

BOOT CAMP

by Tom Hudson

With this issue, A.N.A.L.O.G. begins a new column. "Boot Camp" will examine assembly language on the ATARI computer systems, while presenting useful subroutines to illustrate the techniques discussed in the column. Readers are invited to send topic suggestions or questions to BOOT CAMP, A.N.A.L.O.G. Computing, P.O. Box 23, Worcester, MA 01603.

The ground rules.

Before starting to learn assembly language, let's lay the ground rules.

First, you should have a good reference guide to assembly language operation codes. I suggest 6502 *Assembly Language Programming* by Lance Leventhal (OSBORNE/McGraw-Hill). Of course, there are many such books, and the final choice of what book to use is up to you. Just be sure it covers the 6502 operation codes clearly and completely.

Second, since many program concepts will be shown in BASIC, you should have a working knowledge of BASIC. Assembly language requires a solid background in programming logic, and working in BASIC helps develop this skill. In addition, assembly programming concepts can be grasped more easily if they are first shown in a language the reader is familiar with, such as BASIC. This should not be a problem for most readers, since BASIC is usually the first language learned by personal computer owners. Therefore, from this point on, I will assume that all readers of this column are fluent in BASIC.

Third, you will need an assembler/editor package. All assembly language listings in this column will be compatible with the ATARI **Assembler/Editor** cartridge and OSS's **EASMD** and **MAC/65** assemblers. You can use other assemblers, but some code conversion may be necessary.

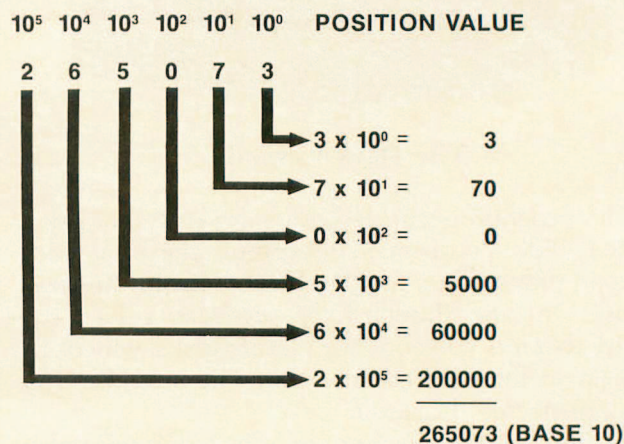
Fourth, you should be able to read flowcharts. Flowcharts are a good way to visualize a program's operation before actually writing any code.

Numbering systems.

Everybody is familiar with the DECIMAL numbering system. We all use this numbering system in everyday mathematical calculations. The word "decimal" is derived from the Latin *decem*, or ten. Therefore, this numbering system is known as "base 10," since there are ten digits, 0-9. Let's take a closer look at the decimal numbering system.

Figure 1 shows how a six digit base 10 number can be broken down into individual digits. Each digit can range from 0-9 in value.

Figure 1.



Above each digit is that digit's POSITION VALUE. The position value is the amount each digit is multiplied by to get the actual value of the digit. You will notice that the position values are shown in powers of ten, since we are working in base 10. The ones position is shown as 10 to the zero power. Any time a number is raised to the zero power, the result is ONE. Therefore, to get the value of the 3 in the last position of the number, we would calculate:

$$\text{DIGIT} \times \text{POSITION VALUE} = \text{VALUE}$$

In this case the calculation would be:

$$3 \times 1 = 3$$

We would conclude that the last position in the number would have a value of 3.

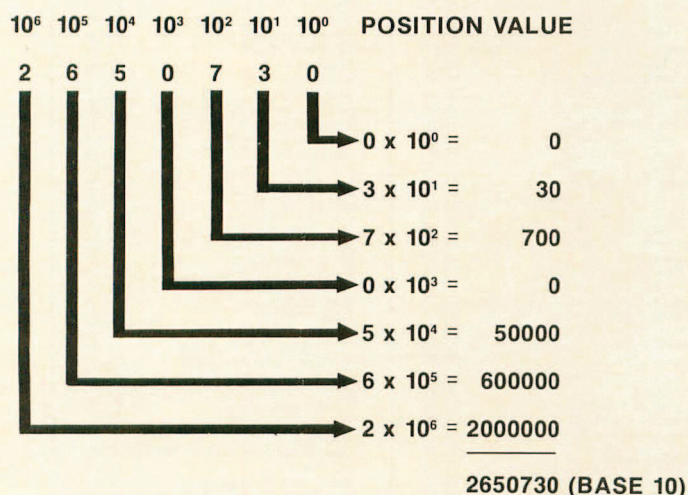
The next position, containing the digit 7, has a position value of 10 to the first power, or 10. The calculation of this digit's value would be:

$$7 \times 10 = 70$$

When we repeat this calculation for each digit in the number and add all the values, we will obtain the value of the number, 265,073 (base 10).

