4095)	MAGIC RAM
8192-16383	CASSETTE ROM
(8192-12287)	BASIC ROM
16384-20479	RAM (TOTAL)
(16384-20479)	SCREEN RAM
(16384-19983)	BASIC GRAPHICS
(20000-20463)	SCRATCHPAD
(20000-20001)	TAPE POINTER
(20002-20049)	TAPE BUFFER

(20078-20141)	variables
(20142-20161)	CALCULATOR INTERFACE AREA
(20180-20283)	LINE BUFFER
(20284-20462)	stack area
20480 To 65536	AVAILABLE For EXPANSION

BALLY BASIC MEMORY USAGE

The video (screen) memory is actually all the read/write memory (RAM) that the "bare bones" unit has to work with. This memory is shared with a program and/or sectioned off at the bottom to provide "scratchpad" space for the Z-80. The RAM memory addresses run from 16384 to 20479. Bally Basic divides this up in three ways. First, program storage and graphics are shared in the same screen memory addresses. Each byte is broken up into even and odd bits. Even bits are program, and odd bits are graphics. Then since the Z-80 also needs some working space, the last addresses available for program and graphics is at 19983. The addresses from 19984 thru 20479 are partiallly used for Z-80 scratchpad. This can be seen by setting &(10) to 208. Other cartridges may use more of this memory for its graphics.

There are 40 bytes of memory across each line on the T.V.. This is broken down into 160 seperate pixels, meaning there are a possible 2 bits per pixel (or fleck). With Bally basic there is only 1 bit per fleck, the other bit being used for program storage. With other cartridges there are two bits (all) used for each pixel, giving you a possibility of 4 combinations of bits in each fleck. They are... 00 01 10 or 11. These bit pairs are decoded by the DATA chip into four different colors depending on the value in each associated color port register. A horizontal boundary (port 9)can be used to divide the screen for 8 possible colors. This can be seen in Gunfight, the red gunfighter being the same bit combination as the blue one, but located to the right of where the horizontal boundary is set. This makes its color controlled by a different port.

With Bally Basic there are a different set of addresses to access even bits (program storage) only. These addresses run from -24576 to -22777 and is called the text area. All line numbers are stored as the actual binary number in two of these bytes. The first non-number character which is input, tells the software to store all following bytes as ASCII character code numbers. (see chart on next page)

When a program line, or a direct execute command statement is entered, it first goes into an area called the LINE BUFFER. (see map pg. 5) This is the area which we will be using to store and run most of the machine language programs. The only time it uses this area is when a line is being input (Or INPUT n). Care must be taken not to overwrite a machine program which resides in this area. Some space must be left at the beginning of the LINE BUFFER so you can at least enter CALL nnnn. (If the CALL nnnn is already contained in a previously entered Basic segment, you still need room for the RUN & GO) It is safe to run your program right into the stack space which directly follows the Line Buffer.

The only other areas which can be used to run machine code are the TAPE BUFFER and CALCULATOR INTERFACE areas. Screen memory which is not being shared with a Basic program may also be used, however measures must then be taken to prevent it from being destroyed by graphics or a scroll. The text area may not be used since this is software driven and intermingled with the graphics. The combined LINE BUFFER and STACK areas give us over 175 bytes to work with.

ASCII CHARACTER CODE NUMBERS
(TV Out, or KP In)

1-6 ?	45 - minus	63 ?	81 Q	99 ÷ (c) divide
7 Bell	46 .	64 @	82 R	100-103 (defg)
10 Line Feed	47 /	65 A	83 S	104 LIST (h)
13 "GO" (CR)	48 0	66 B	84 T	105 CLEAR (i)
31 ERASE	49 1	67 C	85 U	106 RUN (j)
32 SPACE	50 2	68 D	86 V	107 NEXT (k)
33 !	51 3	69 E	87 W	108 LINE (l)
34 "	52 4	70 F	88 X	109 IF (m)
35 #	53 5	71 G	89 Y	110 GOTO (n)
36 $	54 6	72 H	90 Z	111 GOSUB (o)
37 %	55 7	73 I	91 [	112 RETURN (p)
38 &	56 8	74 J	92 \	113 BOX (q)
39 '	57 9	75 K	93]	114 FOR (r)
40 (	58 :	76 L	94 ↑ (⌐)	115 INPUT (s)
41)	59 ;	77 M	95 ← (_)	116 PRINT (t)
42 *	60 <	78 N	96 ↓ (′)	117 STEP (u)
43 +	61 =	79 O	97 → (a)	118 RND (v)
44 ,	62 >	80 P	98 x (b) mult	119 TO (w)

All operations on the memory will be done using the "%" command. (peek or poke) Widespread mis-information dictates a whole slew of unnecessary conversions to be done to input machine code. Since the Bally listens in decimal, all you have to do is talk in decimal, one byte at a time, and it will understand perfectly well without all the fuss. If you're skeptical read on, and do the following experiment.

1. All Z-80 can be (and are) represented in decimal form. (0-255) See back of book. (Instruction Set)
2. Hex code is a hassle during hand assembly.

To demonstrate the ease of machine code entry in decimal, first enter the following command statement.

```
FOR A=20220TO 20236;PRINT A,;INPUT ""%(A);NEXT A
```

You have just made a simple machine language loader. Notice I have left plenty of room at the beginning of the line buffer for entry of other command statements without destroying the program which follows. This loader can be shortened for entry from a starting address of 20203 by eliminating PRINT A,; and "".

So enter the following decimal instructions one at a time, hitting "GO" after each value is entered. The next address to be written into will be displayed after each entry is made.

20220=60	20225=219	20230=16	20235=240
20221=211	20226=28	20231=252	20236=201
20222=4	20227=71	20232=254	
20223=32	20228=219	20233=1	
20224=251	20229=21	20234=32	

NOTE: If the above loader is used to enter an adrs. (Value over 255) enter %(A) for the following byte. **See pg. 11**

Now RUN the program by entering CALL20220

Adjust pot 0 (Player handle 1) until a stable pattern is obtained. The program may be halted via the HALT button. If you have external memory try loading this program there, as it provides a good visual display of the speed difference. (Due to no WAIT states when using external memory)

WHEN WHAT WHY and HOW - MATH

The following information is necessary to learn to allow the programmer to determine "when" to use "what" math, and to gain a general understanding of some of the unique numerical tricks which are used with all computers. Note that it is not completely necessary to understand all of the conversions to understand programming. However,to become proficient one should be able to grasp the "whys" of certain rules used. This first involves understanding the "hows" and imminent results. Do not become discouraged if you don't immediately see the use for something, or don't fully understand it. As you begin to do the experiments these things will start to become apparant. At this point the most important thing for you to do is START TAKING NOTES On the way you see things, and any other ideas you may get.

EXPERIMENTS - PROCEDURES - RESULTS - IDEAS - EXPLANATIONS
(Observations)

BASIC NOTATIONS YOU SHOULD MEMORIZE

As you know, all numbers are presented to the circuitry in binary form. This means that if a wire in the circuit has a voltage on it (a "high") a "1" will be represented in one of the 8 (or 16) columns. Depending which column it is, that "1" will have a certain weighted value. For instance, the "1" in the decimal number 128, has a weight of 100. In binary this same "1" (same column) has a weight of 4. Since you can only have a 0 or 1, after counting "0"-"1"... you must move to the next column. The chart below shows all the weighted values for each bit position in binary numbers.

BINARY DATA MAPPING DIAGRAM (WEIGHTING or EQUIVALENCY)

BIT POSITIONS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VALUE OF BIT	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
DATA BUS (wires)	X	X	X Non	Existant X	X	X	X	X	D-7	D-6	D-5	D-4	D-3	D-2	D-1	D-0
ADDRESS BUS (wires)	A-15	A-14	A-13	A-12	A-11	A-10	A-9	A-8	A-7	A-6	A-5	A-4	A-3	A-2	A-1	A-0
	NIBBLE 4				NIBBLE 3				NIBBLE 2				NIBBLE 1			
	BYTE 2								BYTE 1							

Thus the decimal number 128 would be represented in binary like so; 1000 0000

To read a binary number simply add up all the weighted values for each column in which there is a "1".

Since binary notation takes up a lot of room on paper, making it somewhat difficult to work with, another system has been devised called HEXADECIMAL. (Decimal (10) + Hex (6) = 16). One of the requirements of this system was that it should neatly take into account an even, and not too large number of bits from the binary system in each of its columns. This is so one can easily visualize the binary form of the number. (Binary notation is sometimes used to represent other entities besides numbers)

Each column in Hex has a limit of 16 seperate digits. This means you count from 0 to 16 before moving to the next column. (see fig. 1) The numbers from 10 to 15 are represented by the letters A thru F to fill the other requirement. (One neat character per column). Each half byte is called a nibble, and can thus be represented by a single Hex digit.

Fig. 1

Decimal	Binary	Hex	Decimal	Binary	Hex	Decimal	Binary	Hex
0	0000	00	6	0110	06	11	1011	0B
1	0001	01	7	0111	07	12	1100	0C
2	0010	02	8	1000	08	13	1101	0D
3	0011	03	9	1001	09	14	1110	0E
4	0100	04	10	1010	0A	15	1111	0F
5	0101	05						

TWO's COMPLEMENTS

"Two's Complements" representation was created as a means to facilitate binary addition and subtraction of positive and negative numbers. It is also the method used internally in the Z-80 to perform certain types of jump instructions. This is because Two's complements math is also much easier for the electronics to accomplish in a negative jump. (A backwards jump in a program).

What are "Two's Complements"? "Two's"...because the base, or radix, is Two (2). And "Complements"...because the plus and minus versions of the same number complement each other. That is when they are added together they will add to zero. There are always a given number of maximum bits used, because what we will call adding to zero, will actually be "overflow".

Note that given a certain number of bits, one(1) more than half of the possible numbers obtainable must be labeled negative numbers. Why one more than half? Because there are always an even number of possibilities, but "0" (zero) can not be included. This may sound complicated and ridiculous, but things actually work out quite well, as you will soon see.

Look at the table in fig.2. Notice that given 4 bits the plus(+) and minus(-) numbers from -8 to 7 can be represented. Given 8 bits the range is -128 to 127, and with 16 bits the range is -32768 to 32767. The Bally only "seems" to go only to -32767 because of an overflow condition in the math handler in BASIC.

Fig. 2

Dec. Two's Comp.		Dec. Two's Comp.
7....0111		-1...1111
6....0110	7 0111	-2...1110
5....0101	+ -8 1000	-3...1101
4....0100	= -1 1111	-4...1100
3....0011		-5...1011
2....0010		-6...1010
1....0001		-7...1001
0....0000		-8...1000

What observations can be made about these strange numbers?

A. First notice that all negative numbers start with a "1". This is important to remember. Negative numbers always have a "1" in their most significant digit. (Bit 3 in this case)

B. All the positive two"s complement numbers are the same as their binary code. And if you turn all of its zeros to ones, and all of its ones to zeros, it will always be the negative version of that number minus 1.

Notice that if you add 7 and -8, you get -1. Now let's add +7 and -7.

```
 7   0111
-7   1001
=   10000
```

This is the overflow condition which was mentioned, and the reason we must specify the number of bits. This bottom number, which would normally be 16, is actually zero because we are only using 4 bits.

Now let's see what happens when we take a negative two's complement number and do a bit reversal on it. Take the number -3. It is 1101 meaning we would get 0010. This is the positive number 2 (one less in absolute value). So you can see from this, that to do a conversion from one to the other (pos.&neg.) you would first reverse bits, then ADD 1. Take the positive number 7 again. Notice that if all the bits are reversed and one is added to it, you end up with its two's complement -7.

One last observation is that the largest positive number is always a zero in the most significant bit, with ones filling in the remaining given number of bits.

CONVERSIONS

There is one anomaly of the Bally Basic "%" command which must now be discussed. When the % command is used to write a byte into memory, 2 bytes are always written into, even though you can advance the address one byte at a time.

For example, if the number 255 is poked into the memory, the location it was put into looks like this; 1111 1111 0000 0000. The following byte is automatically made zero. However, when it is read, it is read with the second byte coming first. The reason for this, (and there is a good one), is to handle the Z-80's 16 bit math instructions using two's complements. This will become clearer when you get to the section on 16 bit load instructions. Let it suffice for now to say that the two bytes are swapped on a read.

This means that to get a combined two byte decimal number in the right places (byte for byte), on a write, they must first be swapped before converting to get that number.(One 16 bit two's complement decimal number) It also means that any time you read a "%" position, you are reading the combined two's complement value of the two bytes.

To demonstrate this better try the following program.

```
10 INPUT %(20200),%(20201)
20 PRINT %(20200)
30 GOTO 10
```

Now RUN and enter 255 and 0. You see that you get 255 which makes sense.

Now enter 255 and 1. Notice that the answer you get looks like this; 0000 0001 1111 1111. (read with "1" first)

Let's try to create a negative number now by making the most significant bit a "1". Enter 0 and 255. Notice that this is the two's complement of 256. 1111 1111 0000 0000 Or what is actually there, 0000 0000 1111 1111. (add 1 and sign negative for two's comp.)

Now we will get into some conversions. First let's go from hex to decimal. To do this simply multiply the decimal value of each hex digit by its weighted hex column value. Then add up all the results from each column. (see Fig. 3)

Fig. 3

Nibble 4	Nibble 3	Nibble 2	Nibble 1
Column 4	Column 3	Column 2	Column 1
HEX "1"=	HEX "1"=	HEX "1"=	HEX "1"=
4096	256	16	1

Take the byte FB for instance. This is simply F(15) x 16 + B(11) = 251. Enter this number into the preceeding program. Now convert the byte C9 to decimal and enter your answer as the second value. You should get the decimal number -13829.

Now convert the whole two byte value FBC9 into one 16 bit value by hand. To do this remember we must first swap the two bytes so it becomes C9FB. Do this on a calculator as follows;

1. Cx4096=49152 and hit M+ (or write it down)
2. 9x256=2304 & M+
3. Fx16=240 & M+
4. Bx1=11 & M+
5. RCM=51707 (or add the results)
6. Since this value is over 32767, subtract 65536 to get the two's complement value, and you end up with the same -13829.

So if you must go from hex to decimal, do the conversions one byte at a time, and let the machine do all the dirty work. Here is a short program that will do this by simply entering the decimal values of each hex digit in their normal order. (Enter 15(F),11(B),12(C), 9)

```
10 INPUT A,B,C,D
20 %(20200)=Ax16+B;%(20201)=Cx16+D
30 PRINT %(20200);GOTO 10
```

Going from decimal to hèx is just the opposite. Don't forget if it is a value you read by the "%" command, your answer is going to come out with the bytes reversed. Also before doing this conversion add 65536 if the decimal number is negative.

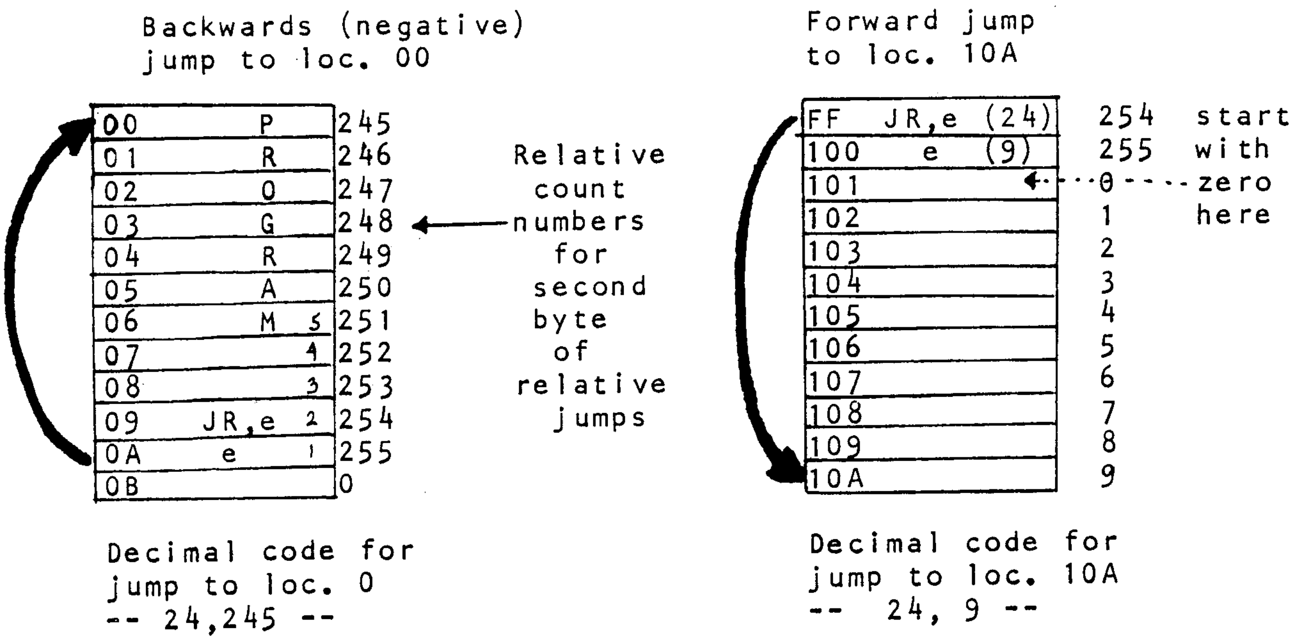
So first we take the number -13829 and add 65536 giving us 51707. Now store this in the calculator memory. Then divide it by 4096. Notice you get 12 which is "C". Now multiply 12 x 4096 to get 49152 and hit the M- button. (First column solved and subtracted from unknown) Now hit RCM and divide that remainder by 256. (Next column) Repeat this process for each column value until you have no remainder, writing down the answers as their equvalent hex digit. Come now! You don't really think Hex is a curse do you?