Here's another concept that we may not think about, but is very interesting. What happens to the number if we shift all the digits to the left, as shown in Figure 2?

Figure 2.



By looking at the final results, you can see that the number has effectively been multiplied by 10, with a result of 2,650,730 (base 10)!

Why did this happen? The answer is actually very simple. When each digit is shifted to the left, its position value is increased by a power of ten. The resulting number will be ten times larger than if it were not shifted. Try shifting the number to the right and see what happens.

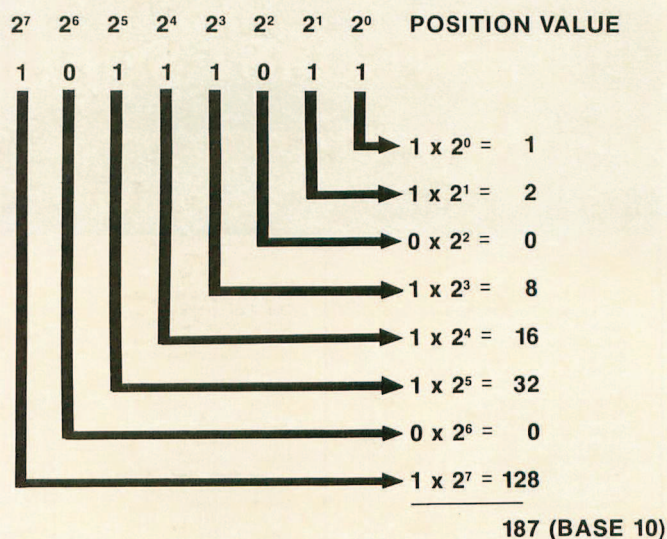
What do we care about all this?

Now that we know exactly how our normal numbering system works, let's apply what we know to a different system, BINARY.

The word "binary" comes from the Latin *bis*, or "double." As you may know, digital computers work with two electrical states, ON and OFF. This situation is perfectly suited for the binary numbering system, or base 2.

The binary numbering system uses only two digits, 0 and 1, but the principle of the numbering system is the same as base 10. Figure 3 shows a number in base 2 and how it can be converted to base 10.

Figure 3.



Once again, the number is shown with the position values above each digit in the number. In base 2, you will notice that the position values are

(continued on next page)

Atari owners, are you backed up?

At last, an inexpensive and easy way to make archival duplicates of your boot tape software. Even simple tape stretching from normal use can suddenly make a program no longer load. BOOT TAPE BACK-UP will generate a copy of most of your autoboot cassette software to provide you with an identical copy to store away for safe keeping.

In computing, back-up is the keyword for professionals-- why be any less than professional yourself? At minimal cost, BOOT TAPE BACK-UP can protect your much larger investment in software and pay for itself many times over. Why put your precious programs at risk?

BOOT TAPE BACK-UP \$14.95

Also Available: BOOT TAPE BACK-UP PLUS
For disc owners (or future disc owners)--Same as above, but will ALSO transfer most of your cassette programs to disc as well. An even greater bargain!

BOOT TAPE BACK-UP PLUS \$18.95

To order;
Send CHECK or MONEY ORDER (include \$1.50 shipping) to:

msb SOFTWARE

P.O. Box 450 New York, NY 10024

NY State residents add sales tax

Note: Intended for personal archival purposes only!
Single stage only Minimum 32K required

Atari TM of Atari, Inc.

powers of two. This means that, unlike the decimal progression of 1, 10, 100, etc. the binary system has a progression of 1, 2, 4, 8 and so on. As a result, the number 10111011 (base 2) is 187 in base 10. Figure 4 shows the binary equivalents of the numbers 0-19. Try using the method shown in Figure 3 to convert these numbers to the base 10 equivalents shown.

Figure 4.

BASE 10	BASE 2	BASE 10	BASE 2
0	0	10	1010
1	1	11	1011
2	10	12	1100
3	11	13	1101
4	100	14	1110
5	101	15	1111
6	110	16	10000
7	111	17	10001
8	1000	18	10010
9	1001	19	10011

Remember how a base 10 number multiplied by 10 when we shifted it left one digit? Let's look at how a binary number is affected by such a shift. Figure 5a shows the number 7 in binary before the shift and Figure 5b shows the number after the shift.

Figure 5a.

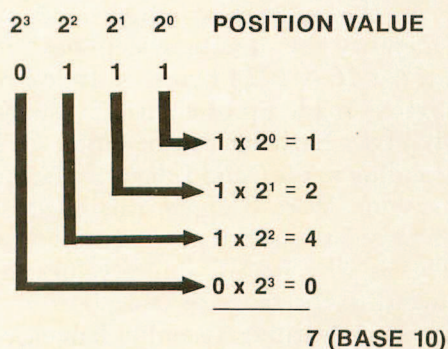
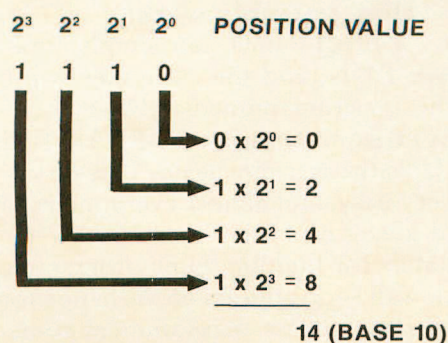


Figure 5b.



The number has been multiplied by 2! By examining this result and the above shift in base 10, we can see that by shifting the digits in a number left or right, we multiply or divide the number by its base number. This concept will come in very handy in later installments of this column, so keep it in mind.

"Funny" numbers.

The mechanics of the binary numbering system are extremely important, but they can cause some problems.

Let's say you want to look at what is in memory location 33011, but don't want to give the number in decimal for some reason. The most logical choice, as far as your computer is concerned, is binary. Unfortunately for us humans, this number comes out as 1000000011110011, and is cumbersome, to say the least. We don't like to handle numbers like this — there are just too many chances to make a mistake.

Fear not! There is yet another numbering system that is compatible with both our friend the computer and our human limitation for handling large numbers. What is this system, you ask? It is called base 16, or HEXADECIMAL.

(continued on next page)

NOW... an X-RATED program for the ATARI!

XBASIC by George Schwenk for Superware.

A low cost upgrade of ATARI® BASIC. Why buy Microsoft BASIC and lose the ease of using ATARI® BASIC with its syntax error checking when XBASIC gives you:

- Integer and String Arrays with In-String Search
- Player Missile Graphics;
- Special Graphics Modes 4-Color 160 x 192 Resolution
- Multicolor Character Modes
- Move, Clear & Fill Memory, DPOKE, DPEEK
- Vertical Blank Sound
- Delay & Timing Functions
- Disk Directory List
- Multiple Byte Fast I/O
- Screen Saves & Loads
- Requires Less Than 3K RAM
- Detailed Manual/ 6 Examples

XBASIC is saved along with your ATARI® BASIC source code and thus is ideal for tape users. All this power for only: **\$29.95**

Ask for a demonstration at your local dealer or order direct (Available as disc or cassette— Please specify)

We Accept MC, VISA, & AMX

Include all embossed information or send check or M.O.P. to: Superware

2028 Kingshouse Rd.
Silver Spring, MD 20904
(301) 236-4459

Dealer Inquiries Invited

Look for our line of FAMILY STRATEGY GAMES coming soon to a dealer near you.

* Please add \$2.00 shipping & handling. MD residents 5% sales tax.

Most A.N.A.L.O.G. readers are familiar with base 16 through the assembly-language arcade games **Fill 'Er Up** and **Livewire**. Both of the BASIC programs used to create these games contained hundreds of "funny" numbers like F7, 6A, and so on. What's the story behind hexadecimal? Read on!

We have already noted that base 10 uses ten digits, 0-9, and that base 2 uses two digits, 0-1. Naturally, then, it follows that base 16 uses sixteen digits.

But wait a minute! Since we humans normally use only the ten digits from the decimal system, we don't have enough for base 16 — we'll have to come up with six more. Rather than invent six new digit symbols, we'll use the letters A-F, which are already in existence. Figure 6 shows the 16 digits used in hexadecimal and their decimal equivalents.