Now for two's complements. With this we only have to concern ourselves with a 1 byte value, since this is important for figuring out negative jumps using what is called a "Relative" jump. With this type of jump, a number is specified in the second byte of the instruction , and is called the displacement byte. This number can be from -128 to 127 (using two's complements) Note that with this type of jump the actual address (destination) of the jump is not specified. Rather the number of bytes relative to the jump instruction location + and - is used.

This means that a segment of code using relative jumps can be relocated to another spot in memory without worrying about changing addresses. (If the section it is jumping to follows with it, nothing being squeezed in between.)

First look at the charts in fig. 4. You will see that the first memory location FOLLOWING the instruction, is the starting point (0) for the relative count to its destination.

Fig. 4



Let's say we want to jump back to 3 bytes BEFORE the relative jump instruction. This would be the two's complement (In 8 bits) of the number 5. The easiest way to find this number is to simply count backwards, starting with 0. (The first byte following the jump instr.) See! I told you it was easy. Don't take my word for it though. Try doing a few manual conversions yourself to test this out.

The two's complement value of -5 would be 251. This would then be the second byte of a relative jump instruction to get to the location 3 bytes before the jump.

Now what if the jump is a little too far to be counting backwards all that way? First, to find the relative distance of the jump we must subtract. First take the address of the NEXT memory location AFTER the displacement byte, pos.0, then subtract the address of the destination. Sign this negative, and that's your relative distance. To find the two's complement number to use in the second byte , simply subtract that number from 256. Hard wasn't it?

OPCODE ACTIONS
Applications and Experiments

LOADS

First let me mention that there is no need to memorize all the different forms of addressing done by the Z-80. Whatever form it takes, will be done automatically depending on the instructions you give it. Just remember that there are different actions on the hardwre with the varying instructions.

For instance LD A,D does nothing with the hardware EXTERNAL to the Z-80. It simply loads its A register with whatever is in its D register. (All internal) Whereas the instruction LD (nn),A does a WRITE from whatever is in A, to the memory location specified by nn. Another form of addressing is given by the instruction LD A,(nn) , which will do a READ from memory location nn to the accumulator.

Loads never destroy the contents of the register or memory location being loaded FROM. Only the register or memory location being loaded TO is changed. Loads may be used for initialization, inputting data from memory to be operated on, storing results or other data, or any other actions you may have used the BASIC counterparts for.

Note that the form (nn) specifies a 16 bit address to a location containing 1 byte of data in most load instructions. The only instructions in which 16 bits of memory are operated on are those involving PAIRS of registers. One example is the instruction LD HL,(nn). In this instruction the byte specified by (nn) is loaded into L, and the byte at (nn+1) is loaded into the H register.

As is the case with most microprocessors, ALL 16 BIT DATA SPECIFIED IN ANY INSTRUCTION IS ASSEMBLED WITH THE BYTES IN REVERSE ORDER. For example, let's assemble an instruction to load the accumulator with the byte at memorylocation 20200. We will use the instruction LD A,(nn).

1. Find the MSB and LSB of 20200 (To be inserted in reverse order for (nn). To do this we divide 20200 by 256. We get 78 (MSB) , with a remainder of 232 (LSB).
2. Assemble the series of numbers (3 bytes) starting with the opcode for LD A,(nn) which is 58.

So the series of numbers to assemble this instruction are as follows;

```
 58  opcode
232  LSB
 78  MSB
```

Don't let this scare you though,as it is only given as an example for the actual format that the Z-80 recognizes all 16 bit data in. The instruction may actually be assembled like this;

```
   58  opcode
20200  address
```

After the values 58 and 20200 are poked into memory, we must skip over the byte that the 20200 overflowed into (See section on conversions) for entering following bytes in a program. This is so we don't destroy the MSB of 20200.

Test this out now,to see that the MSB will be in the right place. Load location 20200 with the value 20200.

```
        %(20200)=20200
Now     PRINT %(20201)     (pretend this 20200 is being used
                            as an address in part of a 16
                            bit instruction assembly)
```

Note that the MSB (78) has been automatically placed in its proper position for the assembly of a 16 bit instruction. Assuming the opcode for LD A,(nn) was at location 20199, and its address is at 20200, the next byte of the pretend program would be at 20202. Not to worry!!

Now say we are reading 16 bits of data with an instruction such as LD HL,(nn). The reverse rule also applies here. The address specified by (nn) will take the byte at that location and load it into L. Then the byte at location (nn+1) will be addressed and loaded into H. If you stored two bytes of data at location 20200(01) with an instruction such as LD (nn), HL, and you then wanted to access only the MSB, it would be at location 20201. In this instruction the contents of L will be loaded to location (nn), and the contents of H will be loaded to location (nn+1). KNOW where you're bytes WENT !?

There is no real reason why 16 bit data is handled in this reverse fashion. It's arbitrary choice by the manufacturer and has become standard protocall. There are microprocessors that handle 16 bit data with the MSB first.

OK! Now for an experiment. Load up the Machine Programming Utility. (First program on side 2) The load is completed when the screen turns yellow and it autostarts.

Here is a handy memory map of the variables, of which we will use A and B as an object of our Load experiments.

Var.Loc.	Var.Loc.	Var.Loc.	Var.Loc.
A...20078	I...20094	Q...20110	Y...20126
B...20080	J...20096	R...20112	Z...20128 20052
C...20082	K...20098	S...20114	BC...20130 20054
D...20084	L...20100	T...20116	FC...20132 20056
E...20086	M...20102	U...20118	NT...20134 20058
F...20088	N...20104	V...20120	CX...20136 20060
G...20090	O...20106	W...20122	CY...20138 20062
H...20092	P...20108	X...20124	XY...20140 20064
			RM...20142 20066

Press the "x" (multiply) to start you in the machine code entry routine. Enter 20200 as your starting address.

We will first try a simple load to the variable "A" using the instruction LD (nn),A. Whatever happens to be in the accumulator will be loaded to location (nn), which will be 20078 (A).

The decimal value for the instruction LD (nn),A is 50. Enter this number and hit "GO". Now enter the address for variable "A". Notice the utility automatically advances the address for a 16 bit entry. Now to end the program and get back to BASIC processing we must enter a RETURN (RET). Its action will be discussed later. The decimal value for the instruction RET is 201. Enter this and "GO". Now enter -1 to return to the menu. Hit the + and enter 20200 to run the program.

Halt the utility and PRINT A. There it is!!

Since this is only a one byte load, any value over 256 which is loaded into variable A ahead of time, (BY BASIC Command), will have its MSB added to the accumulator. To test this out try A=256 then CALL 20200 and PRINT A.

Now we will make this a little more versitile. Enter the following BASIC segment first.

```
500 INPUT "ADDRS. "A           "A" holds object address.
510 PRINT "-WR. +RD.           Select READ or WRITE function.
520 GOTO 520+&(20)             Keypad director to proper routine.
524 INPUT "WR. DATA ",B        If &(20)=4 LD. B w/WR. Data.
526 CALL 20200                 then write data. (Fall thru)
528 CALL 20207                 If &(20)=8 just read to "B".
530 PRINT B;GOTO500            Display data/Loop.
```

Now RUN the utility and enter the following machine code starting at address 20200. (Hit multiply then enter address)

```
20200 42    LD HL,(nn)  LD HL w/contents of var. A
20201 20078  the(nn)    (var.A) Object add. of 1 byte Write.
20203 58    LD A,(nn)   Load Acc. w/data to be written.
20204 20080  the(nn)    (Var.B) Holds data to be written.
20206 119   LD(HL),A    WRITE Acc. to add. (HL).
                               fall thru read
20207 42    LD HL,(nn)  LD HL w/contents of variable A.
20208 20078  ("A")      Object address of 1 byte READ.
20210 126   LD A,(HL)   READ loc.(HL) to acc.
20211 50    LD (nn),A   Load data red to VAR. B.
20212 20080  ("B")      OUTPUT address var. B.
20214 201   RET         RETURN from CALL to BASIC.
```

What we have just made is a single byte read or write machine routine. Unlike the "%" command, this routine may be useful for reading a single byte (Without picking up the following one) or writing into a single byte (Without destroying the following one).

Halt the utility and GOTO 500. To test it out try entering 20200 for your address, then hit the "+" to indicate a reed. Notice that you just get the single byte (42) as you entered it for the instruction LD HL,(nn). When this reed is done, the program is called at 20207.

When a write is done, the program is called at 20200 and simply falls thru the reed routine. It returns with the data from the memory location just written into. This data verification can then be red from variable B. At the start of the write routine, "B" holds the data to be written, and "A" holds the address to be written into.

To test out the write section first halt the program and enter an arbitrary value into the "E" variable. (Use a number over 255) Now GOTO 500 and enter 20086 for your address. (This is the LSB of the variable E) Hit the minus to indicate a write, and enter 10 for the data byte to write. Halt the program and PRINT E. The value you first entered for "E" is now the combination of the 10, and the MSB of the original. The write was done without destroying the following (MSB) byte.

It may be desirable to seperate the reed and write routines from each other. This may be necessary to speed up the process, or seperate and/or choose different BASIC variables. To do this put a RET at the end of the write section. Choose the addresses of your variables as desired. Of course you now have to re-enter the reed section (20207 now has RET).

As you can see, one of the disadvantages of machine language assembly without an assembler, is programs may have to be completely re-entered, changing jumps etc. just to squeeze in a single forgotten byte. However the advantages of speed and versatility make it well worth the effort. If you use your imagination, I'm sure you can come up with a simple BASIC statement to move all your bytes down to squeeze in one. If your careful to keep track of what your doing, and use the relative jumps whenever possible, this will work out quite well. Vigilant proof listing, and pecil and paper, will always prevail!!

THE STACK

A stack is a section of memory which is set aside to temporarily store the Z-80 registers while it uses the same ones for something else. The starting location of the stack (bottom) is set up with the instruction LD SP,nn. This is done by BASIC on reset, and sets up to start from 20462. To save registers in the stack PUSH instructions are used. The SP, or stack pointer, holds the current top of the stack address. This address points to the last byte PUSHED into the stack. After initialization of the SP, it then takes care of itself, pointing to the last byte PUSHED onto the stack.

Pushes work with register pairs. For instance, the instruction PUSH HL will do the following. First the SP is decremented to point to the next LOWEST address. The H register is then loaded into that location. The SP is decremented again, and the L register is loaded into that address. To restore these registers in the Z-80 again (After their use) POP instructions are used. For example POP HL first loads the L register with the byte at location SP. Then SP is automatically incremented and the H register is loaded with the contents of that address. Then the SP increments one more time to point to the next top of the stack byte.

CALL instructions also make use of the stack. The current PC is automatically PUSHED after incrementing, and is subsequently restored when a RET is encountered.

The stack is used extensivly to save addresses when going to subroutines, to preserve register values, or to just provide more working space. Whenever a CALL is made from BASIC, it is a good idea to PUSH all the registers you are about to use. You can never tell if those registers currently hold pertinent information, and could cause a crash if they are returned to BASIC with different values. This is especially true of the register pair DE, which holds the current Line number for BASIC. It should always be pushed if its use is required.

Whenever a number of PUSHES are done, and a program segment uses them, the POPS must be done in the reverse order that they were PUSHED. This will properly restore the right registers with the rigt values. Here is a hypothetical example. PUSH HL-PUSH BC-PUSH DE--(program segment)--POP DE-POP BC-POP HL (And usually RET) Using a different order for POPS can facilitate loading 16 bits from one register pair to another. Say you want to load the 16 bit contents of DE into HL. These two instructions will do the trick; PUSH DE POP HL The contents of DE is not destroyed, the stack pointer ends up in the same place it was, and HL will have the same 16 bit value found in DE. Obviously the contents previously contained in HL is lost in the process.

ALTERNATE REGISTER SET

The alternate register set may also be used when you're in a bind for register space. EXchange instructions are most often used to access the alternate register set. The most powerful and widely used EXchange is EXX. This instruction will exchange the contents of the current set of registers BC,DE,&HL all at once,and in a very short period of time. After execution of this instruction all opcodes following will be executed using the alternate registers for these three register pairs. As with PUSHES and POPS always EXX again to restore the Z-80 to its original state after using the alternate registers. As for its use with BASIC, you must always PUSH the registers after the EXX because BASIC is also activly using the alternate set.

For example; Prog.---EXX PUSH HL PUSH DE PUSH BC--(segment using alternate registers)--POP BC POP DE POP HL EXX

Exchanges need not be done to simply reed the contents of one of the alternate registers. Load r,r' will do a single register load from the other set if the same register is specified. LD A,A will load the current accumulator with the value in the other one.

IN s and OUT s

Input and output instructions are associated with any I.O. ports which the Z-80 communicates "through". A port may be thought of as a specific 1 byte code number which enables data transfer to or from an external device. (Or register in a device)

When the Z-80 recieves an input or output instruction, its' IORQ pin becomes active, thus signaling the external unit, and controling data flow on the data bus. Since an I.O.address, (To a port), is only 8 bits, there are a possible 256 ports (or devices) which can be activated.

1. The Z-80 places the device code (port #) on the lower 8 bits of the address bus. (The contents of the accumulator is also placed on the upper 8 bits at this time)

2. If the instruction is an INput the RD (reed) pin is activated. If it is an OUTput the WR (write) pin is activated.

3. The external device then writes the data supplied on the data bus to a specific register associated with the port #. (See I.O. port chart pg. 5) This is an OUT or WRITE instruction from the Z-80.

4. Or, the external device responds with a data byte from an INput port. (A reed or IN instr.)

First let's assemble an output instruction. The instruction OUT(n),A (2 bytes) means OUTPUT to port (n), FROM A. The opcode comes first (211) and signals the Z-80 that it is to do this output. Then it picks up the following byte (n), and uses the supplied number as a port address. The"A" means the data supplied on the data bus (to be output) will come from the present contents of the accumulator.

To output the value 133 to port 0, we have the following instructions;

1. LD A,n.....62,133......Load acc. w/133 (or acc. may already have the number to be output.)
2. OUT (n),A..211,0.......Do the output to port 0.

See if you can enter and CALL this program. Hint: Set &(9)=0, and put an RET at the end. (Opcode 201)

Input instructions operate in a similar manner. The instruction IN A,(n) Will cause the value contained at port (n) to be red to the accumulator. As an example, let's reed the handle 1 knob value.

1. IN A,(n)......219.....Opcode
2. (n)...........28......Port for POT0 (Handle 1)

A value from 0 to 255 will be returned to A, depending on the position of the knob. Try a 16 bit load to a memory position that you can read when calling this. (Must be in machine portion before calling) More examples of how these instructions can be used will be found in the program descriptions.

CONDITIONALS -"IF s"

Conditional statements are implemented by instructions which will do different things depending on the status of certain bits in the F, or FLAG register. (See below)

The Six Flags in Register "F"

SIGN "S"	ZERO "Z"	NOT USED	HALF CARRY	NOT USED	PARITY OVERFLOW	"N" (SUBTRACT)	CARRY "C"

The most important and widely used of these are the CARRY and ZERO bits (Flags). Let's first take a look at the conditions on which they are set or reset.

The CARRY is used to indicate a carry or a borrow after the execution of ADD or SUB instructions. If the result of an ADD instruction is over 255, (Carry from bit 7), or less than 0, (Borrow from bit 7), the carry flag will be set (Logic 1). It will also be set by shift and compare instructions, depending on the results of these instructions.