Figure 6.

BASE 10	BASE 16
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

Once again, the principle of the hexadecimal (hex) numbering system is the same as the other systems we have examined so far; the only difference is that the letters A-F must be thought of as the numbers 10-15. Figure 7 shows the conversion of the hex number \$F4BE (all hex numbers should be preceded by a "\$") to decimal.

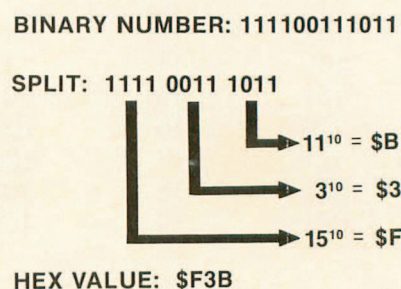
Figure 7.

	16 ³	16 ²	16 ¹	16 ⁰	POSITION VALUE
F	4	B	E		
				→ 14 x 16 ⁰ =	14
			→ 11 x 16 ¹ =		176
		→ 4 x 16 ² =			1024
→ 15 x 16 ³ =					61440
<hr/>					
62654 (BASE 10)					

So how does expressing numbers in hex help us avoid binary monstrosities? It's easy. The number 33011 (1000000011110011 in binary) is \$80F3 in hex. Obviously, this is a much easier number to remember than its binary equivalent.

Another interesting fact is that binary numbers are very easy to convert to hex. First the binary number must be divided into groups of four digits, from right to left. Then each group of four digits (ranging in value from 0-15) can easily be converted to the corresponding hex digit 0-F. Figure 8 illustrates this technique.

Figure 8.



Bytes and bits.

All readers who have owned their computers for more than a few days have at least heard of the terms "BYTE" and "BIT." Usually the term pops up when the memory capacity of the computer is being discussed.

The byte is the unit most often used when referring to memory size. If your system has "16K" of memory, it has 16 X 1024 bytes, or 16384 bytes total. Each byte is made up of eight bits (short for BINARY DIGITS). Each bit can be either OFF or ON, corresponding to the 0 and 1 digits in the binary numbering system. With 8 digits, this means that each byte can have 2 to the 8th power (or 256) combinations. This is why BASIC limits values in the POKE command to the range of 0-255.

In the process of learning assembly language, we will learn to manipulate the memory of your computer to do the things we want. Study these concepts carefully as they will be used in almost every assembly language program you write.

How assembly works.

In BASIC, a programmer can simply type in a program, type RUN, and the computer will begin executing the program immediately. If there is a problem, the programmer presses BREAK, finds the error and RUNs the program again. This makes programming very easy, and almost everyone is happy.

Yes, I said *almost* everyone.

Unfortunately for budding game programmers, a kind of brick wall soon appears on the happy road to the ultimate game. These programmers soon find

that BASIC is far too slow to handle the complex graphics and game logic necessary for an arcade-style game. At the very least, assembly-language sub-routines are necessary to speed things up.

Why is BASIC so slow? Inside the computer is a device called a 6502 MICROPROCESSOR. This little chip of silicon is what makes your computer work. It is capable of performing hundreds of thousands of operations per second, and does so every second your computer is powered on!

Sadly, all this computing power is lost as soon as a BASIC cartridge is inserted into your machine. You see, the microprocessor doesn't understand a single word of English, and the BASIC cartridge must act as an interpreter.

All of this interpreting takes time, and instead of doing the work you want, the poor microprocessor winds up spending most of its time translating BASIC into a language it can understand: binary. And this translation doesn't happen just once — it happens every time a BASIC command is executed! What a waste.

Assembly language, on the other hand, uses what is known as an ASSEMBLER to perform this translation just once. The programmer writes a program in a special format. This is known as the SOURCE code. When ready to execute the program, the programmer processes it with an assembler, which translates the source code into OBJECT code, which is the actual binary machine language. This code can be loaded at any time and executed as fast as the computer can go. It only has to be assembled once.

There are a few tradeoffs involved when using assembly language, however.

First, the programmer must re-assemble a program each time a change is made. This can take quite a bit of time when a large program is involved. For this reason, it is a good idea to flowchart each program before writing any code. This helps reduce logic errors.

Second, the programmer must know where the program will be located in memory. Since the computer's operating system has certain needs, the programmer must be aware of what memory locations are available.