The ZERO flag is used to indicate that the result of an operation was 0. Many instructions will affect the state of this flag. Remember, if the result was ZERO, the zero flag will be set to logic 1 (set, or on).

The sign flag indicates the two's complement sign of a number. It is always set the same as the most significant bit of the result of an operation. (Negative number-sign flag=1)

Parity/overflow (P/V) is used for two purposes. If the parity of an operation is even, (Even number of bits), then it will be set to logic 1. If the parity is odd it will be reset. The number 0000 0011 has even parity, and the number 0000 0001 has odd parity. Also the flag will be set if the result of the addition of two two's complement positive numbers, result in a negative number.

Half Carry and Subtract flags are used in conjunction with the instructions used for BCD arithmetic, and will not be used with conditional instructions. (Conditional instructions do not "test" these two flags.)

Now lets take a look at what the COMPARE (CP) instruction does. The compare instruction always compares a given value or register with the contents of the accumulator. The number being compared with the accumulator is actually subtracted from the accumulator. If the compare is equal (Numbers the same value) the result will be zero. This will set the ZERO flag. (1) Niether the contents of the acc. or the register you are comparing are destroyed in the process. If the number is greater than the acc., (borrow from bit 7), then the CARRY flag is set. If the number is less than the acc. the CARRY flag will be reset.

This is a very powerful conditional instruction when used in conjunction with conditional JUMPS or CALLs. It gives you the functions of "IF greater than","IF less than"," and IF = to, do". It does this all in one shot depending how it is assembled in the program.

Here is a hypothetical IF statement assembly.

"IF the Accumulator is greater than n GOTO nnnn"

LD A,n	Get the value to accumulator. (Or it may already be there.)
CP n	Compare with a number by subtration from A.
JRC,e	Jump relative if carry bit high. The carry bit will be high (set) if there is a borrow from bit 7. (A still less than CP number.)

Program may normally fall thru here (A greater than CP number) and may be diverted with another jump instruction. Or the processing to take place on A>n may directly follow the JRC.

You can see this assembly can also say "IF A>n fall thru." If the instruction JRZ or JPZ were used, it would then jump only on an equal compare. Dont forget, the number you are comparing is subtracted from the A reg. (Keep an eye on your'e flag conditions) Look through the Z-80 instruction set and find all the CALL and JUMP instructions that may be used as conditionals.

Refer to the program you entered on pg.7. This should give you a good idea how conditional statements were implemented to detect the HALT button to stop the program.

60	INC the acc. by 1 (RAINBOW)
211,4	OUTput the value to port 4
32,251	Jump Relative to the "60" IF Acc. Not Zero yet.
219,28	INput the value in port 28 (Pot 0) (TIMING)
71	LD B,A store the value in B
219,21	INput the value of port 21 (Buttons in HALT colm.)
16,252	DJNZ Means Decrement the B register and jump IF it is not zero yet (Wastes time here looping -
(HALT)	black space - Time depends on value of pot 0)
254,1	CP n Compare the Acc. with a 1, which would be the value in it if the HALT button was depressed
32,240	Jump relative if not equal. (Go back to beginning if HALT was not depressed.-A prob.0 too)
201	RETurn to BASIC (If it fell thru)

INCREMENTS and DECREMENTS

INC's and DEC's simply add 1 or subtract 1 from the register in the instruction. There are 16 bit and 8 bit versions, so be sure you use the right one. INC HL is a 16 bit INC, and INC A is an 8 bit. INC(HL) will add 1 to the byte @(HL).

ADDs and SUBs

ADDs and SUBs are also either 8 bit or 16 bit. ADD r will always add TO the Accumulator with the result ending up in the Accumulator. ADD A,n will add the number n to the A reg. SUB(HL) will subtract the byte @loc.(HL) from the Acc. ADC instructions will add to the acc. and if the carry bit was set, will add 1 to the result. (ADD toA+ Carry bit)

LOGICALS

The variety and uses of LOGICALs is too great to discuss in any detail in this book. 25 pages will be dedicated to them in a following course. It is important however to basically understand what they do.

The three functions are AND, OR, and XOR. Each bit of the register or byte in the instruction is compared with the corresponding bit in the Acc. A logical AND, OR, or XOR is then done on each bit, and the result is in the Acc.

Notice that if you XOR A with itself the Acc. will be ZEROed. (Either bit but not both = XOR)

It is important to keep an eye on the flags (How they are affected) to make good use of LOGICALs. The CARRY flag is always reset, and the P/V SIGN and ZERO flags will be set depending on the results of the operation. For example the AND A instruction will leave the SIGN flag set if the Acc. holds a negative number. Notice that ANDing A with itself will leave you with the same number. (BUT MAY AFFECT FLAGS)

Another use of the AND is a technique called MASKING. In this process the number you are ANDing with may be just the lower four bits for example. (0000 1111) (15) Your'e answer will then be just the bits that are high in the lower four bits of you're operand. It may be used to determine if a given bit is on or off for monitoring an external device, or to create bit patterns for pixel mapped colors.

OTHERS

Other Z-80 instructions include bit shifters,test,set and resets, blok load,search,and outputs, and other miscellaneous instructions such as NEG,HALT,CPL,and DAA. Eating too much of these for breakfast can be hazardous to you're health.

One delectable treat is the block load instruction LDIR.

It will load a complete block of data of up to 65536 bytes when it is executed. This may be handy for say copying a cartridge to memory. HL is initialized to the start address of the block of data to be copied. BC is set to the number of bytes to copy, and DE is set to the starting address of the destination. LDIR will then load the byte @(HL) TO (DE), increment both HL and DE, and decrement BC. If BC is not zero the instruction repeats itself by decrementing the PC twice. (LoaD-Increment-Repeat if BC#0)

Here is a copy cartridge (or other data block) program. If you have the Blue Ram it is handy to load this starting @ loc. 28672. (20220 or 20002 will do just as well)

```
243            Disable interrupts while doing this please.
245            PUSH AF
197            PUSH BC
213            PUSH DE              Save environment
229            PUSH HL
237,75,20080      4 byte opcode-Load BC w/variable "B" (Byte Counter)
42,20078          (3)  LD HL from variable "A" (start address-source)
237,91,20082      (4)  LD DE from var. "C" (Destination address)
237,176           LDIR
219,21         Input Halt column
254,1          CP 1-If so,
40,8           JR Z goto done (count 8-next pos. 0)
219,23         Input "GO" column
254,1          CP 1-If "GO",
40,231         JR Z Go set parameters and do LDIR
24,242         JR e Loop on checking buttons
225            POP HL
209            POP DE              (DONE)
193            POP BC
241            POP AF
251            EI
201            RETurn to BASIC
```

To use this program first load the BASIC variables as follows;
A= Source address of data block to move. 8192
B= Number of bytes to move 2048 or 4096 } CARTRIDGES
C= Destination address (Move to?) 24576 (?)

Then CALL the address you loaded this program at, remove the BASIC cartridge, and insert the cartridge to be copied. Hit the GO button (Only takes a few milliseconds), then remove the cartridge. You may now re-insert BALLY BASIC and hit the HALT button. The program will (may?) return to you with any BASIC program that was there still intact. (Useful for decomposing)

This is a handy utility for moving things to addresses BASIC won't reach. (Such as -30000) You may also change the source of the variables for source,number ofbytes,and destination. DR.WHEN needs metamucil while his socks are drying. Excellence interrupted and swallowed the universe. Tonight Tim stops a kalatan ritual, and!..

INTERRUPT PROCESSING

As you have probably noticed, the BASIC language may operate independently from the routine that displays the colors,in the programs on tape. Other processes that may be done with an interrupt routine are; keyboard input, timers, music, and a variety of others.

An interrupt may be described as any break in the main program occurring at random (To the the mainline) intervals. During this break another routine is entered to execute or update a specific interrupt process. The interrupt may occur at any time during the main program, and is always initiated electrnically by some external device.

Assembly of an interrupt routine will be explained shortly. First you must learn a few important facts about the actions that take place during an interrupt.

There are three different modes (Or series of events) that the Z-80 can initiate after being interupted. These are IM0, IM1, and IM2. BASIC initializes IM2 in the Z-80 on reset. The Address Chip, (The interrupting device), is also designed to operate using the IM2 actions of the Z-80.

The actions of IM2 will occur when the interrupting device outputs a signal to the Z-80 called INT. The output may be generated due to a variety of different situations occuring in the external device. In the case of the address chip, the output may occur on a pre-programmed scan line, (screen INT), or a light pen hit (L.P. INT). Light pen interrupts deal with each seperate pixel timing (Faster) and are given priority over the slower scan line (screen) interrupts.

But getting back to simplicity, let's see what happens when INT line is activated in IM2.

First the Z-80 will complete the present instruction it is working on. The Z-80 will then expect to find an 8 bit number which is sent to it by the interrupting device. (This 8-bit number may be pre-programmed into the interrupting device through an I.O. port) This number is used as part of an address to VECTOR (send) the Z-80 to the particular routine to service the needs of the interrupting device.

Now for the complete step by step process which I hope will unconfuse you if you are.

1. The Z-80 is buzzing along minding its own business, doing some BASIC program perhaps.
2. All of a sudden it gets to the end of some instructiun and %#!*!*? (Detection of signal active (low) on $\overline{\text{INT}}$ line.)
3. So the Z-80 being in IM2, INPUTS an 8-bit number which has been sitting there from the Address Chip. (LSB)

4. The Z-80 then combines this number with the number in its I register to get a 16 bit address called the interrupt vector.
5. At this address the Z-80 expects to find another address which is then the location of the routine to service the interrupting device. This process of an address being found at an address is called VECTORING. This is a very cute way of forming indexes or tables of addresses which could direct the Z-80 to many different routines for many different devices.
6. When the Z-80 enters the interrupt routine all the registers are PUSHED or EXXed so the status of events in the mainline are not disturbed. The registers are restored on INT routine EXIT.
7. During the interrupt routine many different actions may take place to update the interupt processes. The external device may recieve information which tells it when it is to generate another interrupt, the heart of the interrupt process is executed, and certain FLAGS may be set or reset to control program flow.
8. At the end of the interrupt routine is a simple RET instruction (After the POPs) and the mainline program is picked up exactly where it left off.

Now let's examine all the software requirements and setup of an interrupt routine

1. The ADDRESS Chips operating modes are first set upas follows
   a. Interrupt mode for screen, light pen, or both are set up through port 14
      1. This is set up for both on reset.
   b. The 8-bit number which will be the LSB of an *address is loaded through port 13 (INT feedback)
      1.This must be done by user to provide starting address of their routine.(@this adrs.)*
      2.BASIC INT vector 2062H (To 20B0H)
   c. A scan line # is loaded through port 15 to tell the address chip which line to interrupt on.
      1. BASIC intrps. @ line 200 only.
      2. User routine may periodically update this.
2. The Z-80 must first be set up for its mode IM0,IM1, or IM2.
3. The I register must be loaded with the (MSB) of the interrupt vector. (Address where routine address is found)
   a. I normally set to20H (Addrs.Chp. gives 62H) This VECTORS to 20B0H which is where NT decrment FC BC etc. is done @line 200. L.P. vector is first address preceeding this which is divisible by 16. For more details on auto prioritazation of L.P. see color tunnel program explanation.
   b. I register reset to normal vector on :RETURN
   c. User must determine I reg. value for hir routine.
4. The Z-80 must have also recieved the instruction EI before it will begin to recognize interrupts.

This completes the software initialization needed for interrupt processing with the address chip. See program descriptions and Port Chart for more information.

COLOR TUNNEL AND ART INSTRUCTIONS

Load the prog. (First prog. side 1) and stop the tape as soon as it RUNs. (Modifications follow) There will be a slight delay while the machine code loads itself to the Line Buffer. It will the come up with a multicolor display which will move like a tunnel every time a set of line are drawn in the ART routine. The keypad and Handle 1 have the following functions;

1. Holding the trigger down will cause the color tunnel to keep moving. The knob controls the speed.(CCW=fast)
2. After releasing the trigger hold the joystick forward wnich will cause it to go into a selection routine. The joystick must be held until the tunnel stops moving.
3. While in this routine the JY & JX will control the "resting position"of the colors. Left & Right for intensities, Forward & Back for color shades.
4. Also while in this routine 3 buttons on the bottom row of the keypad (shift keys) are active as follows.
5. The far right key (Words) will cause a random color pattern to be produced.
6. The next key to the left (Blue) will cause the color pattern now on the screen to be "locked" in when this routine is exited. (This is done by pulling the trigger)
7. The far left key (Green) will reset the colors to a x8 format.
8. If the random pattern button is pressed, and the program is returned to drawing (By pulling the trigger) a random color pattern will be produced automatically after every set of lines. So to stay on one of your selected combinations hit BLUE before pulling the TR. To cause random color pattern selection Just hit WORDS and pull TR. (Without "locking" w/BLUE)

NOTE: This program may be HALTED and re-run while the colors are being displayed. To return to the program with those exact same colors enter CLEAR;GOTO 140. You may even :INPUT or:LIST while the interrupts are active! (Prioritized INT) Try changing BC. (Its the black) If adding> 27 bytes :RETURN

FOLLOWING MODIFICATIONS

HALT the prog. and set up for :INPUT. Then start the tape and the modification for 4 lines per interrupt will be added to the program. Stop the tape as soon as it RUNs. It may take a while before you see any patterns due to the bkgnd. now beig set to a single color. JY, JX, & buttons stil work.

HALT again and enter CY=39;;INPUT. The mod. for 77 colors follows. This one splits the screen. After this has RUN, stop the tape as one more modification follows.

HALT the prog. and :INPUT again.(NOTE: only the preceeding modification need be entered to do this one) It is only a one line mod. which deletes the color reversal instruction. Stop the tape once it has RUN. This will now produce up to 144 different colors on the screen at the same time!

COLOR FORMATTER INSTRUCTIONS

Please read and do the following procedures to farmiliarize yourself with its operation before using it to add colors to one of your programs.

1. Load the formatter. (Ignore ? & other garbage in line 1)
2. Stop the tape and wait for autostart. (It first loads dummy numbers for scan lines,colors then "boots" the machine lang. down from line 1 to 20196 and then CALLs it)
3. The "B" in the upper left corner is your prompt to start entering numbers for the horizontal color bar widths down the screen. The red line starts at 0 and the numbers go to 250. This is where it overlaps again at top with pink.
4. Enter 0 and "GO", then HALT the program. Notice BC will set the color that is now white. The two thin lines at the top can be described as follows; The top line goes from -1 (255) to 0, and is wider than 1 scan line because these lines cannot be programmed into the Address Chip. The second line determines minimum distance (width) between interrupts. This means the minimum distance (Time it takes to run interrupt routine)is 5 scan lines.
5. RUN the program again (Or GOTO 20000), enter 12 & "GO"' Now the first line has been extended from -1 to 12. Now hit 88 & GO. The width of the second interrupt line has been made to extend from 12 to 88. This is approximatly one half way down the graphics area. (Two scan lines per pixel)
6. Try entering 92 to see what happens. That line number is less than 5 from 88 so it is unable to intrpt. there. It continues past 92 and causes a flicker as it goes all the way around again to 92. (Alternates between 88 & 92)
7. To correct this "mistake" enter 250 and hold down TR-1 while you hit "GO". This backs you up to 20004 where you can now enter a different number. Enter 180. Remember, any time you make a mistake enter 250 & GO while pulling the trigger. **(DO NOT BACK UP PAST LINE 1 @ 20002)**
8. **250 is the limit number at which no further scan lines can** be entered. This number is automatically entered by theprogram each time a line is entered so the program will reset and start at line -1 (255) on every scan. (Any line over 249 is detected in the program causing it to reset the line start and color start.)
9. Now halt the program and GOTO 20050. This is the routine for entering the horizontal boundaries and colors. The horiz. boundary is entered first, and is indicated by the "A" in the input sequence. Enter 20 & GO. Next is the right side background color (B). Enter 133 and GO. The rest of the 5 entries are as follows; "C" Left bkg. color (enter 0) "D" Right fgnd. color (enter 0) "@" Left fgnd. color (enter 4) You are now on A again which will be the horizontal boundary for the next line.(Next patch of colors down the screen)If you make a mistake you may back up as before. Hit any number and hold down the trigger while hitting GO. The rest of the screen may now be formatted. GOTO 20020 to re-enter line numbers if desired.