Third, errors can be hard to find. When a program is executing at hundreds of thousands of operations per second, an error cannot always be easily traced to a certain instruction. For this reason, a good debugging package is a must.

Fourth, all arithmetic must be handled explicitly by the programmer. Assembly language does not have square root, sine or cosine functions. It cannot multiply or even divide! Unless the programmer specifies otherwise, the addition and subtraction instructions can only produce numbers from -128 to 127. In the course of this column, we will examine the arithmetic functions that are possible in assembly language and how they are coded.

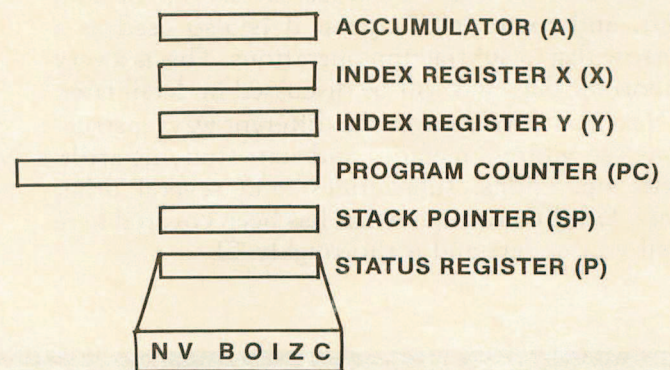
This may sound like a lot of limitations, but the 6502 processor allows the programmer to use the computer's built-in operating system directly, which BASIC has a hard time doing. And, of course, assembly language can be thousands of times faster than BASIC, allowing the programmer to write real-time simulations and arcade-style games.

Now that we've laid the groundwork for assembly-language programming, let's look at the 6502 itself.

Chip off the old block.

The 6502 processor chip has six REGISTERS that we are concerned with. These registers hold specific information and provide work areas for the programmer, and are shown in Figure 9.

Figure 9.



The ACCUMULATOR (A) is the most important register as far as the programmer is concerned. This register is used for all arithmetic operations and most data manipulation. The accumulator is used more than any other.

The INDEX REGISTERS (X and Y) are used to hold memory indexes, counters, or offsets into tables. They can also be used as temporary storage areas.

The PROGRAM COUNTER (PC) is used by the 6502 to keep track of what instruction is being executed. This register is 16 bits long, enabling it to point to any byte in memory (up to 65535, or 64K). Since this register is maintained by the 6502, we will not be referencing it very often.

The STACK POINTER (SP) is used by the 6502 to keep track of a temporary storage region known as the STACK. The stack holds subroutine return addresses and other temporary data. Since this register is maintained by the 6502, we will not be referencing it very often.

The PROCESSOR STATUS REGISTER (P) is made up of 7 individual "flags," or indicators, which inform the programmer of the 6502's current status.

The SIGN flag (N) is ZERO when the result of an operation is positive, and ONE when the result is negative.

The OVERFLOW flag (V) is set to the EXCLUSIVE-OR of bits 6 and 7 of the result of an arithmetic operation. The exclusive-or will be result in a TRUE result if either bit being evaluated is TRUE, but not if both are TRUE. The overflow flag is rarely used, and is not important at this point.

The BREAK flag (B) is set to 1 when a BRK instruction is executed. We will be using the instruction during program testing to stop program execution.

The DECIMAL MODE (D) flag is used to tell the processor to use either binary (0) or binary-coded decimal (1) arithmetic. This flag is important, and the programmer must be aware of its setting at all times.

The INTERRUPT flag (I) enables or disables system interrupts, depending on its setting.

The ZERO flag (Z) is set to 1 when any arithmetic or logical operation produces a zero result. A non-zero result sets the flag to 0.

The CARRY flag (C) holds carries out of add, shift, and rotate instructions. It is also used as a borrow flag in subtraction operations. This is a very important flag, and will be discussed in detail later.