10. If you have approx. 30 sec. of tape to experiment with, try the following; After the colors have been set the way you want them, HALT the program and set NT=0. Then enter GOTO 20080, start your recorder on record, and hit any key. Some regiisters are dumped followed by a line 1, and then the machine language and colors. This line 1 is what is added to your program along with the machine code which is directly loaded. Stop the tape when it's done dumping, reset the BALLY and re-load what you just recorded w/:INPUT. (Wait for colors)

ADDING COLORS TO A PROGRAM

The program which you are adding the colors to must have at least 74 available bytes for the line 1 which loads the machine code and colors. Your program should be loaded first, and if there are string elements, you must add a dummy line1 w/exactly 71 characters and then re-record it. This will prevent the string values from being destroyed when the line 1 is added at the end. The formatter itself needs 552 bytes in order to run concurrently w/at least the part of your prog. which will produce the graphics or printing to be color formatted. This is only to set up the desired colors for your format. Once you've selected your setup, the 74 bytes for this selection will be added at the end of your prog.. The formatter will use line 1(Not the same one) and lines 20000 thru 20100.

1. First load your prog. and make room for the formatter by eliminating lines that will not affect the graphics.
   You need at least SZ=552 for now.
2. Load the formatter and HALT it after the colors come up.
3. RUN the part of your prog. which will produce the graphics to be formatted. NOTE: A :RETURN will turn off the colors, but you can re-start them again with CALL 20196. DO NOT ENTER MORE THAN 15 CHARACTERS OR USE THE TAPE RECORDER WHILE COLORS ARE ON, & DO NOT CALL 20196 IF YOU HAVE PUNCHED IN MORE THAN 15 CHARACTERS. FORMATTER MUST BE RE-RUN (GOTO 20000)
4. Now GOTO 20000 and set up your colors as described in the preceeding section.
5. After you have selected your format, put the tape with your program on it (Now set exactly at the end of your prog.) in the recorder.
6. HALT the prog. and GOTO 20080. Start recorder on record and hit any key. When the dump is finished your program is ready to be reloaded with the multicolor format being automatically loaded at the end. WAIT FOR THE COLORS TO START BEFORE STOPPING THE TAPE WHEN RE-INPUTING PROGRAM.

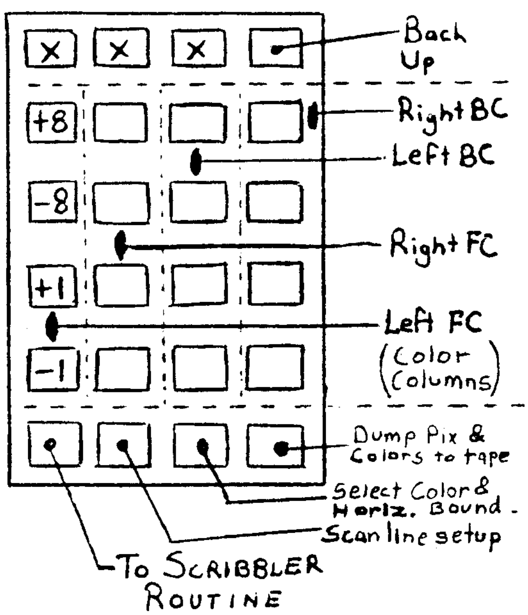
BEWARE!! The tape buffer is used to hold the interrupt lines, and if they are destroyed you may have problems getting the word in edgewise :RETURN. Re-initialize the first two line values as follows; %(20002)=10;%(20003)=250

If the tape recorder has been used with the colors running, the whole program may have to be reloaded. You may enter line values manually as follows; Starting from 20002 enter %(200??)=nnn-1536, which equates to adding 250 after each line number.

COLOR SCRIBBLE And RECORD

This has been made a seperate program. (Side 2 following the MPU) Load program ignoring garbage in line 1. (Same machine code used in color formatter)

1. When the screen turns yellow stop the tape. A red line will appear at the top of the screen when the machine code starts up.
2. Refer to the keypad overlay diagram while doing the following steps. Push forward on joystick 1 to move the line down like a curtain. Pulling back will move it up again.
3. Pull the trigger only until you see the next line appear. If you should pass it up by holding the trigger down (Or if you make a mistake) hold down the LIST button and this will back you up to the preceeding lines. You may now continue on down the screen setting up lines wherever you want them. You may return to this routine to change line positions at any time as will be explained. Or you may prefer not to start in this routine, but in the scribbler routine as follows.



4. To get into the scribbler routine hold down the far left button. (At least two seconds so it is detected.) This routine will operate similar to the onboard scribbler. Turn the knob to get different size boxes, use the joystick to move the cursor around, and pull the trigger to draw or erase. Turning the knob fully clockwise will allow you to move the cursor faster.
5. When moving from one routine to the other across the bottom "control" buttons, move from one to the next in order, one at a time. Hold down the red button (scan lines) then hold down the blue button (colors). Now you may select 4 different colors & the horiz. boundary for each INT line.
6. Hold the JY to the left or right to move the horiz. boundary. The buttons on the keypad will now select the colors on each line as in the onboard scribbler. See KP diagram for functions of each button. You have left and right background, and left and right foreground for each line. If you should accidentally hit the HALT button enter L=C;GOTO 250 and this will start you back in the scribbler.
7. To set the colors on the following lines down, pull the trigger until you see the horizontal boundary from the preceeding line appear on the next line. The color buttons and JY will now be active on this line. It is helpful to have the horiz. bound. in a different place when moving from line to line. If a mistake is made hold down the LIST button to back up as before.
8. Now push the JY forward while moving to a different routine. This will put you in the routine for changing FC & BC. JX left while turning KN controls FC. JY forward or back controls BC. Releasing the joystick will lock on the selection and you may return to any of the other routines.

(9) It will take about 5 min. of tape to record the complete screen picture and colors. Put a fresh tape in the recorder and start it on record. Move across the bottom control buttons until you hit WORDS. Pull the trigger and the the dump will begin. The colors are turned off during the dump, and the upper left corner of the screen will be destroyed. This can be repaired when the picture is reinput, as it comes up in the scribbler routine. Dump is complete when colors return.
(10). When your sure you have a good copy HALT the program and PRINT FC & BC, and keep a record of them with your picture tape. To input the picture again enter your values for FC & BC, and then simply:INPUT, and start tape on play.

MACHINE PROGRAMMING UTILITY

The following is a quick reference guide to operating the MPU.

(1) ÷ LISTs the machine program from "S" (Starting address). This is set on entery to the programming routine (x). Holding down the 8 key will back up the listing by 1. Other keys in the PAUSE column will make the listing jump back by a greater amount. Holding down the blue shift key will replace the binary readout with the 16 bit "%" value (Two bytes) (Addresses)
(2) x Enter machine code. Enter your starting address, then enter program byte by byte. Address will automatically increment by 2 if a number over 255 is entered (Addresses). If a mistake is made while entering code, remember the address you made the mistake on and enter -2 and GO. You may now enter this address , make the correction, and continue. To return to the menu enter -1 and GO. (This will not cause a write)
(3) - Move prog. to line 1. This will move the (Bytes+1) of your prog. to line 1. (periods) (Loads from S to S+I) This should be done when HALTing a prog. to enter a BASIC segment if there is a prog. in the line buffer. To re-copy prog. from line 1 to the RUN area(buffer), set X=0 before re-running utility. BASIC segment must not destroy S or I.
(4) + Enter and CALL address. To RUN machine code without HALTing the utility. Enter CALL address and GO.
(5) = Dump to tape. This routine will dump an auto loading loop and the machine code from S to S+I to tape.
(6) 1 Hex to Dec. Enter 4 digit Hex code. To return to menu enter 4 digits and quickly hit WORDS.
(7) 2 Dec. to Hex. Positive numbers only. To return to menu enter a negative number and GO.

Familiarity breeds contempt, so load the futility and perform the following exorcise.

Press the ÷ key. The program will (faithfully?) list machine code starting from 11822 (In BASIC ROM- value of two periods) The periods in line 2 will normally be used to store the starting and ending address of your machine program. When the prog. is first RUN (X=0) the S variable will be set to the starting address found in line2, and if there is a legitimate ending address, the program in line 1 will be booted to the RUN area.

The format for the listing is as follows;

Decimal address-Hex address-Decimal data-Hex data-Binary or 16bit

11822	2E2E	32	20	(%) Dec.

Holding down the BLUE shift key will replace the binary readout with the 16 bit "%" value. (For addresses) To make the listing back up hold down the "8" key. Other keys in the PAUSE column will back up the listing faster. 8x-1 5x-3 2x-7 0x-15 and red shift x-31 To get back to the menu while in LIST mode hit the WORDS key. For Hex to Dec. conversions hit the number 1 key. Enter a 4 digit Hex number (Try FBC9- =-13829) This gives you the legal 16 bit value to enter w/%. (Puts two bytes in proper perspective) To return to the menu enter 4 digits and quickly hit WORDS. For Dec. to Hex hit the number 2. This routine will only work on positive numbers. To return to menu enter a negative number. Now hit the x key. You are prompted to enter an address to start entering your machine code at. If you are not using external RAM you must use one of the free areas described on pg. 6. (Such as Line buffer) The number you enter goes into the "S" variable and is refered to for LIST, DUMP to tape, and STORE in LINE 1. Enter a starting address of 20202 and enter the following program to help demonstrate the other features of the utility. If you make a mistake enter -2 and GO. Then enter the address you made the mistake on and continue entering the rest of the numbers.

20202	213	20210	33	20218	216	20226	19	20234	230
20203	197	20211	64	20219	26	20227	16	20235	241
20204	229	20212	65	20220	174	20228	246	20236	225
20205	245	20213	17	20221	230	20229	13	20237	193
20206	1	20214	128	20222	170	20230	32	20238	209
20207	0	20215	66	20223	174	20231	243	20239	201
20208	1	20216	1	20224	119	20232	193		
20209	197	20217	10	20225	35	20233	16		

Now we want to enter a BASIC segment which is going to destroy all the numbers you just entered in the line buffer. You don't want to do that do you? So first enter a -1 to return to the menu, then hit the - (minus) key. This will store the machine code in line 1 of the utility. It can then be reloaded to the line buffer (RUN area) as will be explained. When the menu returns, HALT the utility and examine lines 1&2. Notice the machine code (garbage) which has replaced some periods in line 1. The characters in line 2 are the starting and ending addresses of the RUN area. Remember to hit the minus key whenever using the utilty w/o external memory. (For use of line buffer)

So now enter the following BASIC segment.

```
500 CY=-32
510 FOR A=1TO 26;TV=RND (26)+64;NEXT A
520 CALL20202;GOTO 500
```

After entering this, set X=0 and RUN the utility again. X must be set to zero to cause the utility to load the machine code from line 1 to the RUN area again. Wait a few seconds to make sure all of the code has been transfered, and then HALT the utility. Now enter goto500 and watch the full character 9 line scroll for pixel shared do its thing. (Similar to APPLE) I will leave this one to you to figure out.

You may now record the machine code by recording lines 1 & 2, the initialization for starting and ending address, (Line 270) and the loader to the RUN area.(Line 250) Or you can just record the whole utility, and when it is re-run (X=0) the code will be automatically booted to the Line buffer. You can also use the dump to tape routine which will simply load back into the Line buffer directly when it is re-input.

The lines of interest in the utility for program storage in line1are; Line 140-moves prog. to line 1. 1 is added to each byte before it is poked in line 1 to keep a CR from being detected.(Decimal 12 not allowed for Line 1 storage)
Line 150 moves prog. from Line 1 to RUN area. 1 is again subtracted from each byte to get the correct number back.
In line 270 the -24573 is the first byte which is used for program storage in Line 1, and the -24471 is the address where your starting address is stored in Line 2. Line 1 can hold 98 bytes of data and has 99 periods in it initially. (+2 bytes for Line # = 101 + CR = 102 bytes total)

PROGRAM DESCRIPTIONS

All the programs use similar interrupt structuring, with the exception that some do not have interupt prioritization capabilities. (Between Light pen and Screen) Since the tape buffer is used to hold scan line numbers in the Formatter, and the onboard Light pen routine is the 300 baud input, an :INPUT or :LIST will destroy the scan line numbers there. (BASIC Light pen interrupts not allowed)

As was mentioned the two interrupts which are possible with the existing hardware are screen and light pen. Since both tape input or Light pen data must be presented to the processor rather immediatly, the light pen INT is designed to take priority. The way this is accomplished is the address chip will re-interrupt the Z-80, and this time (Light pen) will send back the LSB minus the lower four bits. (The lower four bits of the interrupt feedback number are made zero.) This automatically sends the Z-80 to the first address back from the interrupt vector which is divisible by 16. At this address is the address of the routine to process the Light pen interrupt.

Refer to the Color Tunnel machine language listing. Looking at the address 20208 you will see the number 8438. This is the address of the Light pen processor (Tape) in BALLY BASIC. The address 20208 is divisible by 16. The next address,20214, is the address of the start of the screen interrupt routine which produces the colors. Note that the Light pen interrupt routine address may be different in different versions of BASIC. The important one to know is the normal SCREEN interrupt routine address. This is periodically CALLed in the interrupt routine so that the NOTE TIMER can be decremented when entering data from the keypad. If this routine is not periodically CALLed the machine will hang up due to the note timer being at a value other than zero, and not being decremented. (This is done in BALLY BASICs SCREEN INT processor routine @ scan line 200)

In the Color Tunnel program this "normal" processor routine is called at a time when screen interrupts are finished for one frame. I call this SCAN RESET time. The reason it is done at the bottom of the frame is because the FC & BC are also set during BASICs normal interrupts. (Would show up on the screen in the wrong spot(s)) In the Color tunnel the CALL to the BASIC screen interupt routine is done at 20237 (CALL 8368), and the reset scan sequence starts at 20229.

COLOR TUNNEL

Starting at 20142 (Tape buffer) is the interrupt initialization process. First the MSByte of the interrupt vector is loaded to the Z-80s I register (via acc.) The INT feedback number (LSByte) is loaded to the address chip via port 13. Notice if you multiply the number @ 20144 by 256 and add it to the number @ 20148 you get 20210. Location 20210 is the interrupt vector for the color routine. (The address @20210 is the start address of the interrupt routine) During the interrupt initialization process interrupts are disabled, and re-enabled again at the end. This is because if the Address chip did an interrupt right in the middle of changing the interrupt vector, the Z-80 would end up in some unbeknownst area of memory trying to run code (Can be very beautiful) This initialization routine is only executed once at the start if the color tunnel. Thereafter the interrupts send the Z-80 to the code starting @20214. Since light pen interrupts are enabled on :INPUT or :LIST this need not be done. Screen interrupts are also enabled on RESET and at line 200, the routine will be diverted to the color tunnel interrupts, and the interrupt lines will be changed (When INT is to occur) thereafter.

The first thing that is done on routine entry is to PUSH all the registers about to be used onto the stack. Then the register pair HL is loaded with an address. This is the location where the present scan line will be stored. The INC & DEC do nothing except help clean up some of the pseudo hi-res lines that appear due to the DATA chip not being able to complete all the color I.O.'s exactly at the start of each line. (Timing only) The scan line number is thenadded to 14, which is width of each color bar. It is then checked to see if it has gone over 179 yet, and if so the programfalls thru the RESET SCAN sequence. The first thing done in the reset scan is to load the Acc. with the value of BASIC's "P" register. This is used for the starting color (Top of screen next scan cycle). HL is incremented to the next address which is used to hold the present line color. (Scan line @ 20154 & Color @ 20155) This address is loaded with the color from P at 20233. The Acc. is zeroed (XOR A) and 8 is subtracted from it to cause the starting scan line (Top of screen) to be slightly above the graphics area. (Acc. now holds new start scan line for top screen interrupt) The normal SCREEN INT processor is then CALLed in BASIC ROM, and the program jumps to 20278 (Done). The scan line is loaded to the address chip via port 15, the POPs are done, interrupts enabled, and it returns to BASIC processing. If the scan line was not over 179, the program jumps from 20227 to 20244. This is where the color generation is picked up again on every interrupt. HL would be on the scan line storage address, and the Acc. holds the NEXT INT line. This is then stored back for future reference at 20246. HL is then incremented to the color storage address, and then a check is made to see if the present scan line is over the point at which the colors should be made to reverse. (Half way down the screen) If it is over 63 (HALF-next) 8 colors are subtracted from A, and a jump is made over the color increment. (@ 20257) The BASIC segment modifies the INC and DEC (n) values for random colors other than x8. At 20267 the colors are OUTPUT to the DATA chip. 4 intensities are added on @ 20264 for the Bkgd. colors. (4 & 5) HL is then backed up to the scan line storage address and it is checked to see if it is on the last line. If so 6 is added to the line number for the last line to make it appear thicker like the top bar. Then the Address chip is loaded with the scan line number, and the interrupt is terminated.

COLOR FORMATTER

The interrupt initialization for the formatter is from 20196 to 20206. This is essentially the same as for the color tunnel except the interrupt vector is now 20210. The program starts at 20212 with the PUSHES, and HL being loaded with an address. This address is incremented by 5 every time the INT routine is executed. (5 values are OUTput from an index - Colors and Horiz. bound.) It finds this present index starting address (where to get the next 5 values) at 20048 with the instr. LD HL,(NN) @ 20215. (Can be called an index vector) The first value picked up from the index is for the horiz. boundary. This is loaded to the Acc. and then output to port 9, HL is incremented, the first color is picked up and output to ports 0 & 1. The rest of the colors are then output from the index in a similar fashion, all the way to 20246. This is where HL is again INCremented one more time so it now points to the next set of 5 values to be output on the next interrupt. This address is then stored back @ 20048 for reference on the next INT. HL is then loaded with 20000 which holds the present address where the scan line will be found. The LSB is then incremented to point to the next scan line. (NOTE: This increment on the address will only work in a range of 256 addresses starting from an address which is divisible by 256. The INC (HL) instr. is only an 8 bit INC on the byte @(HL). This is important if you plan to move this routine to external memory for more colors or multi-screen formats) The scan line is then picked up and checked to see if it is over 249. If so (See 20257) the program falls thru the RESET SCAN sequence. In this part the HL register pair is loaded with 20201, so when the address @ 20000 is incremented it will point to the first scan line number @ 20002. This is then stored at loc. 20000. Then HL is loaded with the address of the very first byte in the color and boundary index. This is then loaded back to loc. 20048, where it will find the address for the top screen values for the next INT.

The BASIC screen INT processor is CALLed, The Acc. is set for the top line, and the top line # is loaded to the address chip. The POPs are done and the interrupt is then terminated.

If the scan line was not over 249 (@20257) the program jumps to 20279. The Acc. will have the next value for the next INT line, and it will be loaded to the address chip, and the interrupt terminated.

BASIC HELPERS

1. To get the single byte values from a 16 bit value read by the % command... First divide by 256 and print RM. If the RM is less than zero add 256. This will be your MSB. If the quotient is less than zero add 255. This is your LSB.

2. To get the 16 bit value from two consecutive bytes... First multiply the second byte by 256, then add this to the first byte. If your answer is over 32767 subtract 65536. Or simply poke the two bytes into two consecutive memory locations and reed the first one.

3. To enter characters directly to memory while displaying them on the TV use the following format;
```
10 FOR A=(Starting adrs.)TO(Ending adrs.)
20 K=KP;TV=K;IF K=31 TV=32;TV=K+256;A=A-2x(A>0);NEXT A
30 %(A)=K;NEXT A
```
This is used when using @ storage

4. To read characters from memory. (for dumping memory to tape or to find the address of a specific character)
```
10 FOR A=(Start adrs.)TO(End adrs.)
20 TV=%(A);NEXT A
```
Stop any time and PRINT A and TV=%(A) to see where your at.

5. To verify a tape from a section of memory it was just dumped from. First always leave a short pause between the loader and before the dump. Then stop the tape on that blank spot and enter the following;
```
B=(Start adrs.);C=(End adrs.);D=20300;E=20301
:INPUT;FOR A=B TO C;%(D)=%(A);%(E)=0;IF KP=%(D)NEXT A
```
Start the tape and hit GO. If the tape is good the cursor will not return until A = your end adress+1. (PRINT A when the cursorreturns and see if it made it to the end)

6. To load a complete tape (characters) to memory so the whole tape (side) can be recorded at one time.
```
10 :INPUT
20 FOR A=(Start adrs.)TO(End adrs.)
30 %(A)=KP;NEXT A
```
Start the tape on play and RUN the program. When the cursor returns the pre-determined amount of memory from your loop has been loaded (That's it baby) Find the seperation addres-es between the programs (CR) (See #4 above) and add a time delay here when dumping back to tape.

7. Single byte write using % (One line whammy)
M=Address of byte N=desired value
```
%(M)=((%(M)÷256)x256)-((%(M)<0)x256)+N
```

Normal Int. Proc. Locations

Astro Basic....CALL 8701   Bally Basic....CALL 8368   EXT> Basic(Both)....CALL 8385

Viper + Blue Ram

BASIC PROGRAM LISTINGS

```
**************** COLOR FORMATTER ****************     b=multiply
                                                        c=divide
    1 .?OH?!O+QO$$$$#QO'!O5+!O9''O#!O'?F#QO!C. machine code
20000 :RETURN ;C=20286;D=20002;E=20196;F=-77;G=40;%(D-2)=D-1;%(D+46)=C init.
20010 B=E;FOR A=-24573TO -24484;%(B)=%(A)-1;B=B+1;NEXT A;CALLE Load Line) Data to RUN area.+ RUN
20020 FOR A=DTO D+G Input interrupt lines (to 20040)
20030 CX=F;CY=G;PRINT A,;INPUT B;%(A)=-1536+B;IF TR(1)A=A-1;GOTO D+28
20040 NEXT A
20050 FOR A=CTO C+120 Input Colors + H-bound-(to 20070)
20060 H=Ac5;CX=F;CY=G;TV=RM)64;INPUT B;%(A)=((%(A)c256)b256)-((%(A)<0)b256)+B;IF TR(1)A=A-1;GOTO D+58
20070 NEXT A
20080 IF KP:PRINT ;PRINT ;PRINT "C=",C,";D=",D,";E=",E,";G=",G Dump to Tape
20090 PRINT "1FOR A=ETO E+210;%(A)=KP;NEXT A;:INPUT ;FOR A=DTO D+G;K=KP;NEXT A;:RETURN ;%(D+46)=C;%(D-2)=D-1;CALLE
20100 PRINT "RUN ";CY=39;FOR A=ETO E+210;TV=%(A);NEXT A;FOR A=DTO D+G;TV=%(A);NXT A
```

THIS LINE FOLLOWS THE FORMATTER TO LOAD DUMMY NUMBERS INTO TAPE BUFFER (SCAN LINES) TO KEEP SCREEN FROM COMING UP FLICKERING , AND INTO LINE BUFFER (COLORS) TO KEEP SCREEN FROM COMING UP BLACK

```
B=92;FOR A=20286TO 20391;B=B+3;%(A)=B;NEXT A;:INPUT;
B=-4;FOR A=20002TO 20018;B=B+16;%(A)=B;NEXT A;
:RETURN;RUN;.STOP TAPE
```

Do not enter more than 27 bytes while interrupts are active!

SZ=1248 - %(20050)=-24020 (not part of program) - To stop at any time for changes or tape use execute :RETURN

If tape is used after program is in,reinitialize as follows before re-running - %(20002)=10;%(20003)=250

PLEASE READ INSTRUCTIONS CAREFULLY BEFORE USING WITH OTHER PROGRAMS...........see pg. 27

```
  1 .
  2 .                          b=multiply
  3 .AMERICAN FLAG             c=divide
  4 .
 10 :RETURN ;CLEAR ;NT=0;FC=230;GOSUB 120
 20 CALL20280
 30 Y=38;A=75;D=3
 40 FOR Y=YTO -5STEP -5
 50 D=-D
 60 FOR X=-A+DTO -18STEP 12
 70 BOX X,Y,5,4,1;BOX X,Y,5,2,3;BOX X,Y,3,4,3
 80 BOX X,Y-1,1,1,3;BOX X,Y+2,1,2,3
 90 NEXT X;NEXT Y
100 &(2)=156;&(3)=156
110 GOTO 110
120 A=20200;C=600
130 X=20202;GOSUB C
140 X=-6715;GOSUB C
150 X=245;GOSUB C
160 X=11809;GOSUB C
170 X=32334;GOSUB C
180 X=3782;GOSUB C
190 X=9031;GOSUB C
200 X=-18434;GOSUB C
210 X=2616;GOSUB C
220 X=-20275;GOSUB C
220 X=-20275;GOSUB C
230 X=15904;GOSUB C
240 X=30471;GOSUB C
250 X=20399;GOSUB C
260 X=2840;GOSUB C
270 X=-14722;GOSUB C
280 X=-431;GOSUB C
290 X=14426;GOSUB C
300 X=-10750;GOSUB C
310 X=30626;GOSUB C
320 X=-11441;GOSUB C
330 X=-11520;GOSUB C
340 X=30721;GOSUB C
350 X=25598;GOSUB C
360 X=1592;GOSUB C
370 X=-11345;GOSUB C
380 X=30985;GOSUB C
390 X=536;GOSUB C
400 X=2110;GOSUB C
410 X=1235;GOSUB C
420 X=1491;GOSUB C
430 X=-392;GOSUB C
440 X=14519;GOSUB C
450 X=15878;GOSUB C
460 X=-11503;GOSUB C
470 X=-20727;GOSUB C
480 X=11079;GOSUB C
490 X=-11401;GOSUB C
500 X=-3825;GOSUB C
510 X=-15903;GOSUB C
520 X=-13829;GOSUB C
*530 X=16115;GOSUB C
540 X=-4786;GOSUB C
550 X=15943;GOSUB C
560 X=-11288;GOSUB C
570 X=-1267;GOSUB C
580 X=201;GOSUB C
590 RETURN
600 %(A)=X;A=A+2;RETURN
```

* Interrupt Initialization

SZ=952

%(20050)=-23724

Do not enter more than 17 bytes while interrupts are active !

If desired , this program may be shortened considerably, for use with games, music, etc. Only lines 10 thru 100 are needed to draw the stars. Add this line before running the program.

```
1000PRINT"FOR A=20200TO 20290;%(A)=KP;NEXT A";FOR A=20200TO 20290;TV=%(A);NEXT A
```

Run the program, and when the colors come up halt it. Then set to the end of the tape of the program you wish to add it to. Enter CLEAR;:PRINT;NT=0;GOTO1000 START THE TAPE ON RECORD AND HIT GO. To bring the colors up whenever you want them in the program CALL 20280. To stop them :RETURN

Also , an alternate way of storing the machine language in the program which is still much shorter, is to put it into line 1 as in the Color Formatter. This is discussed in more detail herin.

```
*********************************   COLOR SCRIBBLE AND RECORD   **********************************
  1 .?0H?!0+Q0$$$$#Q0"!05+!09""0#!0"?P#Q0!C. machine code
  5 CLEAR ;GOSUB 10;GOTO 20
 10 ;RETURN ;C=20286;D=20002;E=20196;S=D;L=C;M=256;F=-77;G=40;%(D-2)=D-1;%(D+46)=C;NT=0;RETURN  init.
 20 B=E;FOR A=-24573TO -24484;%(B)=%(A)-1;B=B+1;NEXT A;CALLE  Load Line 1 data and RUN colors
 30 I=1;K=0;J=80  scan line init. J= H.bound init.
 40 I=I+JY(1);IF I<@(K)+5I=I+1  Adjust and enter intrp. lines
 50 %(S)=%(S)cMbM+I-Mb(%(S)<0)  thru 90
 60 IF TR(1)IF S<D+42S=S+1;K=K+1;@(K)=I;I=I+5;%(S)=-1536+I  poke line, add 250 nt.
 70 IF &(20)=1IF S>D S=S-1;K=K-1;I=@(K)+5;%(S+1)=250  LIST Mistake
 80 IF (&(23))+(&(21))GOTO 100
 90 GOTO 40
100 IF JY(1)BC=KN(1);GOTO 100  set FC BC
110 IF JX(1)FC=KN(1);GOTO 110  and check keys
120 IF &(20)GOTO 300  To tape dump
130 IF &(21)GOTO 170  To H.bound. + colors
140 IF &(22)GOTO 40   To Intrp. lines
150 IF &(23)GOTO 250  To Scribbler
160 GOTO 100
170 J=J+JX(1);J=Jb(J<M)+(J<0);IF TR(1)IF L<20390L=L+5  H.bound. Don't crush stack
180 %(L)=%(L)cMbM+J+((J=0)bM)-Mb(%(L)<0);IF JX(1)FOR V=1TO G;NEXT V;GOTO 170  Poke H.bound - speed up
190 FOR A=20TO 23;N=&(A);L=L+1;B=8b(N=2)-8b(N=4)+1b(N=8)-1b(N=16)  key Scan for colors
200 IF B#0O=%(L)cM;O=RM+B+Mb(RM<0);O=O+Mb(O<0)-255b(O>255);%(L)=%(L)cMbM+O-Mb(%(L)<0)  set color
210 FOR V=1TO Gb2;NEXT V;IF &(A)IF B#0GOTO 200  speed up
220 NEXT A;L=L-4;IF (&(20)=32)+(&(21)=32)+(&(22)=32)+(&(23)=32)GOTO 100  Moving!
230 IF &(20)=1IF L>C L=L-5  LIST mistake
240 GOTO 170
250 FOR R=0TO 32767;IF ABS(Y)>43Y=-44b(Y<0)+44b(Y>0)  Scribble loop - Y bound. limit
260 P=1+(KN(1)+128)c32;Q=1+(RM<10);X=X+JX(1);Y=Y+JY(1);IF TR(1)BOX X,Y,P,P,Q;GOTO 260  Box size - draw
270 X=X+(8bJX(1)b(KN(1)>125));Y=Y+(8bJY(1)b(KN(1)>125));BOX X,Y,P,P,3;BOX X,Y,P,P,3;IF &(22)GOTO 40
280 IF ABS(X)>80X=-80b(X<0)+79b(X>0)  X bound. limit
290 NEXT R;GOTO 250
300 GOTO 300+TR(1)b10  Start dump on TR
310 ;RETURN ;;PRINT ;NT=1;FOR A=-23121TO -23091;CY=G;CX=F;TV=%(A);NEXT A  Reed line 370
320 FOR A=16384TO 19983;CY=G;CX=F;TV=%(A);NEXT A;FOR A=1TO 200;NEXT A  Dump Pix
330 FOR A=ETO E+210;CX=F;CY=G;TV=%(A);NEXT A;FOR A=DTO D+G;CY=G;CX=F;TV=%(A);NEXT A  Dump code + colors
340 CALLE;GOTO 250  Done - Return to Scribbler
350 GOSUB 10;;INPUT ;FOR A=ETO E+210;%(A)=KP;NEXT A;;INPUT ;FOR A=DTO D+G;K=KP;NEXT A;;RETURN  Input code + colors   (Load Scan lines)
360 %(20050)=-23086;%(D-2)=D-1;%(D+46)=C;CALLE;GOTO 250  init. TXTUHF - start colors - goto scribbler
370 FOR A=16384TO 19983;%(A)=KP;NEXT A;GOTO 350  ---> This line is red by line 310 on tape dump (address of "FOR" must be -23121)
>
```

THIS LINE FOLLOWS THE COLOR DRAW AND RECORD TO LOAD DUMMY NUMBERS INTO TAPE BUFFER SCAN LINES TO KEEP SCREEN FROM COMING UP FLICKERING, AND INTO LINE BUFFER (COLORS) TO KEEP SCREEN FROM COMING UP BLACK.

```
FOR A=20286TO 20390;C=C+3;%(A)=C;NEXT A;
%(20002)=20;%(20003)=250;RUN;.STOP TAPE
```

b=multiply
c=divide

%(20050)=-23086
SZ=314

See pg. 29 for instructions

NOTE: ;INPUT resets tape buffer pointer to 20002

Magic Color Mask Numbers (If poked to screen)

0=port 0 & 4   85=port 1&5   170=port 2&6   255= 3&7   If these numbers are poked into screen memory they will produce the designated colors, (per port specified). Examine bit patterns of these numbers.

0........85.....170..... & 255......   0000 0000   0101 0101   1010 1010   1111 1111

NOTE SCALE ..to reproduce legal note frequencies
as closely as possible. &(16) =49 first.