Next issue, we'll cover the different ways instructions can address memory and start studying arithmetic operations, subroutines, and several other areas. Until then, study what has been covered here until you understand it thoroughly. □



FIRST BORN IN 1978!

the original & continuously updated

CCA Data Management System

Now Available For Atari Computers **\$ 99.50**
For Apple Computers **150.00**
For CPM Based Computers **225.00**

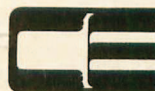
CCA Data Management System

Uses

- Business
 - Accounts Receivable
 - Accounts Payable
 - Inventories
 - Billing
 - Lists and Rosters
- Home Phone Lists
- Budgets, Hobbies

Features And Capabilities

- Long record lengths
- Up to 24 fields per record
- **Not Copy Guarded**
- Alpha numeric items
- Numeric only items
- Add, update, scan, etc. files
- 10-Level sort ascending, descending, allows alphabetizing data file.
- Contact your local dealer for details or write us for our catalog



DIVISION OF CUSTOM ELECTRONICS, INC.
SOFTWARE

238 Exchange St., Chicopee, Massachusetts 01013
(413) 592-4761

Mastercard & VISA Accepted

• Dealer And Distributor Inquiries Invited

• Closed Mondays — Open Daily 'Til 5:30 — Fridays 'Til 8

VERY LOW PRICES GET YOUR ATTENTION VERY GOOD SERVICE KEEPS IT

AT RCE WE NOT ONLY PROVIDE OUR PATRONS WITH LOW PRICES . . . WE BACK THEM WITH SUPPORT!!! FACTORY AUTHORIZED SERVICE CENTER SUPPORT FOR OVER TWENTY DIFFERENT BRANDS OF HOME ELECTRONICS INCLUDING . . . ATARI, FOURTH DIMENSION, MICRO-SCI, SANYO, FRANKLIN, PANASONIC AND U.S. PIONEER. APPLE WARRANTY SERVICE AVAILABLE.

FACTORY AUTHORIZED SERVICE COMBINED WITH PRICES LIKE THESE:

ATARI HARDWARE		MONITORS		PRINTERS		RCE COMMANDER 2400
	LIST RCE		GREEN SCREEN		STAR MICRONICS	SPECIFY 400 or 800 version
600XL COMPUTER . . .	\$199 \$CALL	BMC 12"	\$89	GEMINI 10	\$CALL	2400 - 1 \$199
800XL COMPUTER . . .	\$299 \$CALL	ZENITH 12"	\$99	GEMINI 15	\$CALL	2400 - 2 \$169
1400XL COMPUTER . .	\$599 \$CALL	SANYO 12"	\$205	MODEMS		OUR PRICES ARE ALWAYS GOING DOWN CALL FOR LATEST REDUCED PRICE
1450XL COMPUTER . .	\$999 \$CALL	BLACK & WHITE		HAYES SMARTMODEM 300 . .	\$209	
1010 RECORDER	\$100 \$75	SANYO 9"	\$145	MICROBITS MPP 1000	\$169	
810 DISK DRIVE	\$599 \$429	SANYO 12"	\$189	INTERFACES		
850 INTERFACE	\$219 \$165	AMBER SCREEN		MICROBITS MPP 1100	\$89	
1020 PRINTER	\$299 \$239	ZENITH 12"	\$115	DISK DRIVES		
1025 PRINTER	\$549 \$439	AMDEK 12"	\$179	RANA	\$319	

SEND FOR RCE'S FREE ATARI HARDWARE AND SOFTWARE CATALOG

WE ALSO CARRY A FULL LINE OF APPLE/FRANKLIN AND IBM COMPATIBLE SOFTWARE!!



**ORDER TOLL-FREE
800-547-2492**

IN OREGON: (503) 479-4711

RALSTON CLEARWATERS ELECTRONICS
536 N.E. 'E' STREET GRANTS PASS, OR 97526

ALL BRANDS ARE REGISTERED TRADE MARKS

FOR CUSTOMER SERVICE CALL: (503) 479-4711 or (503) 479-4150

TERMS:
SHIPPING: Add 6% of total transaction for UPS brown (ground) or 9% for UPS blue (air), Parcel Post, or any special arrangements. Minimum shipping charge - \$6.00
PAYMENT: Cashier's checks, certified checks, money orders, and bank wires honored immediately. Visa & Master Charge accepted. Allow 20 days for personal checks to clear.
REFUNDS: 10% restocking charge on all returns or exchanges. No refunds on opened software. Call first.
GUARANTEE: All products with full manufacturer's warranty. Sanyo and Apple warranty available.
We have full repair and service facilities for all electronic repairs with HP, Dynascan, Pioneer, Sanyo and Apple trained and certified technicians. For any technical service call them for instant advice or questions right on their benches at (503) 479-4150.
REPAIRS: Call for details on quality guaranteed discount repair and reconditioning service.
We have been repairing electronic equipment for 12 years and love it!