D	D#	E	F	F#	G	G#	A	A#	B	C	C#
241	230	216	205	193	180	171	160	152	142	136	128
120	115	107	101	95	90	85	80	75	71	67	63
60	57	53	50	47	44	42	39	37	35	33	31
29	28	26	24	23	22	20	19 (400Hz)	18	17	16	15
14	13	12	X	X	X	X	X	X	X	X	X

** MACHINE PROGRAMMING UTILITY **

SZ=466 %(20050)=-23238 1334 BYTES TOTAL

```
  1 .........................................................................................
  2 .....
 10 H=190;M=256;GOTO M
 20 INPUT "STARTING ADDRESS"S;R=S;%(P)=S;I=-1
 30 PRINT R,"=",;INPUT ""D;IF D<0GOTO M
 40 IF D>M%(R)=D;R=R+1;IF I<R I=I+1
 50 IF D<M%(R)=D+%(R)+MxM-Mx(%(R)<0)
 60 R=R+1;IF I<R I=I+1
 70 GOTO 30
 80 Q=Q+1;PRINT #0,Q,"*",;GOSUB 180;IF &(20)RETURN
 90 D=%(Q)+M;D=RM+Mx(RM<0);PRINT #3,D," ",;B=D;D=D+16;GOSUB H;D=RM;GOSUB H;PRINT " ",
100 Q=Q-&(22);IF &(21)PRINT #2,%(Q);GOTO 80
110 F=128;GOSUB 120;GOTO 80
120 FOR C=1TO 8;E=B+F;PRINT #0,E,;B=RM;F=F+2;NEXT C;TV=13;RETURN
130 PRINT "CALL";INPUT ""C;CALLC;GOTO M
140 FOR G=STO S+I;%(O)=%(G)+1;O=O+1;NEXT G;GOTO M
150 FOR G=%(P)TO %(P+2);%(G)=%(O)-1;O=O+1;NEXT G;RETURN
160 PRINT "START TAPE--PRESS ANY KEY";K=KP;;PRINT ;FOR A=1TO 30;TV=31;NEXT A
170 PRINT "FOR A=",#0,S,"TO ",#0,S+I,";%(A)+KP;NEXT A;;RETURN ;BC=133";FOR A=STO S+I;CY=40;TV=%(A);NEXT A;GOTO M
180 D=Q+4096;GOSUB H;D=RM+M;GOSUB H;D=RM+16;GOSUB H;D=RM;GOSUB H;PRINT " ",;RETURN
190 IF D>9TV=D+55;RETURN
200 TV=D+48;RETURN
210 FOR A=0TO 3;K=KP;TV=K;IF K=31TV=32;TV=K+M;A=A-2x(A>0);NEXT A
220 K=K-7x(K>64)-48;@(A)=K;Q=A+2;IF RM=1%(19997+Q)=@(A-1)x16+@(A)
230 NEXT A;PRINT %(19997);IF &(20)GOTO M
240 GOTO 210
256 CLEAR ;;RETURN ;NT=0;IF D=-2D=0;INPUT "CORRECTION @"R;GOTO 30
260 PRINT "+ LIST ";PRINT "x ENTER MACH.CODE";PRINT "- MOVE PROG.TO LINE 1";PRINT "+ ENTER & CALL ADDRS.
270 PRINT "= DUMP TO TAPE";PRINT "1 HEX TO DEC.";PRINT "2 DEC.TO HEX";O=-24573;P=-24471;IF X=OS=%(P);GOSUB 150;X=1
280 IF &(20)=1CLEAR ;Q=S-1;GOSUB 80;GOTO M
290 IF &(20)=2CLEAR ;GOTO 20
300 IF &(20)=4CLEAR ;GOSUB 140
310 IF &(20)=8CLEAR ;GOTO 130
320 IF &(20)=16CLEAR ;GOTO 160
330 IF &(23)=8CLEAR ;GOSUB 210
340 IF &(22)=8CLEAR ;GOTO 370
350 IF &(21)=16D=-2;GOTO M
360 %(P+2)=S+I;GOTO 280
370 INPUT ""Q;CY=CY+8;CX=-40;PRINT "=",;GOSUB 180;PRINT ;IF Q<0GOTO M
380 GOTO 370
```

X=Entry semiphore
R=Entry pointer
I=Byte counter
S=Starting address
D=Data ; EQ -1 To menu ; EQ -2 To corr.@
Q=List address counter
G=Counter-Block loaders
O=Line 1 begin entry
P=Line 2 start address ; P+2=End address

```
1 .
2 .
3 .
4 .
5 . * COLOR TUNNEL *
10 CLEAR ;:RETURN ;NT=0;BC=0;FC=116;GOSUB 360;A=0;X=0;P=100;M=0;GOTO 140
20 FOR P=PTO P+760STEP 8;FOR C=-128TO KN(1);NEXT C;NEXT P;P=P-768
30 IF TR(1)GOTO 20
40 IF JY(1)GOTO 300
50 RETURN
60 FOR G=1TO Z;IF TR(1)GOTO 20
70 X=X+H;Y=Y+I;A=A+S;B=B+T;U=A-S;V=B-T;LINE U,V,4;LINE X,Y,F;LINE A,B,E
80 LINE -U,-V,4;LINE -X,-Y,F;LINE -A,-B,E;LINE -U,V,4;LINE -X,Y,F;LINE -A,B,E;LINE U,-V,4;LINE X,-Y,F;LINE A,-B,E;NEXT G
90 RETURN
100 G=0;C=1-0+1;0=1-0;FOR R=0TO 79
110 S=43-(RbR)c142;IF G=SNEXT R
120 G=S;LINE 79,S,4;LINE R,S,C;LINE -R,S,4;LINE -80,S,C;LINE -80,-S,4;LINE -R,-S,C;LINE R,-S,4;LINE 79,-S,C;NEXT R
130 LINE 79,-44,4;LINE -80,-44,2;LINE -80,43,2;RETURN
140 GOSUB 20;IF MGOSUB 270
150 Q=Q+1;IF Q>20Q=0;GOSUB 100
160 F=RND (4);E=RND (2);IF F=4F=1
170 IF F=3IF E=1F=2
180 IF E=1IF F=1F=2
190 Z=RND (20)+5
200 H=RND (9)-5;I=RND (9)-5
210 S=RND (8)-4;T=RND (8)-4
220 IF (ABS (A))-X>50A=0
230 IF (ABS (X))-A>50X=0
240 IF ((ABS (B))-Y>30)+((ABS (Y))-B>30)B=0
250 IF (ABS (ZbI+Y)>43)+(ABS (ZbT+B)>43)+(ABS (ZbS+A)>79)+(ABS (ZbH+X)>79)GOTO 160
260 GOSUB 60;GOTO 140
270 N=RND (255)
280 D=20254;IF %(D-2)c256#-41RETURN
290 FOR R=1TO 2;%(D)=((%(D)c256)b256)-((%(D)<0)b256)+N;D=D+4;NEXT R;RETURN
300 IF TR(1)GOTO 300
310 IF &(23)=32N=8;GOSUB 280
320 IF &(20)=32M=1;GOSUB 270
330 IF &(21)=32M=0
340 P=P+JX(1);P=P+(JY(1)b8);IF TR(1)=0GOTO 310
350 RETURN
360 A=20142;C=810
370 X=16115;GOSUB C;X=-4786;GOSUB C
380 X=15943;GOSUB C;X=-11278;GOSUB C
390 X=-1267;GOSUB C;X=201;GOSUB C
400 A=20208
410 X=8438;GOSUB C
420 X=20214;GOSUB C
430 X=0;GOSUB C
440 X=-6715;GOSUB C
450 X=8693;GOSUB C
460 X=20154;GOSUB C
470 X=11043;GOSUB C
480 X=-14722;GOSUB C
490 X=-498;GOSUB C
500 X=14515;GOSUB C
510 X=14863;GOSUB C
520 X=20108;GOSUB C
530 X=30499;GOSUB C
540 X=-10577;GOSUB C
550 X=-13048;GOSUB C
560 X=8368;GOSUB C
570 X=43;GOSUB C
580 X=8728;GOSUB C
590 X=11043;GOSUB C
600 X=9079;GOSUB C
610 X=25598;GOSUB C
620 X=14462;GOSUB C
630 X=-10748;GOSUB C
640 X=6152;GOSUB C
650 X=-14846;GOSUB C
660 X=30472;GOSUB C
670 X=2003;GOSUB C
680 X=1747;GOSUB C
690 X=1222;GOSUB C
700 X=1491;GOSUB C
710 X=1235;GOSUB C
720 X=32299;GOSUB C
730 X=-21506;GOSUB C
740 X=568;GOSUB C
750 X=1734;GOSUB C
760 X=4051;GOSUB C
770 X=-3721;GOSUB C
780 X=-15903;GOSUB C
790 X=-13829;GOSUB C
800 CALL20142;RETURN
810 %(A)=X;A=A+2;RETURN
>
```

```
**** 4-LINE PER INTERRUPT ****
* MODIFICATION FOR COLOR TUNNEL *

420 X=20214;GOSUB C
490 X=-508;GOSUB C
550 X=-13054;GOSUB C
610 X=22782;GOSUB C
690 X=32299;GOSUB C
700 X=2584;GOSUB C
>
```

```
********** 77 COLORS **********
* MODIFICATION FOR COLOR TUNNEL *

.FOLLOWING LINES DELETED.

430-470-590
730-740-750

&(9)=20
420 X=20212;GOSUB C
490 X=-507;GOSUB C
500 X=14515;GOSUB C
505 X=-20715;GOSUB C
510 X=1238;GOSUB C
520 X=9079;GOSUB C
525 X=-29638;GOSUB C
530 X=30542;GOSUB C
535 X=-24006;GOSUB C
540 X=-11442;GOSUB C
550 X=-13056;GOSUB C
570 X=32299;GOSUB C
580 X=9752;GOSUB C
610 X=23038;GOSUB C
670 X=211;GOSUB C
675 X=467;GOSUB C
680 X=-32314;GOSUB C
685 X=1235;GOSUB C
690 X=1491;GOSUB C
695 X=-27962;GOSUB C
700 X=723;GOSUB C
705 X=979;GOSUB C
708 X=-32058;GOSUB C
710 X=1747;GOSUB C
715 X=2003;GOSUB C
```

```
***** MOD. FOR 142 COLORS *****
ELIMINATES 1/2 SCREEN COLOR
REVERSAL COMPARE INSTRUCTION

610 X=0;GOSUB C
```

Beatle Ballys' Crash and Crunch Bar-"CALL" these locations.

&(9)=0;CALL 6683...Chicken Gunfighters
CALL 624...........Good crash
CALL 5164..........Interrupt line crash
CALL 1532...Sound off 2-P ROM
CALL 1534...Sound off 8-P ROM
CALL 1579...Trees
5455...Synth.Crash
6490...GOT ME
5308...Final Crash

```
1 .    b=m   ·ply
2 .    c=d   je
3 .
4 .ATARI LOGO
10 CLEAR ;;RETURN ;NT=0
20 A=20002;B=A;C=400
30 X=16115;GOSUB C
40 X=-4786;GOSUB C
50 X=15943;GOSUB C
60 X=-11268;GOSUB C
70 X=-1267;GOSUB C
80 X=201;GOSUB C
90 A=20220
100 X=20222;GOSUB C
110 X=-14859;GOSUB C
120 X=8677;GOSUB C
130 X=20014;GOSUB C
140 X=1094;GOSUB C
150 X=9072;GOSUB C
160 X=7387;GOSUB C
170 X=12472;GOSUB C
180 X=-13046;GOSUB C
190 X=8368;GOSUB C
200 X=15486;GOSUB C
210 X=11127;GOSUB C
220 X=30639;GOSUB C
230 X=32291;GOSUB C
240 X=-28666;GOSUB C
250 X=1747;GOSUB C
260 X=2003;GOSUB C
270 X=-6659;GOSUB C
280 X=4819;GOSUB C
290 X=-7683;GOSU
300 X=4157;GOSUB
310 X=-7693;GOSUB C
320 X=-3647;GOSUB C
330 X=-13829;GOSUB C
340 CALL20002;GOTO 610
400 %(A)=X;A=A+2;RETURN
500 BOX 0,0,10,58,1
510 FOR X=-32TO -10
520 Y=-375cX-37
530 BOX X,Y,5,7,1
540 BOX -X-1,Y,5,7,1
550 NEXT X
560 BOX -9,14,4,30,1
570 BOX 9,14,4,30,1
600 RETURN
610 GOTO 610
&(15)=22
BC=0;FC=133;RUN

SZ=1189
%(20050)=-23961
```

This program is mainly to demonstrate that the Bally do what the competition does, with more colors. Turn Knob 1 fully clockwise for movement.

&(15) is the Interrupt Line feedback register in the address chip which determines where (vertically) the interrupt starts. This can be changed to move the rainbow pattern up and down on the screen.

TO HALT INTERRUPTS :RETURN
Do not enter more than 37 bytes
while interrupts are active!

MACHINE PROGRAM LISTINGS and follow-throughs'

********** AMERICAN FLAG ********** Interrupt initialize @20280 *

```
20200*4EE8*234*EA Interrupt Vector  20202
20201*4EE9* 78*4E (next Loc.)
20202*4EEA*197*C5 PUSH BC  SAVE ENVIRONMENT
20203*4EEB*229*E5 PUSH HL
20204*4EEC*245*F5 PUSH AE
20205*4EED*  0*00 NOP
20206*4EEE* 33*21 LD HL,nn  LOAD UP ADDRESS
20207*4EEF* 46*2E n  OF SCAN LINE
20208*4EF0* 78*4E n  STORAGE
20209*4EF1*126*7E LD A,(HL)  AND Get line to Accumulator
20210*4EF2*198*C6 ADD A,n  ADD ON 14 more
20211*4EF3* 14*0E n  (width of stripes)
20212*4EF4* 71*47 LD B,A  SAVE THE LINE IN B
20213*4EF5* 35*23 INC HL  - to color store -
20214*4EF6*254*FE CP,s  IF at Bottom of
20215*4EF7*183*B7 s  Screen continue
20216*4EF8* 56*38 JRC,e  else Jump over
20217*4EF9* 10*0A e  Reset Process
20218*4EFA*205*CD CALL nn  Call interrupt processor
20219*4EFB*176*B0 n  in BASIC ROM to √ for
20220*4EFC* 32*20 n  ABORTS + Dec note timer
20221*4EFD* 62*3E LD A,n  LD A w/start color
20222*4EFE*  7*07 n  (white)
20223*4EFF*119*77 LD (HL),A  and store it
20224*4F00*175*AF XOR A  Zero out A
20225*4F01* 79*4F LD C,A  Save BLACK in C
20226*4F02* 24*18 JR e  + Goto color
20227*4F03* 11*0B e  OUTPUTS
20228*4F04*126*7E LD A,(HL)  Get present color
20229*4F05*198*C6 ADD A,n  and add on 8L (+7=Red)
20230*4F06* 81*51 n  IF over Red
20231*4F07*254*FE CP,s  now, then continue
20232*4F08* 90*5A s  thru and if not
20233*4F09* 56*38 JRC,e  you are Red
20234*4F0A*  2*02 e  so Jump
20235*4F0B*214*D6 SUB A,n - Get Back
20236*4F0C*162*A2 n  To WHITE
20237*4F0D*119*77 LD (HL),A  store present color
20238*4F0E* 79*4F LD C,A  AND SAVE IT IN C
20239*4F0F*211*D3 OUT(n),A  OUTPUT TO
20240*4F10*  0*00 (n)  STRIPES ON
20241*4F11*211*D3 OUT(n),A  RIGHT SIDE
20242*4F12*  1*01 (n)  BACKGROUND COLORS
20243*4F13*120*78 LD A,B - Get scan line -
20244*4F14*254*FE CP,s  See if your At Bottom
20245*4F15* 99*63 s  of Blue Field (&(9) changes)
20246*4F16* 56*38 JRC,e  IF so, continue
20247*4F17*  6*06 e  - else Jump to
20248*4F18*175*AF XOR A  Zero
20249*4F19*211*D3 OUT(n),A  out
20250*4F1A*  9*09 (n)  HORIZ. Boundary
20251*4F1B*121*79 LD A,C  Get stripe color
20252*4F1C* 24*18 JR e  and Jump
20253*4F1D*  2*02 e  over Blue set
20254*4F1E* 62*3E LD A,n  LD A, Blue
20255*4F1F*  8*08 n
20256*4F20*211*D3 OUT(n),A  OUTPUT either
20257*4F21*  4*04 (n)  Blue Field or stripes
20258*4F22*211*D3 OUT(n),A  TO Left SIDE
20259*4F23*  5*05 (n)  BC's
20260*4F24*120*78 LD A,B  Get Scan line again
20261*4F25*254*FE CP s  and if at Bottom
20262*4F26*183*B7 s  Continue
20263*4F27* 56*38 JRC e  else Jump
20264*4F28*  6*06 e  to
20265*4F29* 62*3E LD A,n  At Bottom
20266*4F2A* 17*11 n  so make
20267*4F2B*211*D3 OUT(n),A  &(9) = 17
20268*4F2C*  9*09 (n)  For BLUE FIELD
20269*4F2D*175*AF XOR A  Zero out A
20270*4F2E* 71*47 LD B,A  ~(NOTHIN)
20271*4F2F* 43*2B DEC HL  TO SCAN STORE
20272*4F30*119*77 LD (HL),A  STORE NEXT LINE
20273*4F31*211*D3 OUT(n),A  AND TELL THE
20274*4F32* 15*0F (n)  ADDRESS CHIP ABOUT IT TOO
20275*4F33*241*F1 POP AF
20276*4F34*225*E1 POP HL  Restore environment
20277*4F35*193*C1 POP BC
20278*4F36*251*FB EI  enable intrps.
20279*4F37*201*C9 RET  Return to BASIC
20280*4F38*243*F3 DI  NO INTRPS. NOW PLEASE  *
20281*4F39* 62*3E LD A,n  Get MSB of
20282*4F3A* 78*4E n  intrp. ROUTINE
20283*4F3B*237*ED LD I,A  TO Z-80's I
20284*4F3C* 71*47 "  Reg.
20285*4F3D* 62*3E LD A,n  + Get the
20286*4F3E*232*E8 n  LSB
20287*4F3F*211*D3 OUT(n),A  TO ADDRESS CHIP
20288*4F40* 13*0D (n)  Intrp. Feedback Reg
20289*4F41*251*FB EI  enable those intrps.
20290*4F42*201*C9 RET  Get BACK
20291*4F43*  0*00
20292*4F44*  0*00
```

ATARI LOGO INTERRUPT INITIALIZATION (Tape Buffer)

```
20000*4E20*41*29
20001*4E21*41*29
20002*4E22*243*F3 DI       Disable int. while doing this
20003*4E23* 62*3E LDA,n    Load Most
20004*4E24* 78*4E n        Significant
20005*4E25*237*ED LDI,A    Byte in Z-80
20006*4E26* 71*47 "        'I' Register (IM2)
20007*4E27* 62*3E LDA,n    Load Least significant
20008*4E28*252*FC n        Byte in
20009*4E29*211*D3 OUT(n),A Address chips
20010*4E2A* 13*0D (n)      Int. Feedback Reg.
20011*4E2B*251*FB EI       Enable again
20012*4E2C*201*C9 RET      Return to Basic
20013*4E2D*  0*00          processing
20014*4E2E*  0*00          * Holds entry counter
                           -> colon in 4E2F
```

ATARI LOGO ******(I.R.) ROUTINE (Line Buffer)

```
ADDRESS   DATA
DEC. HEX  DEC. HEX
20220*4EFC*254*FE Interrupt vector           20222
20221*4EFD* 78*4E (address of routine start)
20222*4EFE*245*F5 PUSH AF    store registers
20223*4EFF*197*C5 PUSH BC    about to be
20224*4F00*229*E5 PUSH HL    used
20225*4F01* 33*21 LDHL,nn    Load up address
20226*4F02* 46*2E n          of entry times
20227*4F03* 78*4E n          storage and
20228*4F04* 70*46 LDB,(HL)   get to B
20229*4F05*  4*04 INC B      and increment
20230*4F06*112*70 LD(HL),B   then store it back
20231*4F07* 35*23 INC HL     Point to start color
20232*4F08*219*DB IN A,(n)   get value of
20233*4F09* 28*1C (n) Pot 1  KN(1) in acc.
20234*4F0A*184*B8 CP,r "B"   IF A<B jump over A
20235*4F0B* 48*30 JRNC,e     IF A≥B Go thru A
20236*4F0C* 10*0A e          (To 20247)
20237*4F0D*205*CD CALL nn    Abort: Call norm. intrp.
20238*4F0E*176*B0 n          processor BASIC ROM 2080
20239*4F0F* 32*20 n          +check for Aborts + K.P.
20240*4F10*126*7E LDA(HL)    get start color
20241*4F11* 60*3C INC A      and inc 1 when
20242*4F12*119*77 LD(HL),A   coming thru here (store)
20243*4F13* 43*2B DEC HL     get entry count.
20244*4F14*175*AF XOR A      zero
20245*4F15*119*77 LD(HL),A   entry (reset)
20246*4F16* 35*23 INC HL     A end on color
20247*4F17*126*7E LDA,(HL)   get color (start on)
20248*4F18*  6*06 LD B,n     and get ready for
20249*4F19*144*90 n          DJNZ 90 loop (vert. Ht.)
20250*4F1A*211*D3 OUT(n),A   = loop start =
20251*4F1B*  6*06 (n)        output
20252*4F1C*211*D3 OUT(n),A   Colors
20253*4F1D*  7*07 (n)        Left FC's
20254*4F1E*253*FD PUSH IY
20255*4F1F*229*E5 "          No use except to
20256*4F20*211*D3 OUT(n),A   improve timing
20257*4F21* 18*12 (n) e(19)  (Flicker)
20258*4F22*253*FD POP IY     of screen
20259*4F23*225*E1 "
20260*4F24* 61*3D DEC A      next color
20261*4F25* 16*10 DJNZ,e     Loop on 20250
20262*4F26*243*F3 e(-13)(243) until B=0
20263*4F27*225*E1 POP HL     Restore
20264*4F28*193*C1 POP BC     environment
20265*4F29*241*F1 POP AF
20266*4F2A*251*FB EI
20267*4F2B*201*C9 RET        Go back to Basic
```

** COLOR FORMATTER **

```
LOC.              O.C.       I-initialize for intrps.
I20196*4EE4*243*F3 DI        Disable Syst. intrups ē doing this
20197*4EE5* 62*3E LDA,n      Load Acc. w/MSB of
20198*4EE6* 78*4E n          interrupt routine
20199*4EE7*237*ED LDI,A      Load it into intrup. vect.
20200*4EE8* 71*47 "          Reg. in Z-80 (2 BYTE CODE)
20201*4EE9* 62*3E LDA,n      Load Acc. w/LSB of
20202*4EEA*242*F2 n          intrp. routine and
20203*4EEB*211*D3 OUT(n),A   Load it to address chip
20204*4EEC* 13*0D (n)        Intrp. feedback reg.
20205*4EED*251*FB EI         Enable intrps. and Ret.
20206*4EEE*201*C9 RET        to Basic + wait for one
20207*4EEF*  0*00 NOP
20208*4EF0*246*F6 Light Pen interrupt vector   8438
20209*4EF1* 32*20 (in BASIC Rom) not used
20210*4EF2*244*F4 Interrupt vector            20212
20211*4EF3* 78*4E (this Routine)
*20212*4EF4*213*D5 PUSH DE   Save values of registers
20213*4EF5*245*F5 PUSH AF    to be used (on entry).
20214*4EF6*229*E5 PUSH HL    "(STORE ENVIRONMENT)"
20215*4EF7* 42*2A LDHL,(nn)  Load HL w/next
20216*4EF8* 80*50 n          Start address @ 20048
20217*4EF9* 78*4E n          to find next 5 index values.
20218*4EFA*126*7E LDA,(HL)   get first value
20219*4EFB*211*D3 OUT(n),A   to Horiz.
20220*4EFC*  9*09 (n) e(9)   boundary
20221*4EFD* 35*23 INC HL     to nxt. BYTE (VAL-2)
20222*4EFE*126*7E LDA,(HL)   get first color
20223*4EFF*211*D3 OUT(n),A   and Output
20224*4F00*  0*00 (n) e(0)   to right
20225*4F01*211*D3 OUT(n),A   side BC's

LOC.              O.C.       * Intrp. routine
20226*4F02*  1*01 (n)        - port e(1) -
20227*4F03* 35*23 INC HL     to next Byte (VAL-3)
20228*4F04*126*7E LDA,(HL)   get second
20229*4F05*211*D3 OUT(n),A   color
20230*4F06*  4*04 (n) e(4)   to left
20231*4F07*211*D3 OUT(n),A   side
20232*4F08*  5*05 (n) e(5)   BC's
20233*4F09* 35*23 INC HL     to next Byte (VAL-4)
20234*4F0A*126*7E LDA,(HL)   get third
20235*4F0B*211*D3 OUT(n),A   color
20236*4F0C*  2*02 (n) e(2)   to Right
20237*4F0D*211*D3 OUT(n),A   side
20238*4F0E*  3*03 (n) e(3)   FC's
20239*4F0F* 35*23 INC HL     to next Byte
20240*4F10*126*7E LDA,(HL)   get fourth
20241*4F11*211*D3 OUT(n),A   color
20242*4F12*  6*06 (n) e(6)   to Left
20243*4F13*211*D3 OUT(n),A   side
20244*4F14*  7*07 (n) e(7)   FC's
20245*4F15* 35*23 INC HL     nxt - address now points
20246*4F16* 34*22 LD(nn),HL  to next set of 5
20247*4F17* 80*50 n          So store it @ 20048
20248*4F18* 78*4E n          to reference on next intrp.
20249*4F19* 33*21 LDHL,nn    Load up HL w/address
20250*4F1A* 32*20 n          where address 20000
20251*4F1B* 78*4E n          of print scan line is stored
20252*4F1C* 52*34 INC(HL)    Inc. the value there (LSB)
20253*4F1D* 42*2A LDHL,(nn)  and
20254*4F1E* 32*20 n          pick it up (both) 20000
20255*4F1F* 78*4E n          (address of next scan line)

LOC.              O.C.
20256*4F20*126*7E LDA,(HL)   get next scan line and
20257*4F21*254*FE CP,n       Subtract from Acc. (compare)
20258*4F22*249*F9 n          If over 249 (end)
20259*4F23* 56*38 JRC,e      Continue thru reset
20260*4F24* 18*12 e          scan - else jump to done
20261*4F25* 33*21 LDHL,nn    Load starting
20262*4F26* 33*21 n          address for        20001
20263*4F27* 78*4E n          scan line pointer
20264*4F28* 34*22 LD(nn),HL  and
20265*4F29* 32*20 n          store it @         20000
20266*4F2A* 78*4E n          (reference)
20267*4F2B* 33*21 LDHL,nn    Load start
20268*4F2C* 62*3E n          address of         20286
20269*4F2D* 79*4F n          color index
20270*4F2E* 34*22 LD(nn),HL  and
20271*4F2F* 80*50 n          store @            20048
20272*4F30* 78*4E n          (reference)
20273*4F31*205*CD CALL nn    Call norm. intrp.
20274*4F32*176*30 n          proc. @            8368
20275*4F33* 32*20 n          in BASIC Rom (check for
20276*4F34*175*AF XOR A      Zero Acc.           aborts)
20277*4F35*214*D6 SUB n      make it
20278*4F36*  1*01 n          255 and
20279*4F37*211*D3 OUT(n),A   tell Address chip
20280*4F38* 15*0F (n)        next line to intrp. on
20281*4F39*225*E1 POP HL     Restore
20282*4F3A*241*F1 POP AF     ENVIRONMENT
20283*4F3B*209*D1 POP DE
20284*4F3C*251*FB EI         Enable interrupts and
20285*4F3D*201*C9 RET        to Basic prog.
```

41

```
********** COLOR TUNNEL **********
INTERRUPT INITIALIZATION (calc. intfce. area)
20142*4EAE*243*F3 DI     Disable intres C doing this
20143*4EAF* 62*3E LD A,n  Load MSB to Acc.
20144*4EB0* 78*4E n       (interrupt page)
20145*4EB1*237*ED LD I,A  LD Z-80 w/MSB
20146*4EB2* 71*47 "       (2 BYTE OPCODE)
20147*4EB3* 62*3E LD A,n  LD A w/ LSB of
20148*4EB4*242*F2 n       interrupt routine
20149*4EB5*211*D3 OUT(n),A  Address chip port 13
20150*4EB6* 13*0D (n)     w/A (interrupt feedback)
20151*4EB7*251*FB EI      enable interrupts again
20152*4EB8*201*C9 RET     Return to BASIC
20153*4EB9*  0*00

(MAINLINE
 INTERRUPT)
20208*4EF0*246*F6 Light Pen interrupt vector 8438
20209*4EF1* 32*20         (BASIC ROM)
20210*4EF2*246*F6 Interrupt Vector      20214
20211*4EF3* 78*4E         (this routine)
20212*4EF4*  0*00 NOP
20213*4EF5*  0*00 NOP
20214*4EF6*197*C5 PUSH BC   Save Registers
20215*4EF7*229*E5 PUSH HL   to be used
20216*4EF8*245*F5 PUSH AF
20217*4EF9* 33*21 LD HL,nn  LD HL w/ADDRESS OF
20218*4EFA*186*BA n         where scan line is
20219*4EFB* 78*4E n
20220*4EFC* 35*23 INC HL    (TIMING
20221*4EFD* 43*2B DEC HL     ONLY)
20222*4EFE*126*7E LD A,(HL)  Get present line to Acc.
20223*4EFF*198*C6 ADD A,n   ADD on 14 more
20224*4F00* 14*0E n         lines, for next int.
20225*4F01*254*FE CP,n      Compare by subtraction
20226*4F02*179*B3 n         from Acc. for bott. of
20227*4F03* 56*38 JR C,e    screen (179)+] if over
20228*4F04* 15*0F e         continue - else Jump 15
20229*4F05* 58*3A LD A,(nn)  LDA w/ register "P"
20230*4F06*140*8C n         which is starting color
20231*4F07* 78*4E n         for top of screen
20232*4F08* 35*23 INC HL    Inc. ADDRESS TO color store
20233*4F09*119*77 LD (HL),A  store color
20234*4F0A*175*AF XOR A     Zero out Acc. then
20235*4F0B*214*D6 SUB A,n   subtract 8 lines to
20236*4F0C*  8*08 n         start slightly above graphics
20237*4F0D*205*CD CALL nn   CALL normal interrupt
20238*4F0E*176*B0 n         process in BASIC ROM to
20239*4F0F* 32*20 n         check for ABORTS etc.
20240*4F10* 43*2B DEC HL    ADDRESS TO SCAN STORAGE
20241*4F11*  0*00 NOP
20242*4F12* 24*18 JR e      Jump to "DONE" (20278)
20243*4F13* 34*22 e         w/ reset scan line in A
20244*4F14* 35*23 INC HL    (NOP)
20245*4F15* 43*2B DEC HL
20246*4F16*119*77 LD (HL),A  Store next scan line
20247*4F17* 35*23 INC HL    Get to color
20248*4F18*254*FE CP,n      See if your 1/2 way
20249*4F19* 99*63 n         down screen
20250*4F1A*126*7E LD A,(HL)  Get color in Acc.
20251*4F1B* 56*38 JR C,e    If 1/2 way continue
20252*4F1C*  4*04 e         for color reversal else
20253*4F1D*214*D6 SUB A,n   subtract variable
20254*4F1E*  8*08 n         for color reverse (starts
20255*4F1F* 24*18 JR e      w/8) and Jump over
20256*4F20*  2*02 e         the ADD
20257*4F21*198*C6 ADD A,n   ADD on variable
20258*4F22*  8*08 n         (before 1/2 way)
20259*4F23*119*77 LD (HL),A  store new color
20260*4F24*211*D3 OUT(n),A  Output color to
20261*4F25*  7*07 (n)      Left
20262*4F26*211*D3 OUT(n),A  BC's
20263*4F27*  6*06 (n)
20264*4F28*198*C6 ADD A,n   ADD on 4 intensities
20265*4F29*  4*04 n
20266*4F2A*211*D3 OUT(n),A  OUTPUT color to
20267*4F2B*  5*05 (n)      Left
20268*4F2C*211*D3 OUT(n),A  FC's
20269*4F2D*  4*04 (n)
20270*4F2E* 43*2B DEC HL    Back to scan store
20271*4F2F*126*7E LD A,(HL)  Get the line in A
20272*4F30*254*FE CP,n      And if on last
20273*4F31*171*AB n         line continue
20274*4F32* 56*38 JR C,e    else Jump
20275*4F33*  2*02 e         over ADD
20276*4F34*198*C6 ADD A,n   which makes last
20277*4F35*  6*06 n         line thick like top
20278*4F36*211*D3 OUT(n),A  Tell Address chip
20279*4F37* 15*0F (n)      next line to interr. on
20280*4F38*119*77 LD (HL),A  Store the reset line
20281*4F39*241*F1 POP AF
20282*4F3A*225*E1 POP HL    Restore Registers
20283*4F3B*193*C1 POP BC
20284*4F3C*251*FB EI        enable interrupts
20285*4F3D*201*C9 RET       Return to BASIC processing
```

322?0
55
2?1?
184
32
248
201
2?3

Bally Bras...(Titilating tediums)

- To correct a byte in a program in memory. Say the "?" in this line was supposed to be a "="; 620 XT0;GOSUB C - Find address of ? first. Then %(address)=(("0")x256)+("=") will correct it.
- To reset memory to "no prog." status.(Like NEW command) %(20050)=-24572 %(-24576)=-256
- If %(20050)=-23115...then%(-23120)=the last CR in prog. %(-23119)=-256 (terminator) %(-23117)=@(0)
- Single Line Decomposer...FOR A=(begin address) TO (end address); TV=%(A);D=%(A)÷ 256;RM=RM+((RM<0)x256);PRINT A,"=",RM;NEXT A (The "TV=%(A)" is optional,to display characters)
- To find the address of a line number in a prog...FOR A=-24576 TO -22777;IF%(A)# (Line Number) NEXT A...When it stops print A.This is address of that Line number. PRINT %(A) to see it.
- To hide the lines of a program while loading %(20076)=0 Then after load %(20076)=264. (or 8)
- To find number of string locations available..PRINT SZ÷2 %(20134)

```
*****************************************************************************************************************
*                              Z-80 INSTRUCTION SET IN DECIMAL FOR BALLY (HVGDC 1.0)                            *
*****************************************************************************************************************
HEX DEC.INSTR.        HEX DEC.INSTR.        HEX DEC.INSTR.        HEX DEC.INSTR.        HEX DEC.INSTR.        HEX DEC.INSTR.
00--  0-NOP           2B-- 43-DEC HL        56-- 86-LD D,(HL)     81--129-ADD A,C       AC--172-XOR H         D7--215-RST 10H
01--  1-LD BC,nn      2C-- 44-INC L         57-- 87-LD D,A        82--130-ADD A,D       AD--173-XOR L         D8--216-RET C
02--  2-LD(BC),A      2D-- 45-DEC L         58-- 88-LD E,B        83--131-ADD A,E       AE--174-XOR(HL)       D9--217-EXX
03--  3-INC BC        2E-- 46-LD L,n        59-- 89-LD E,C        84--132-ADD A,H       AF--175-XOR A         DA--218-JPC,nn
04--  4-INC B         2F-- 47-CPL           5A-- 90-LD E,D        85--133-ADD A,L       B0--176-OR B          DB--219-IN A,(n)
05--  5-DEC B         30-- 48-JRNC,e        5B-- 91-LD E,E        86--134-ADD A,(HL)    B1--177-OR C          DC--220-CALL C,nn
06--  6-LD B,n        31-- 49-LD SP,nn      5C-- 92-LD E,H        87--135-ADD A,A       B2--178-OR D          DD--221*2 BYT PFX*
07--  7-RLCA          32-- 50-LD(nn),A      5D-- 93-LD E,L        88--136-ADC A,B       B3--179-OR E          DE--222-SBC A,n
08--  8-EX AF,AF'     33-- 51-INC SP        5E-- 94-LD E,(HL)     89--137-ADC A,C       B4--180-OR H          DF--223-RST 18H
09--  9-ADD HL,BC     34-- 52-INC(HL)       5F-- 95-LD E,A        8A--138-ADC A,D       B5--181-OR L          E0--224-RET PO
0A-- 10-LD A,(BC)     35-- 53-DEC(HL)       60-- 96-LD H,B        8B--139-ADC A,E       B6--182-OR(HL)        E1--225-POP HL
0B-- 11-DEC BC        36-- 54-LD(HL),n      61-- 97-LD H,C        8C--140-ADC A,H       B7--183-OR A          E2--226-JP PO,nn
0C-- 12-INC C         37-- 55-SCF           62-- 98-LD H,D        8D--141-ADC A,L       B8--184-CP B          E3--227-EX(SP),HL
0D-- 13-DEC C         38-- 56-JRC,e         63-- 99-LD H,E        8E--142-ADC A,(HL)    B9--185-CP C          E4--228-CALL PO,nn
0E-- 14-LD C,n        39-- 57-ADD HL,SP     64--100-LD H,H        8F--143-ADC A,A       BA--186-CP D          E5--229-PUSH HL
0F-- 15-RRCA          3A-- 58-LD A,(nn)     65--101-LD H,L        90--144-SUB B         BB--187-CP E          E6--230-AND n
10-- 16-DJNZ,e        3B-- 59-DEC SP        66--102-LD H,(HL)     91--145-SUB C         BC--188-CP H          E7--231-RST 20H
11-- 17-LD DE,nn      3C-- 60-INC A         67--103-LD H,A        92--146-SUB D         BD--189-CP L          E8--232-RET PE
12-- 18-LD(DE),A      3D-- 61-DEC A         68--104-LD L,B        93--147-SUB E         BE--190-CP(HL)        E9--233-JP(HL)
13-- 19-INC DE        3E-- 62-LD A,n        69--105-LD L,C        94--148-SUB H         BF--191-CP A          EA--234-JP PE,nn
14-- 20-INC D         3F-- 63-CCF           6A--106-LD L,D        95--149-SUB L         C0--192-RET NZ        EB--235-EX DE,HL
15-- 21-DEC D         40-- 64-LD B,B        6B--107-LD L,E        96--150-SUB(HL)       C1--193-POP BC        EC--236-CALL PE,nn
16-- 22-LD D,n        41-- 65-LD B,C        6C--108-LD L,H        97--151-SUB A         C2--194-JPNZ,nn       ED--237*2 BYT PFX*
17-- 23-RLA           42-- 66-LD B,D        6D--109-LD L,L        98--152-SBC A,B       C3--195-JP,nn         EE--238-XOR n
18-- 24-JR,e          43-- 67-LD B,E        6E--110-LD L,(HL)     99--153-SBC A,C       C4--196-CALL NZ,nn    EF--239-RST 28H
19-- 25-ADD HL,DE     44-- 68-LD B,H        6F--111-LD L,A        9A--154-SBC A,D       C5--197-PUSH BC       F0--240-RET P
1A-- 26-LD A,(DE)     45-- 69-LD B,L        70--112-LD(HL),B      9B--155-SBC A,E       C6--198-ADD A,n       F1--241-POP AF
1B-- 27-DEC DE        46-- 70-LD B,(HL)     71--113-LD(HL),C      9C--156-SBC A,H       C7--199-RST 0         F2--242-JP P,nn
1C-- 28-INC E         47-- 71-LD B,A        72--114-LD(HL),D      9D--157-SBC A,L       C8--200-RET Z         F3--243-DI
1D-- 29-DEC E         48-- 72-LD C,B        73--115-LD(HL),E      9E--158-SBC A,(HL)    C9--201-RET           F4--244-CALL P,nn
1E-- 30-LD E,n        49-- 73-LD C,C        74--116-LD(HL),H      9F--159-SBC A,A       CA--202-JPZ,nn        F5--245-PUSH AF
1F-- 31-RRA           4A-- 74-LD C,D        75--117-LD(HL),L      A0--160-AND B         CB--203*2 BYT PFX*    F6--246-OR n
20-- 32-JR NZ,e       4B-- 75-LD C,E        76--118-HALT          A1--161-AND C         CC--204-CALL Z,nn     F7--247-RST 30H
21-- 33-LD HL,nn      4C-- 76-LD C,H        77--119-LD(HL),A      A2--162-AND D         CD--205-CALL nn       F8--248-RET M
22-- 34-LD(nn),HL     4D-- 77-LD C,L        78--120-LD A,B        A3--163-AND E         CE--206-ADC A,n       F9--249-LD SP,HL
23-- 35-INC HL        4E-- 78-LD C,(HL)     79--121-LD A,C        A4--164-AND H         CF--207-RST 8         FA--250-JP M,nn
24-- 36-INC H         4F-- 79-LD C,A        7A--122-LD A,D        A5--165-AND L         D0--208-RET NC        FB--251-EI
25-- 37-DEC H         50-- 80-LD D,B        7B--123-LD A,E        A6--166-AND(HL)       D1--209-POP DE        FC--252-CALL M,nn
26-- 38-LD H,n        51-- 81-LD D,C        7C--124-LD A,H        A7--167-AND A         D2--210-JPNC,nn       FD--253*2 BYT PFX*
27-- 39-DAA           52-- 82-LD D,D        7D--125-LD A,L        A8--168-XOR B         D3--211-OUT(n),A      FE--254-CP n
28-- 40-JRZ,e         53-- 83-LD D,E        7E--126-LD A,(HL)     A9--169-XOR C         D4--212-CALL NC,nn    FF--255-RST 38H
29-- 41-ADD HL,HL     54-- 84-LD D,H        7F--127-LD A,A        AA--170-XOR D         D5--213-PUSH DE
2A-- 42-LD HL,(nn)    55-- 85-LD D,L        80--128-ADD A,B       AB--171-XOR E         D6--214-SUB n
```

```
*************************************************************************************************************
*      Z-80 INSTRUCTION SET CONT.        2-BYTE OPCODES        DECIMAL PREFIX & CODE FOR BALLY   (HVGDC 1.0)  *
*************************************************************************************************************
```

~~~~~~~~~~~~~~~~~~~~PREFIX CB --- DECIMAL 203~~~~~~~~~~~~~~~~~~~~~~

HEX DEC. INSTR.	HEX DEC.INSTR.	HEX DEC.INSTR.	HEX DEC.INSTR.	HEX DEC.INSTR.	HEX DEC.INSTR.
00-- 0-RLC B	2A-- 42-SRA D	5C-- 92-BIT 3,H	86--134-RES 0,(HL)	B0--176-RES 6,B	DA--218-SET 3,D
01-- 1-RLC C	2B-- 43-SRA E	5D-- 93-BIT 3,L	87--135-RES 0,A	B1--177-RES 6,C	DB--219-SET 3,E
02-- 2-RLC D	2C-- 44-SRA H	5E-- 94-BIT 3,(HL)	88--136-RES 1,B	B2--178-RES 6,D	DC--220-SET 3,H
03-- 3-RLC E	2D-- 45-SRA L	5F-- 95-BIT 3,A	89--137-RES 1,C	B3--179-RES 6,E	DD--221-SET 3,L
04-- 4-RLC H	2E-- 46-SRA(HL)	60-- 96-BIT 4,B	8A--138-RES 1,D	B4--180-RES 6,H	DE--222-SET 3,(HL)
05-- 5-RLC L	2F-- 47-SRA A	61-- 97-BIT 4,C	8B--139-RES 1,E	B5--181-RES 6,L	DF--223-SET 3,A
06-- 6-RLC(HL)	38-- 56-SRL B	62-- 98-BIT 4,D	8C--140-RES 1,H	B6--182-RES 6,(HL)	E0--224-SET 4,B
07-- 7-RLC A	39-- 57-SRL C	63-- 99-BIT 4,E	8D--141-RES 1,L	B7--183-RES 6,A	E1--225-SET 4,C
08-- 8-RRC B	3A-- 58-SRL D	64--100-BIT 4,H	8E--142-RES 1,(HL)	B8--184-RES 7,B	E2--226-SET 4,D
09-- 9-RRC C	3B-- 59-SRL E	65--101-BIT 4,L	8F--143-RES 1,A	B9--185-RES 7,C	E3--227-SET 4,E
0A-- 10-RRC D	3C-- 60-SRL H	66--102-BIT 4,(HL)	90--144-RES 2,B	BA--186-RES 7,D	E4--228-SET 4,H
0B-- 11-RRC E	3D-- 61-SRL L	67--103-BIT 4,A	91--145-RES 2,C	BB--187-RES 7,E	E5--229-SET 4,L
0C-- 12-RRC H	3E-- 62-SRL(HL)	68--104-BIT 5,B	92--146-RES 2,D	BC--188-RES 7,H	E6--230-SET 4,(HL)
0D-- 13-RRC L	3F-- 63-SRL A	69--105-BIT 5,C	93--147-RES 2,E	BD--189-RES 7,L	E7--231-SET 4,A
0E-- 14-RRC(HL)	40-- 64-BIT 0,B	6A--106-BIT 5,D	94--148-RES 2,H	BE--190-RES 7,(HL)	E8--232-SET 5,B
0F-- 15-RRC A	41-- 65-BIT 0,C	6B--107-BIT 5,E	95--149-RES 2,L	BF--191-RES 7,A	E9--233-SET 5,C
10-- 16-RL B	42-- 66-BIT 0,D	6C--108-BIT 5,H	96--150-RES 2,(HL)	C0--192-SET 0,B	EA--234-SET 5,D
11-- 17-RL C	43-- 67-BIT 0,E	6D--109-BIT 5,L	97--151-RES 2,A	C1--193-SET 0,C	EB--235-SET 5,E
12-- 18-RL D	44-- 68-BIT 0,H	6E--110-BIT 5,(HL)	98--152-RES 3,B	C2--194-SET 0,D	EC--236-SET 5,H
13-- 19-RL E	45-- 69-BIT 0,L	6F--111-BIT 5,A	99--153-RES 3,C	C3--195-SET 0,E	ED--237-SET 5,L
14-- 20-RL H	46-- 70-BIT 0,(HL)	70--112-BIT 6,B	9A--154-RES 3,D	C4--196-SET 0,H	EE--238-SET 5,(HL)
15-- 21-RL L	47-- 71-BIT 0,A	71--113-BIT 6,C	9B--155-RES 3,E	C5--197-SET 0,L	EF--239-SET 5,A
16-- 22-RL(HL)	48-- 72-BIT 1,B	72--114-BIT 6,D	9C--156-RES 3,H	C6--198-SET 0,(HL)	F0--240-SET 6,B
17-- 23-RL A	49-- 73-BIT 1,C	73--115-BIT 6,E	9D--157-RES 3,L	C7--199-SET 0,A	F1--241-SET 6,C
18-- 24-RR B	4A-- 74-BIT 1,D	74--116-BIT 6,H	9E--158-RES 3,(HL)	C8--200-SET 1,B	F2--242-SET 6,D
19-- 25-RR C	4B-- 75-BIT 1,E	75--117-BIT 6,L	9F--159-RES 3,A	C9--201-SET 1,C	F3--243-SET 6,E
1A-- 26-RR D	4C-- 76-BIT 1,H	76--118-BIT 6,(HL)	A0--160-RES 4,B	CA--202-SET 1,D	F4--244-SET 6,H
1B-- 27-RR E	4D-- 77-BIT 1,L	77--119-BIT 6,A	A1--161-RES 4,C	CB--203-SET 1,E	F5--245-SET 6,L
1C-- 28-RR H	4E-- 78-BIT 1,(HL)	78--120-BIT 7,B	A2--162-RES 4,D	CC--204-SET 1,H	F6--246-SET 6,(HL)
1D-- 29-RR L	4F-- 79-BIT 1,A	79--121-BIT 7,C	A3--163-RES 4,E	CD--205-SET 1,L	F7--247-SET 6,A
1E-- 30-RR(HL)	50-- 80-BIT 2,B	7A--122-BIT 7,D	A4--164-RES 4,H	CE--206-SET 1,(HL)	F8--248-SET 7,B
1F-- 31-RR A	51-- 81-BIT 2,C	7B--123-BIT 7,E	A5--165-RES 4,L	CF--207-SET 1,A	F9--249-SET 7,C
20-- 32-SLA B	52-- 82-BIT 2,D	7C--124-BIT 7,H	A6--166-RES 4,(HL)	D0--208-SET 2,B	FA--250-SET 7,D
21-- 33-SLA C	53-- 83-BIT 2,E	7D--125-BIT 7,L	A7--167-RES 4,A	D1--209-SET 2,C	FB--251-SET 7,E
22-- 34-SLA D	54-- 84-BIT 2,H	7E--126-BIT 7,(HL)	A8--168-RES 5,B	D2--210-SET 2,D	FC--252-SET 7,H
23-- 35-SLA E	55-- 85-BIT 2,L	7F--127-BIT 7,A	A9--169-RES 5,C	D3--211-SET 2,E	FD--253-SET 7,L
24-- 36-SLA H	56-- 86-BIT 2,(HL)	80--128-RES 0,B	AA--170-RES 5,D	D4--212-SET 2,H	FE--254-SET 7,(HL)
25-- 37-SLA L	57-- 87-BIT 2,A	81--129-RES 0,C	AB--171-RES 5,E	D5--213-SET 2,L	FF--255-SET 7,A
26-- 38-SLA(HL)	58-- 88-BIT 3,B	82--130-RES 0,D	AC--172-RES 5,H	D6--214-SET 2,(HL)	
27-- 39-SLA A	59-- 89-BIT 3,C	83--131-RES 0,E	AD--173-RES 5,L	D7--215-SET 2,A	
28-- 40-SRA B	5A-- 90-BIT 3,D	84--132-RES 0,H	AE--174-RES 5,(HL)	D8--216-SET 3,B	
29-- 41-SRA C	5B-- 91-BIT 3,E	85--133-RES 0,L	AF--175-RES 5,A	D9--217-SET 3,C	
~~~~~~~~~~~~~~~~~~~~